UNIVERSITAT JAUME I

ESCOLA SUPERIOR DE TECNOLOGIA I CIÈNCIES EXPERIMENTALS

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Sistema de control automático de una planta de laboratorio basada en 2 depósitos

TRABAJO FINAL DE GRADO

AUTOR

Aitor Roig Alemany

DIRECTOR

Roberto Sanchis Llopis

Castellón, Septiembre de 2015

Agradecimientos

En primer lugar, quiero dar las gracias al señor Ignacio Morell, que, cuando contaba yo con solo quince años, en su concurso "*El Gusto por Investigar*", me abrió las puertas de la UJI de par en par.

Agradezco al técnico, que en su momento, hizo la interconexión de los sistemas de doble depósito.

Mi agradecimiento para todos los profesores de la carrera, por haberme enseñado tanto.

Muchísimas gracias al señor Roberto Sanchis, por su gran ayuda, por su disponibilidad, y por no mirar el reloj en momentos clave, es admirable.

Quiero dar las gracias a mi compañera de viaje, Anna, por su paciencia y apoyo.

Pero a quienes más tengo que agradecer la realización del presente proyecto, es a mi cariñosa hermana, Amanda, que con sus constantes muestras de cariño, me levanta la moral, y a mis padres que siempre están disponibles a cualquier hora, para escuchar, para aconsejar, para animarme, apoyarme, alegrarse…

¡GRACIAS A TODOS!

Resumen

El objetivo del presente Trabajo Final de Grado es el desarrollo del sistema de control automático de una planta de laboratorio basada en 2 depósitos.

Dicha planta puede configurarse para implementar sistemas de control de una entrada y una salida, o de varias entradas y varias salidas, en función de la medición de los diversos niveles y caudales.

Las tareas realizadas han sido:

- Determinación de la configuración hidráulica de la planta (conexión entre depósitos).
- Selección de la electrónica necesaria, tanto de potencia para las bombas, como para la lectura de señales de los sensores.
- Selección y configuración de un sensor de nivel por ultrasonidos.
- Programación del firmware de la tarjeta de microcontrolador (de la familia Arduino) para el procesado de las señales y la comunicación con el PC.
- Programación de la aplicación de PC en Matlab, para implementar los distintos algoritmos de control y la comunicación con la tarjeta de microcontrolador.

El entorno desarrollado en Matlab es flexible para permitir la implementación de algoritmos de control diversos, permitiendo su fácil modificación y uso, tanto para personas expertas como gente que no lo es en el uso de dicha herramienta.

Se han desarrollado con detalle, a modo de ejemplo práctico, los siguientes algoritmos de control:

- Control PID de sistema de una entrada y una salida.
- Control PID de sistema de dos entradas y dos salidas, con y sin desacoplamiento.

La comunicación e interacción entre Matlab y Arduino se ha realizado mediante el envío de mensajes por puerto serie, definiéndose un protocolo para las distintas tareas: envío de medición, envío de acción de control, etc. En principio, la idea es que la tarjeta

Arduino funcione como una tarjeta de adquisición de datos con cierta capacidad de procesamiento (por ejemplo para el muestreo basado en eventos), de forma que el algoritmo de control se implemente en Matlab.

Además, el sistema de control desarrollado es flexible y fácilmente adaptable a la adición de nuevos elementos en el sistema de control de la planta (actuadores o sensores).

Índice

Índice de figuras

Índice de tablas

Memoria

1.1. Objetivo

El objetivo del presente Trabajo Final de Grado es el desarrollo del sistema de control automático de un sistema de doble depósito, incluyendo el diseño de la interconexión entre dos sistemas de depósitos para que se pueda configurar entre un control monovariable con un sistema de depósitos acoplados o un control multivariable.

Para ello se deberá definir y programar la electrónica de señal necesaria para la adquisición de datos (nivel del líquido en los sistemas de depósitos) y de potencia para atacar las bombas, y desarrollar el firmware y software necesarios para poder implementar en un PC diversos algoritmos de control de forma flexible.

1.2. Antecedentes

Hace años, en el departamento de *Ingeniería de Sistemas Industriales y Diseño*, se compraron varios sistemas de depósitos para las prácticas de laboratorio, reservándose dos de ellos para montar un sistema de control multivariable para investigación, conectando los dos sistemas entre sí.

Los dos sistemas son iguales y están formados por un depósito inferior de acumulación, desde donde se bombea agua al depósito superior, en el que hay un sensor de nivel capacitivo. Durante la ascensión del líquido, se puede medir el caudal con la ayuda de un caudalímetro de pulsos integrado en el sistema. El agua del depósito superior se descarga por gravedad hacia el depósito de acumulación inferior a través de una válvula manual que puede ajustarse libremente, cambiando así la sección del orificio de descarga.

La idea será, por tanto, decidir cómo interconectar hidráulicamente esos dos sistemas para crear un sistema más complejo, que permita ser configurado de diversas formas (como sistema de una entrada y una salida o como sistema de dos entradas y dos salidas).

Además, uno de los dos sistemas de depósitos tiene el sensor de nivel capacitivo estropeado, por lo cual se deberá buscar un nuevo sensor que mida el nivel, con la restricción de que su coste no sea elevado.

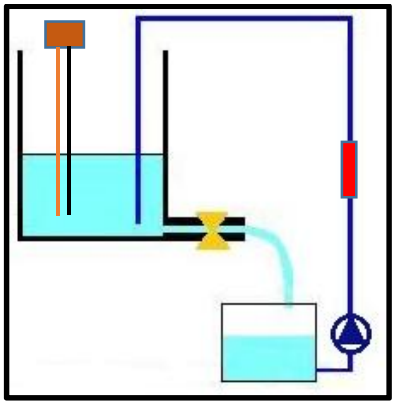


Figura 1. Sistema depósitos

1.3. Requisitos de diseño

Los requisitos que debe cumplir el presente proyecto serán:

- Una interconexión entre los dos sistemas de depósitos para que sea fácil poder pasar de un sistema de doble depósito acoplado, a un sistema multivariable.

- Se deberá definir un amplificador, que permita suministrar a las bombas una tensión de 15V y una corriente de 2A a partir de una fuente de alimentación adecuada.

- Como uno de los sensores de nivel capacitivo no funciona correctamente, deberá buscarse una alternativa a éste, con el requisito de que sea de bajo coste.

- La plataforma electrónica para la adquisición de datos deberá tener el menor coste posible.

- El software que implementa los algoritmos de control deberá ejecutarse en un PC que se comunique con la plataforma de adquisición. El entorno será flexible para poder implementar diversos algoritmos de control de forma sencilla.

1.4. Análisis de soluciones

Para la adquisición de datos, el mercado ofrece varias posibilidades, desde la utilización de tarjetas de adquisición de datos, las cuales se integran en el ordenador, hasta microcontroladores, o placas basadas en microcontroladores.

Después de observar los precios de estas posibilidades, el proyecto se centra en los microcontroladores o placas de microcontroladores, ya que las tarjetas de adquisición de datos tienen un coste muy elevado y son muy poco flexibles, a la hora de añadir nuevos elementos.

En cuanto a la elección el microcontrolador, existen varias opciones, como por ejemplo, diseñar una placa o escoger uno de los que hay en el mercado con su placa. De los numerosos microcontroladores del mercado, en el laboratorio de Automática se ha trabajado principalmente con las tarjetas de la familia Arduino (que incorporan microcontroladores de Atmel) y tarjetas de Microchip (familias PIC y dsPIC).

Antes de comparar un tipo de microcontroladores con el otro, o de decidir diseñar una placa, se hará una breve explicación en términos sencillos de qué es un microcontrolador.

Un microcontrolador es un dispositivo que se puede programar para hacer distintas tareas y para ello requiere de la ayuda de ciertos periféricos, pero también necesita otros elementos adicionales para poder interactuar con el mundo. Es, en cierto modo, como si se tuviera el CPU de un ordenador, es capaz de hacer muchas cosas, pero sin el teclado, el mouse y una pantalla donde visualizar la información no se podría hacer un uso adecuado del mismo. Esto ocurre también con el microcontrolador, sin teclados, sin conexión al ordenador, sin pantallas, o sin otras interfaces no se puede utilizar el microcontrolador correctamente.

Arduino es una plataforma de código abierto con la cual se pueden crear proyectos de electrónica digital, domótica, robótica, etc. de forma rápida, sencilla y económica. En la red se pueden encontrar cientos de proyectos con Arduino. Sus placas más populares son Arduino UNO y Arduino MEGA, basadas en los microcontroladores ATmega320 y ATmega2560 respectivamente de Atmel.

La principal diferencia entre Arduino y PIC es que PIC es una familia de microcontroladores propiamente dichos, y Arduino es una placa de desarrollo basada en un microcontrolador. En el mercado hay muchas placas de desarrollo basadas en microcontroladores PIC pero aparte de ser mucho más caras que las placas de Arduino, no han llegado a ser tan populares.

Sabiendo esto, la pregunta que puede surgir es ¿por qué Arduino se ha hecho tan popular si no son los primeros en hacer placas de desarrollo? Son varios los factores que lo han hecho conocido. El más destacado es que las placas de desarrollo Arduino son *Open Source*, tanto a nivel de hardware como de software, en Microchip, en cambio, se debe pagar una licencia para usar el software. También existe la posibilidad de que uno se fabrique la suya sin infringir derechos de Autor, incluso, fabricar placas exactamente iguales y venderlas y no violar la ley, pues es un sistema de código abierto. Además el software de programación también es libre.

Arduino es bastante útil para las personas aficionadas de la electrónica o para estudiantes que desean introducirse en este mundo. Y es que, para hacer una aplicación de Arduino, solo basta con comprar cualquiera de sus placas, descargar el software de programación y en cuestión de horas o un par de días estará lista la primera aplicación (para ello son necesarias nociones de programación en lenguaje C). Además, es práctico, por ejemplo, para alguien que no sea experto en la utilización de microcontroladores, con Arduino podría diseñar un sistema de iluminación innovador mucho más rápido que si lo hiciera con PIC.

Por tanto, normalmente, Arduino es más fácil de aprender que PIC, y como es bastante económico se utilizará un Arduino para este proyecto, ya que diseñar una placa de desarrollo o un microcontrolador, sería mucho más costoso que si se compra directamente el Arduino. Además se tendrá más flexibilidad para posibles cambios de sensores o bombas, que si se diseñara una placa, la cual se diseñaría para la estructura actual, y no tendría en cuenta todos los posibles sensores que se pueden conectar en el futuro y hacer la misma función.

Una vez decidida la familia de microcontrolador, se buscará una tarjeta amplificadora para controlar la tensión que le llega a las bombas. Para ello se busca una de las placas más comunes en Arduino, la L298N, pero la corriente máxima que soporta esta tarjeta es de 2A y la tensión máxima que permite es de 12V y cada una de las bombas

que se tiene en el laboratorio consume 2A, por lo que se necesitarían 2 placas de este tipo, ya que si no, no llegaría la corriente necesaria a las bombas. Tras hacer alguna prueba con este tipo de placas se observa que se calientan mucho y que hay muchas pérdidas. También se sabe que las bombas pueden llegar a funcionar a 15V y este tipo de placas no lo permite.

Tras buscar otro tipo de placas finalmente se decide diseñar una placa amplificadora que no se caliente tanto, y permita que le lleguen a las bombas los 15V y los 2A a cada una sin ningún tipo de problema.

En cuanto al software, a utilizar las opciones más comunes son la utilización de Matlab u Octave. Otra de las opciones que hay es realizar un programa, programado con C, Java, o cualquiera de los lenguajes de programación existentes, pero Matlab y Octave, ofrecen una gran ayuda al programador. Por lo que la opción que se escoge es la utilización de Matlab u Octave.

En la documentación del programa Octave se describe a éste como un lenguaje de programación de alto nivel, orientado al Cálculo Numérico. Proporciona una consola para resolver problemas lineales y no lineales con el ordenador y para desarrollar experimentos numéricos.

Octave puede ser copiado, modificado y redistribuido libremente bajo los términos de la licencia GNU GPL tal como se publica por la *Free Software Foundation.*

Octave fue diseñado para ser una herramienta dentro de la línea de comandos del sistema operativo GNU, aunque posteriormente ha sido portado a muchos más sistemas operativos. También en un principio, fue un lenguaje de programación independiente pero ha ido convergiendo a Matlab hasta el punto de buscar la compatibilidad con él. Tampoco ha sido nunca un objetivo dotarle de interfaz gráfica, pero se pueden encontrar ya un par de ellas con calidad suficiente. Todo lo contrario a Matlab, donde uno puede realizar interfaces gráficas para su aplicación, con una gran ayuda de Matlab, el cual facilita mucho la programación de aplicaciones con interfaces gráficas.

Aunque Octave es capaz de ejecutar la mayoría del código escrito en Matlab, tanto su historia como su arquitectura interna son completamente distintas. Una de las diferencias más evidentes es que están escritos en lenguajes de programación distinto, Matlab en C y Octave en C++.

Octave es hoy en día una herramienta inferior a Matlab pero para tratarse de algo totalmente gratuito desarrollado por una comunidad de ingenieros, científicos y entusiastas, se trata de una herramienta de una gran calidad. Para pequeños proyectos es una alternativa a Matlab completamente viable, el cual, por motivos de licencias, está divido en paquetes. Cada uno cumple una función específica y puede ser adquirido a parte. Esto impone una limitación añadida a Matlab porque, aunque una empresa o una universidad se haya gastado grandes cantidades de dinero en licencias de Matlab, es posible que no haya adquirido el *toolbox* que se necesita.

Otras plataformas de cálculo para Ciencia e Ingeniería como Scilab o IDL cuentan con sus propios lenguajes de programación, lo que hace que sean descartados, ya que con la opción Matlab y Octave, se tienen lenguajes de programación más conocidos, usados por un mayor número de programadores, aunque ellos no utilicen Matlab u Octave. Esto permite que el programa que se realizará lo entienda un mayor número de personas, sin tener que aprender un nuevo lenguaje de programación.

Para la sustitución del sensor de nivel capacitivo, se buscan nuevos sensores capacitivos, o variantes. Una de las variantes es el sensor de nivel barométrico, pero la instalación de este tipo de sensor resulta complicada, así que se descarta (además de por el hecho de ser muy cara). Así pues se mira el precio de un sensor de nivel capacitivo, y aunque es algo más barato que el barométrico es, igualmente, es muy caro. De este modo se intenta encontrar otra variante, y ésta no es otra que los sensores de ultrasonidos, que son muy económicos y aunque la precisión no es tan exacta, mediante código se puede mejorar un poco con la ayuda de un filtrado.

Y, finalmente, el último tema a tratar es cómo conectar el Arduino al ordenador, las opciones que hay son: por puerto serie (la más usada, y la necesaria para cargar un programa nuevo al Arduino), mediante Ethernet o por WiFi. Y como las opciones de Ethernet y WiFi encarecerían el resultado final, además de que provocaría que el Arduino necesitara de una alimentación, se decide optar por la opción del puerto serie, la cual no encarece el producto, y ya le proporciona la tensión al Arduino, sin necesidad de una fuente de alimentación. Además de facilitar el código y la comprensión de éste a futuros usuarios.

1.5. Descripción general de la solución adoptada

Una vez analizadas las opciones, en este apartado se describirá de forma general la solución final adoptada.

En primer lugar, se ha analizado cómo conectar los dos sistemas de depósitos para que se pueda cambiar la configuración de sistema de doble depósito acoplado a sistema multivariable fácilmente. Para ello se han conectado los depósitos superiores entre sí con una válvula intermedia, para regular el caudal que pasa de un depósito al otro, y también los inferiores, para que ninguno de los inferiores se quede sin líquido para ser bombeado por las bombas.

La tensión a las bombas llegará mediante un amplificador diseñado para el presente proyecto, el cual estará conectado a una fuente de alimentación de 15V.

La adquisición de los datos se realizará con un Arduino MEGA. El Arduino está conectado al ordenador mediante el puerto serie. Además de estar conectado mediante el pin 11 y 12 al amplificador. Con el pin 11 se envía la señal PWM, que le debe llegar a la bomba del sistema con el sensor de nivel capacitivo, y con el 12 a la otra bomba. El sensor capacitivo está formado por tres cables, éstos se conectan al Arduino de la siguiente manera: el negro a tierra, el blanco a V_{CC} y el marrón al pin de entrada analógica A0. Por otro lado, el sensor que se ha escogido para medir el nivel del líquido en el otro depósito es un sensor de ultrasonidos, el cual será conectado la GND con la GND, la V_{CC} con los 5V del Arduino, el *trigger* al pin 5 y el *echo* al pin 4.

Los datos que reciba el Arduino de estos sensores los enviará al ordenador, a petición de éste. La aplicación de control del PC estará realizada con el programa Matlab. Esta aplicación estará preparada para implementar el algoritmo de control del sistema de doble depósito y del sistema multivariable.

En el futuro se puede experimentar en el hecho de cómo influye la apertura de las válvulas en el sistema. Para el ejemplo que se expone en este proyecto las válvulas estarán totalmente abiertas, excepto en el caso del sistema de doble depósito acoplado, que, en ese caso, se cerrará la válvula de la bomba sobre la que se esté actuando.

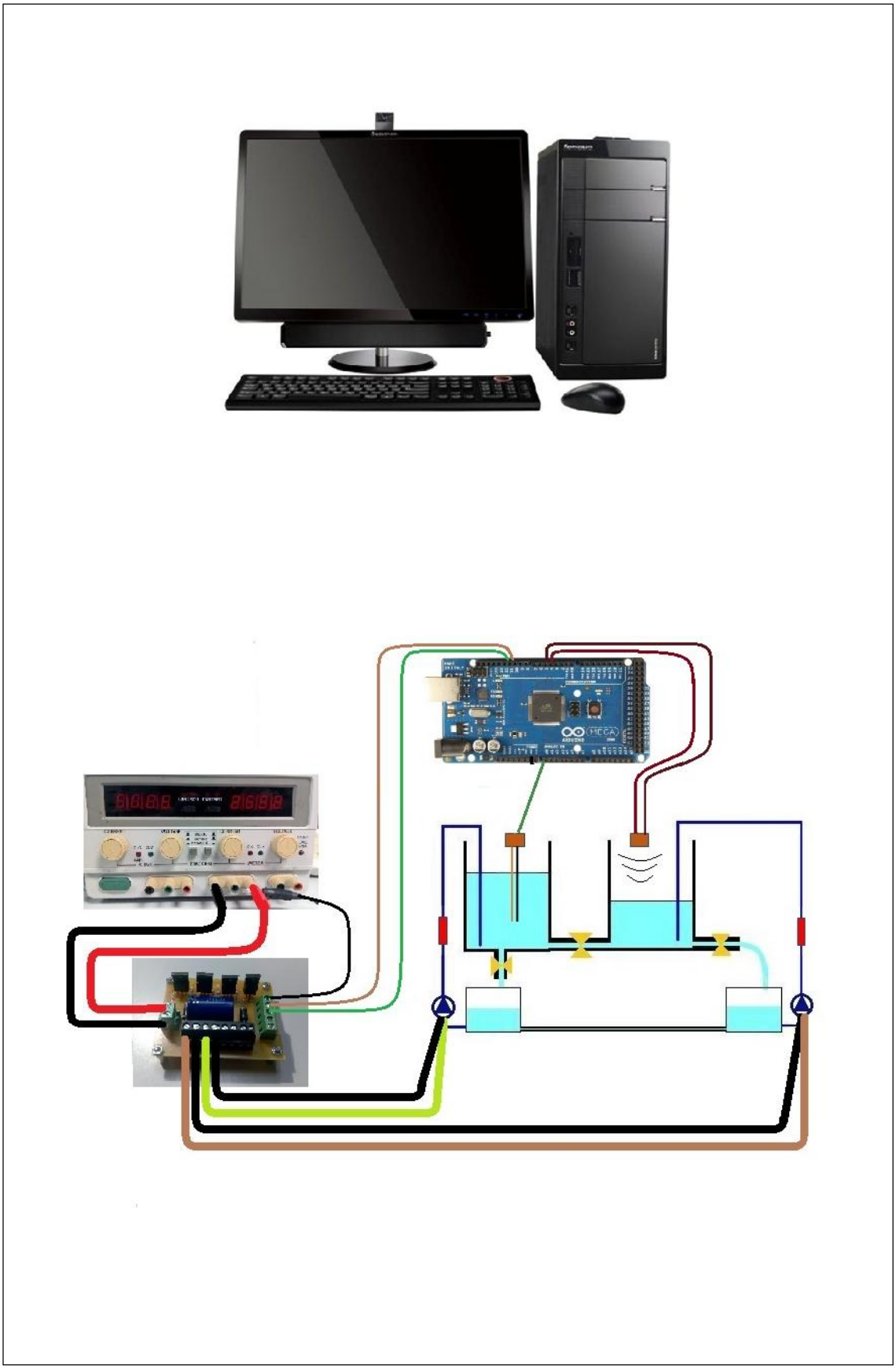


Figura 2. Diseño de la planta

1.6. Diseño del amplificador para las bombas

Para el diseño de la placa controladora, la principal característica que se ha buscado es que soporte los 2 A por motor, y que pueda suministrar los 15 V con una caída de tensión muy baja. Este problema se ha solucionado eligiendo componentes discretos que soporten dicho amperaje con soltura, y que tienen una caída de tensión en conducción muy baja.

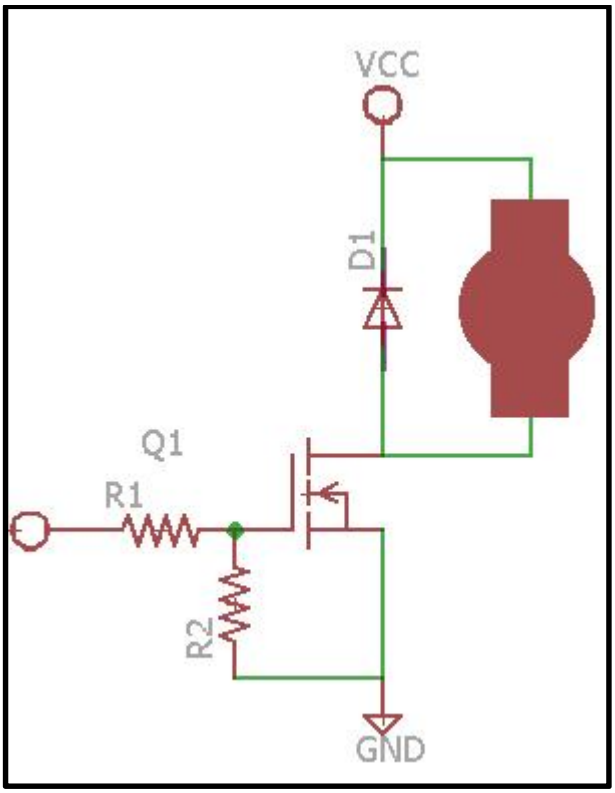


Figura 3. Esquema eléctrico para una bomba

En la figura 3 se observa el diseño utilizado, el cual es simple de analizar, ya que la función principal del transistor es abrir y cerrar el circuito de la bomba, para que por este circule corriente o no. Lo que se hace es colocar la señal PWM a la entrada del transistor. El transistor al ser bastante rápido conmutará la señal, provocando conmutaciones del circuito cerrado de la bomba según el estado alto de la señal PWM. Lo que se consigue con este esquema es que el motor vea una señal continua (media de la señal PWM).

Como se puede ver en la figura 4 el transistor escogido es un MOSFET, ya que estos son los más adecuados para este sistema, ya que se necesitan velocidades de

conmutación rápidas y que el consumo no sea excesivo, es decir, que la R_{DS} sea pequeña para una tensión de entrada de 5V (tensión proporcionada por el Arduino).

El transistor MOSFET elegido es el IRLIZ44GPbF, cuyas características más relevantes son las siguientes:

- V_{DSS} = 60V
- R_{DS} = 0.028Ω (para V_{GS}=5V)
- I_D = 30A

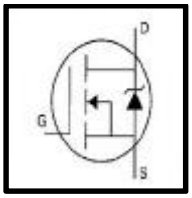


Figura 4. MOSFET

Una característica a destacar de este transistor es la R_{DS} (resistencia entre Drenador y Fuente), que es muy pequeña. En la figura 4 se puede observar el esquema proporcionado por el fabricante y llegar a la conclusión que la R_{DS} estará en serie con la bomba. Con esta resistencia y el consumo de la bomba se puede calcular la caída de tensión máxima que tendrá el transistor cuando se apliquen los 15V a las bombas.

$$V = R \cdot I$$

$$V = 0.028 \cdot 2 = 56mV$$

Mediante el cálculo anterior se obtiene la caída de tensión en el transistor, el resultado es bastante bueno, ya que la caída de tensión es insignificante gracias a la pequeña resistencia, lo que ayuda a aprovechar toda la tensión para la bomba, y a disminuir la potencia disipada por el transistor.

Luego también se pone un diodo, el cual sirve para proteger al circuito, no a la bomba. Este diodo se le conoce como diodo de recirculación. Está inversamente polarizado porque al conmutar un circuito con una inductancia, la corriente se dice que "se invierte" en la bobina, regresando esa corriente al circuito. Si se tiene un diodo, esta corriente lo polariza directamente y se descarga a través de él, sin afectar al resto del circuito.

Finalmente se añadirá un condensador a la entrada de la tensión de alimentación para filtrar posible fluctuaciones.

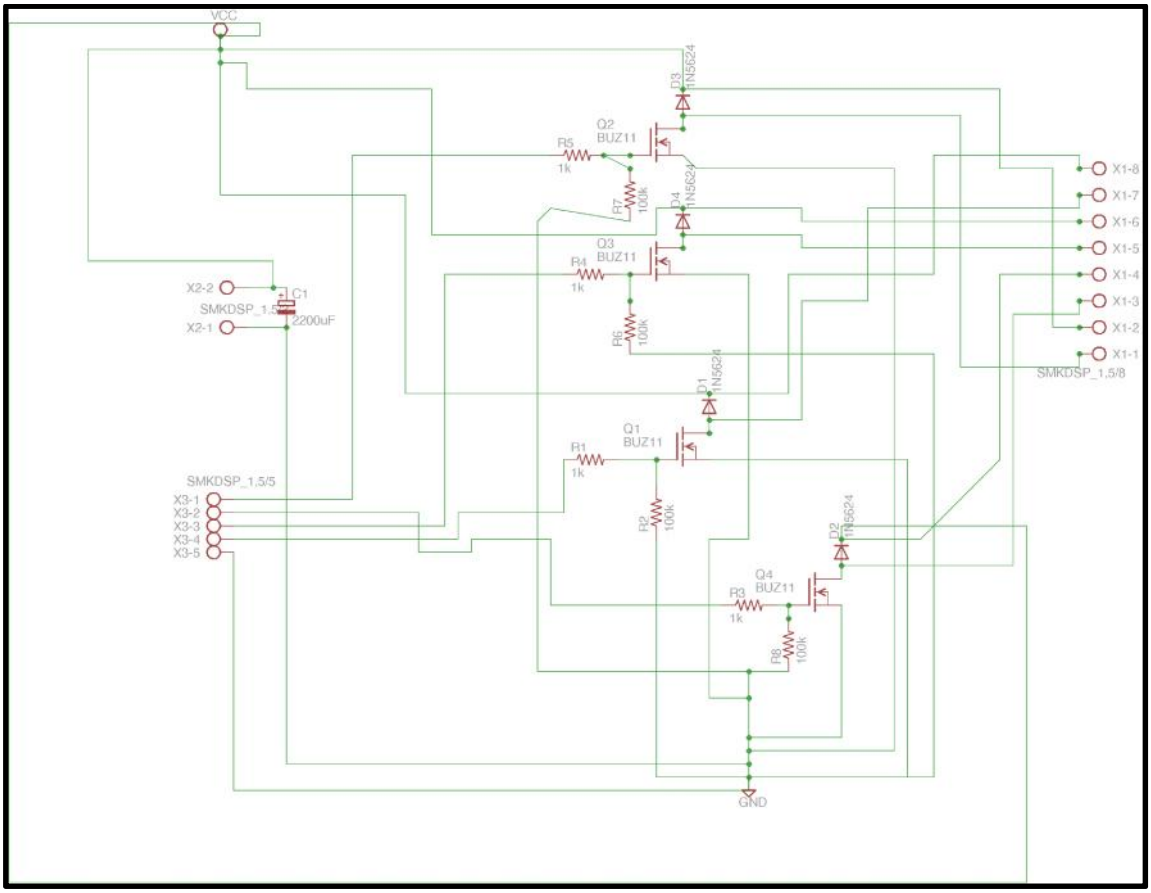


Figura 5.Esquema eléctrico completo

Listado de componentes del circuito amplificador:

Componente	Referencia	Precio (€)	Unidades
MOSFET	IRLIZ44GPbF	2.05	4
Diodo		0.16	4
Resistencia 1K		0.03	4
Resistencia 100K		0.04	4
Conector 8 pines		1.23	1
Conector 5 pines		1.31	1
Conector 2 pines		0.93	1
Condensador 220nF		1.12	1

Tabla 1.Componentes electrónicos usados

PCB son las siglas de *Pinted Circuit Board,* que en español se conoce como PCB o como circuito impreso. Circuito impreso también puede ser el conjunto de PCB y componentes ya insertados.

El PCB es un material no conductor (baquelita, fibra de vidrio, etc.) que posee varias láminas o caras de un material conductor (cobre). Con estas láminas conductoras se crean las pistas por donde irá la electricidad. Cuantas más láminas tiene el PCB, más complejo es.

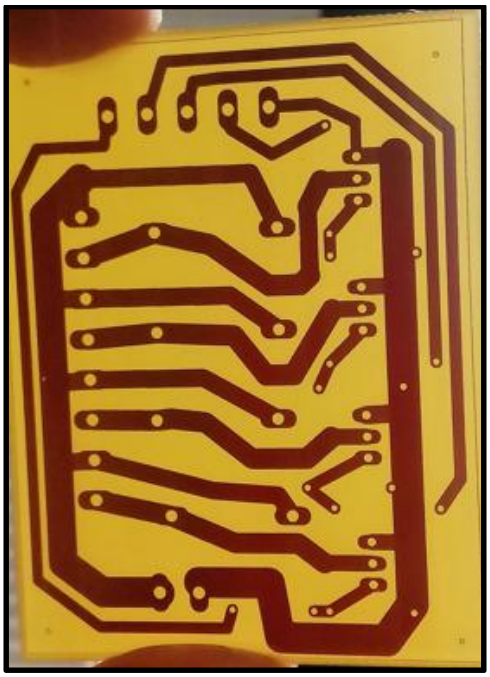


Figura 6.PCB

Esta placa está realizada únicamente sobre una de sus caras (cara bottom), ya que no se trata de un circuito muy complejo y se puede realizar perfectamente de esta manera, con el consiguiente ahorro tanto económico como de materiales que esto reporta. Está compuesta de varios componentes eléctricos (resistencias, condensadores, transistores y diodos) que, añadidos al circuito, se encargan de comunicar entre sí las distintas partes del mismo.

Estas pistas conectarán los componentes electrónicos para conseguir el funcionamiento deseado.

Para escoger los componentes se debe tener en cuenta que éstos pueden ser THROUGH-HOLE, es decir, de agujeros pasantes o en términos coloquiales, se les puede

llamar convencionales. Este método consiste en insertar los terminales de los componentes en los orificios del PCB.

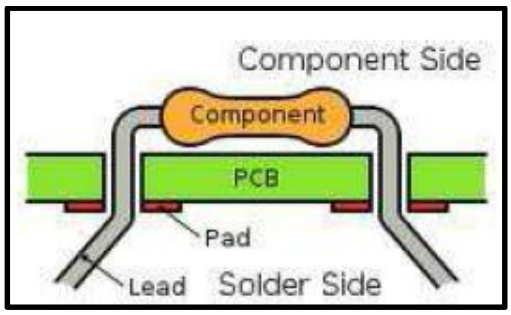


Figura 7.THOUGH-HOLE

O también pueden ser SMD (Surface Mount Devices) y estos componentes se colocan sobre el pad del PCB.

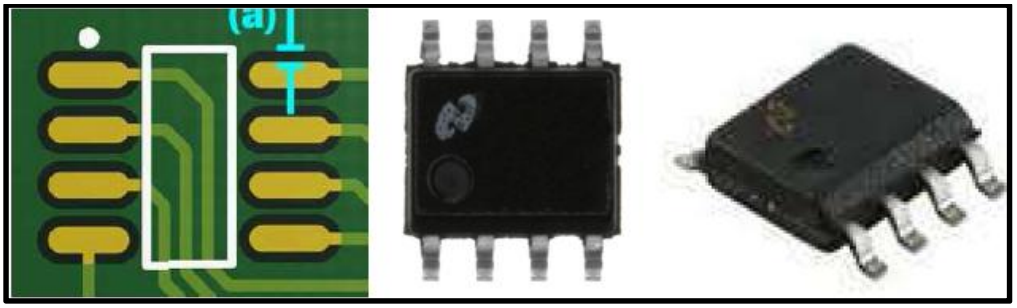


Figura 8.SMD

Para el presente proyecto se ha decidido usar los del tipo THROUGH-HOLE en todos los componentes, ya que son más económicos.

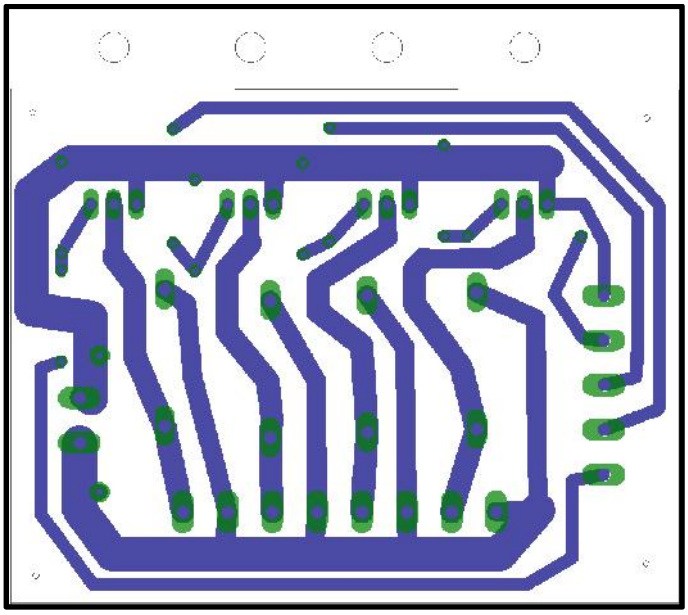


Figura 9. Cara para soldadura

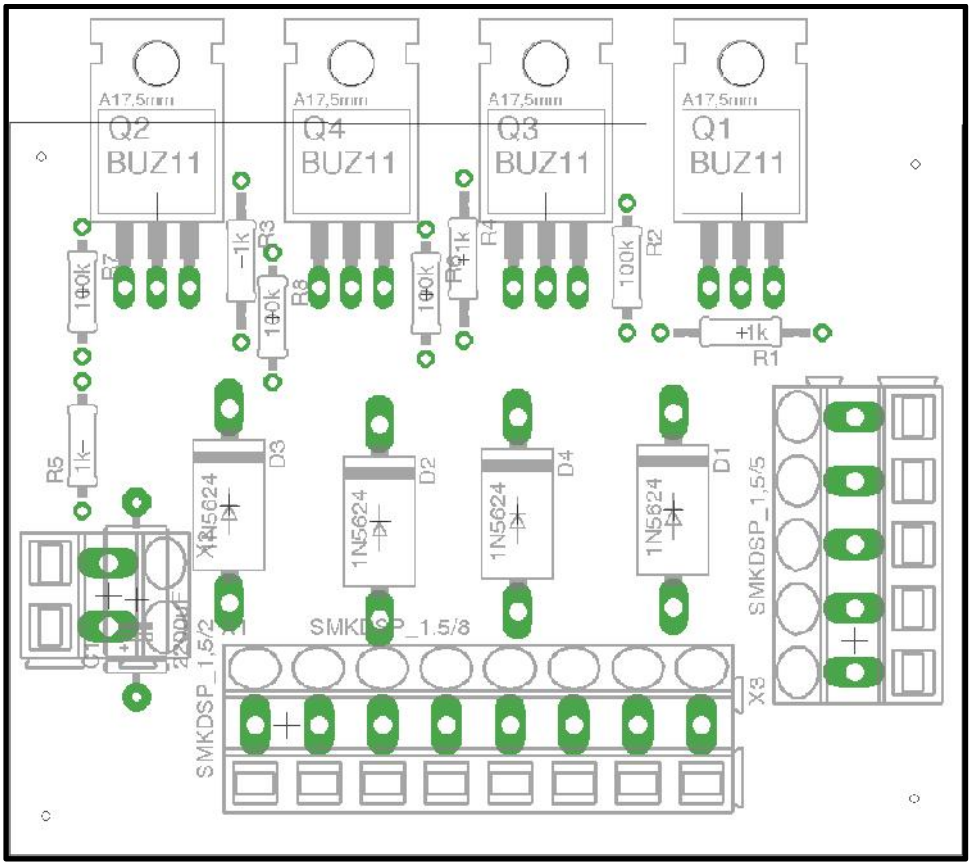


Figura 10. Componentes electrónicos

Una vez explicado esto se procede a explicar el proceso de fabricación de la PCB.

Se pueden utilizar muchos programas para realizar las pistas. Entre los más utilizados están el PCB Express y el Eagle, que es el que se ha usado en este proyecto.

Una vez hecho el diseño, se imprime el circuito en papel vegetal.

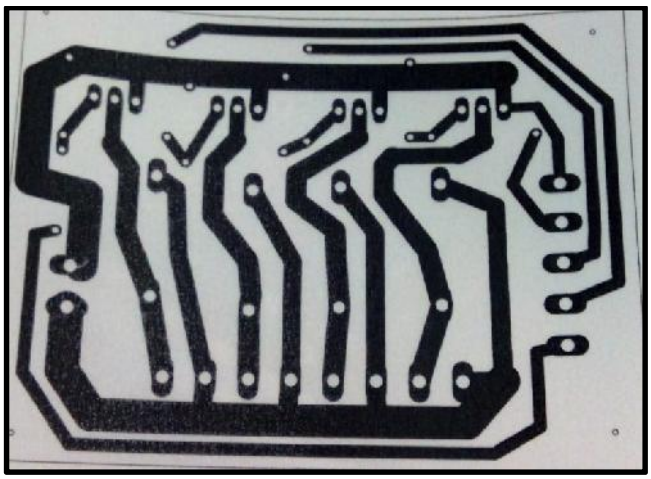


Figura 11. Cara soldadura impresa en papel vegetal

Figure 1

En este punto se saben las dimensiones físicas de la placa y se puede cortar un trozo de placa fotosensible aproximadamente del mismo tamaño. Más adelante, se harán algunos ajustes cortando la parte sobrante.

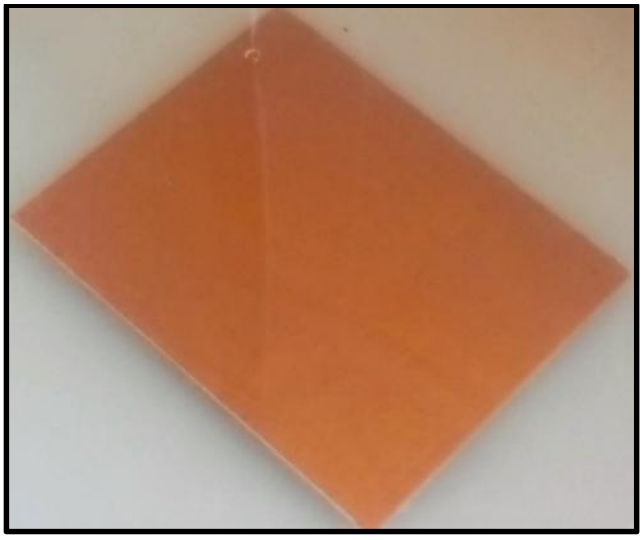


Figura 12. Placa de baquelita

Una vez hecho esto, se pone la placa con el circuito impreso en la insoladora de rayos UVA. Hay que prestar especial atención sobre el lado a exponer, ya que esto es un fallo común y hay que tenerlo en cuenta porque se puede insolar una placa que luego no sirva.

Figura 13. Rayos UVA

Este proceso dura entre 4 y 5 minutos, dependiendo del circuito. Una vez pasado este tiempo ya se tiene la placa lista. Es la hora de bañarla en un revelador durante unos minutos, moviendo la placa como si se estuviera revelando una foto (de hecho, el proceso es el mismo). Se podrá observar como las pistas irán apareciendo.

Figura 14.Tratamiento revelador

Cuando se vean las pistas perfectamente definidas, se podrá retirarla y comenzar el atacado químico. Para ello, se usará sodio persulfato cristal, ya que es de los más fiables y poco contaminantes que hay en el mercado, aunque el proceso es un poco lento.

Figura 15. Persulfato, y su aplicación

Poco a poco el ácido se comerá el cobre que ha sido “tocado” por la luz pero no el que está protegido por la pequeña película que define las pistas.

Cuando se vean todas las pistas bien definidas, ya se puede sacar la placa del ácido y limpiarla con agua y un cepillo de dientes. Cabe señalar que si las pistas son muy finas, es preferible frotar con un poco de algodón y acetona para no arrancarlas.

Figura 16. Limpieza placa

Aquí se obtiene el resultado final con un acabado casi perfecto. Para comprobar si ha quedado algún resto de cobre sin quitar, se puede mirar a contra luz la placa y así poder apreciar hasta el más mínimo detalle (pistas cortadas y restos de cobre).

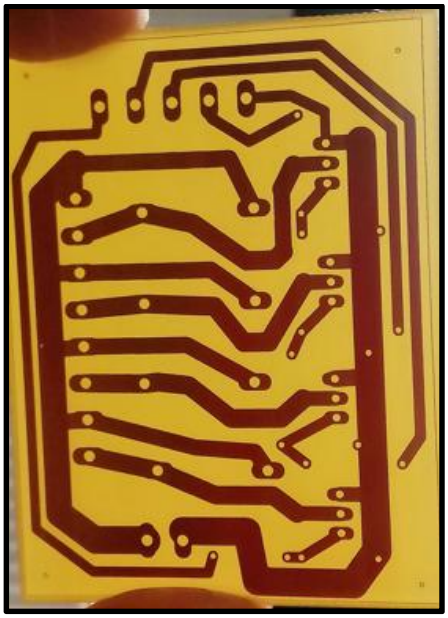


Figura 17. PCB

Ahora que ya se tiene la placa, con una taladradora se realizan los agujeros donde se pondrán los componentes electrónicos. Si el agujero es muy grande, es conveniente hacer agujeros de diámetro pequeño, y luego ir incrementando el diámetro. A la placa se le pueden añadir 4 agujeros que sirvan de soporte a la placa, y así quede más estable.

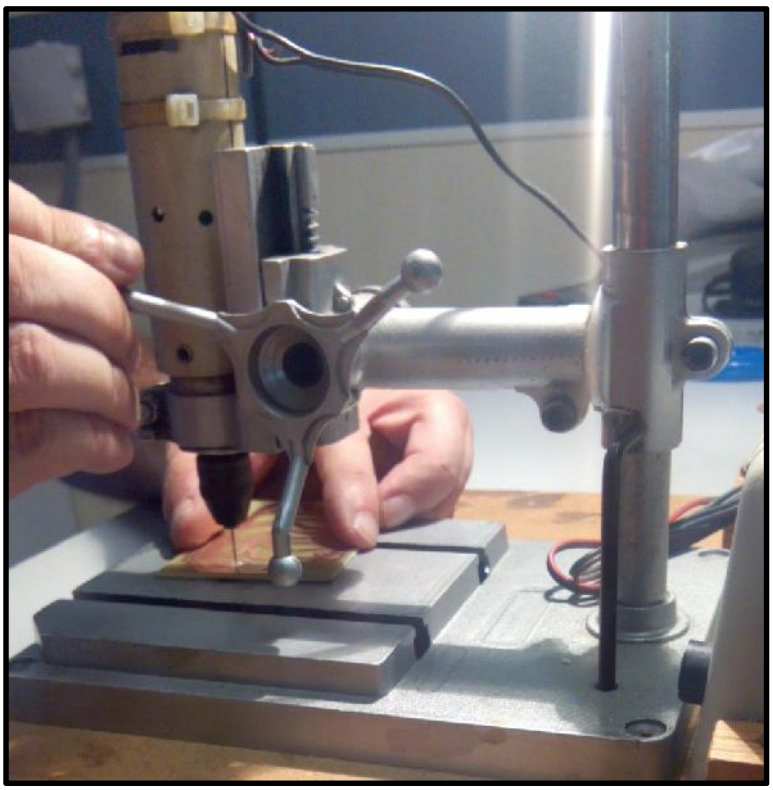


Figura 18. Taladrando la PCB

Figura 19. Diodos y condensadores

Finalmente, una vez hechos todos los agujeros, ya se pueden soldar los componentes electrónicos.

Figura 20. Soldando

Siendo el resultado final, el de la figura 21.

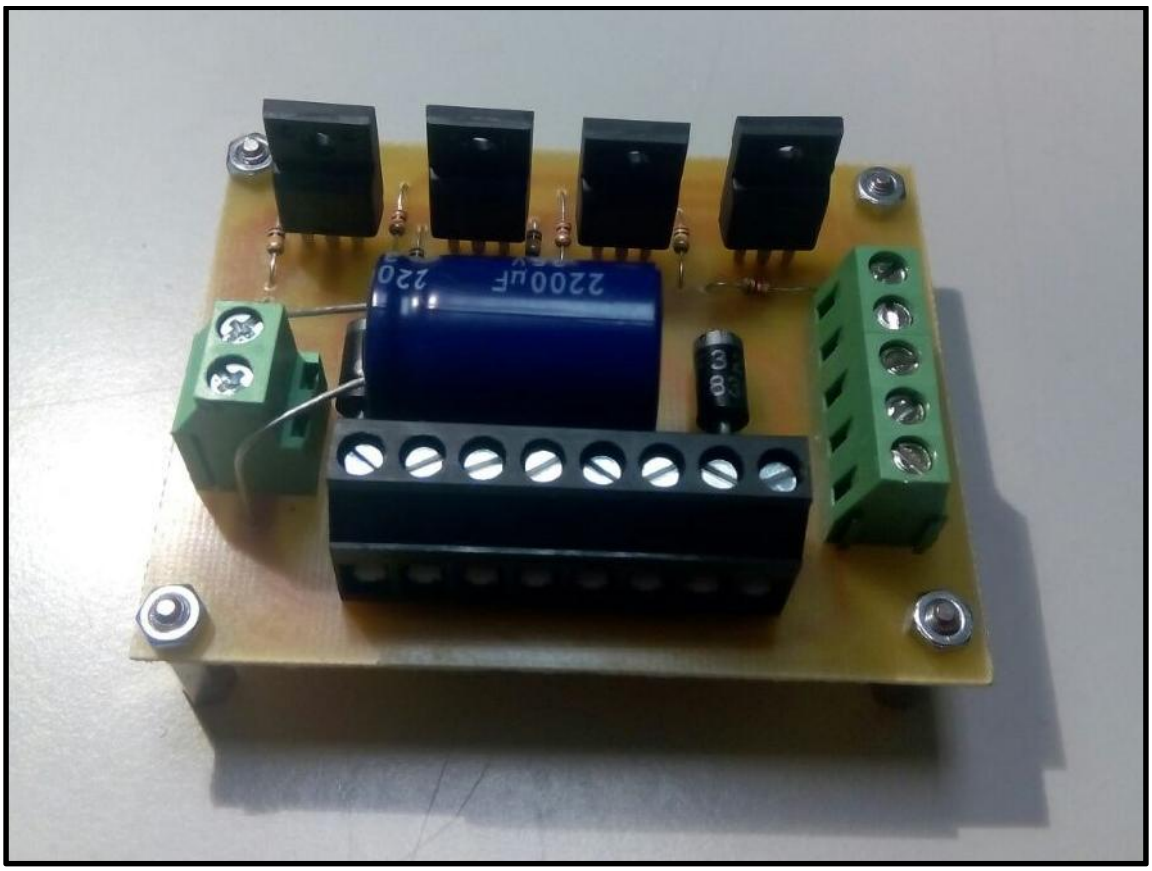


Figura 21. Resultado final de la PCB

1.7. Configuración de un sensor de nivel por ultrasonidos

Como se ha dicho anteriormente, en el presente proyecto se usa un sensor de ultrasonidos para medir el nivel del agua, por ello en este apartado se explicará un poco el funcionamiento de este tipo de sensor.

Los ultrasonidos son, antes que nada, sonido, exactamente igual que los que oímos normalmente, salvo que tienen una frecuencia mayor que la máxima audible por el oído humano. Ésta comienza desde unos 16 Hz y tiene un límite superior de aproximadamente 20 KHz, mientras que el ultrasonidos utiliza sonido con una frecuencia de 40 KHz.

El funcionamiento básico de los ultrasonidos como medidores de distancia se muestra de una manera muy clara en el esquema de la figura 22, donde se tiene un receptor

que emite un pulso de ultrasonido que rebota sobre un determinado objeto y la reflexión de ese pulso es detectada por un receptor de ultrasonidos:

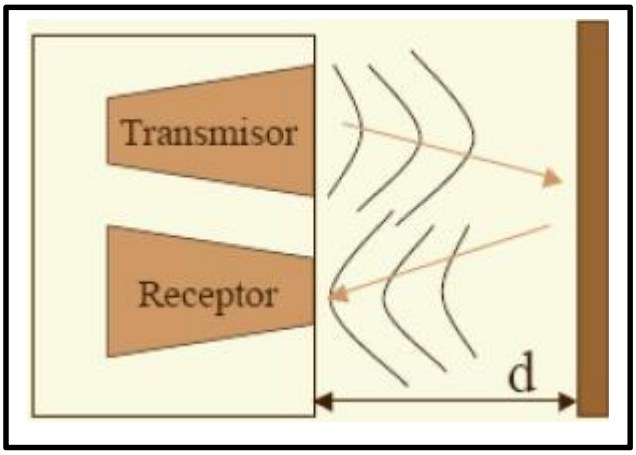


Figura 22. Funcionamiento ultrasonidos

La mayoría de los sensores de ultrasonidos de bajo coste se basan en la emisión de un pulso de ultrasonido cuyo lóbulo, o campo de acción es de forma cónica. Midiendo el tiempo que transcurre entre la emisión del sonido y la percepción del eco, se puede establecer la distancia a la que se encuentra el obstáculo que ha producido la reflexión de la onda sonora, mediante la fórmula:

$$d = \frac{1}{2} \cdot V \cdot t$$

Donde *V* es la velocidad del sonido en el aire y *t* es el tiempo transcurrido entre la emisión y recepción del pulso.

A pesar de que su funcionamiento parece muy sencillo, existen factores inherentes tanto a los ultrasonidos como al mundo real, que influyen de una forma determinante en las medidas realizadas. Por tanto, es necesario un conocimiento de las diversas fuentes de incertidumbre que afectan a las medidas para poder tratarlas de forma adecuada, minimizando su efecto en el conocimiento del entorno que se desea adquirir. Entre los diversos factores que alteran las lecturas que se realizan con los sensores de ultrasonido cabe destacar:

- El campo de actuación del pulso que se emite desde un transductor de ultrasonido tiene forma cónica. El eco que se recibe como respuesta a la reflexión del sonido indica la presencia del objeto más cercano que se encuentra

dentro del cono acústico y no especifica en ningún momento la localización angular del mismo. Aunque la máxima probabilidad es que el objeto detectado esté sobre el eje central del cono acústico, la probabilidad de que el eco se haya producido por un objeto presente en la periferia del eje central no es en absoluto despreciable y ha de ser tenida en cuenta y tratada convenientemente.

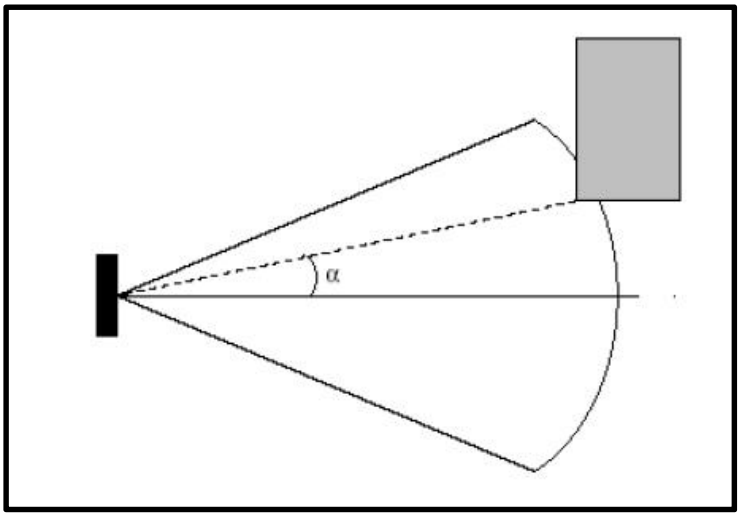


Figura 23. Posibles problemas del ultrasonidos 1

- La cantidad de energía acústica reflejada por el obstáculo depende en gran medida de la estructura de su superficie. Para obtener una reflexión altamente difusa del obstáculo, el tamaño de las irregularidades sobre la superficie reflectora debe ser comparable a la longitud de onda de la onda de ultrasonido incidente.
- En los sensores ultrasonido de bajo coste se utiliza el mismo transductor como emisor y receptor. Tras la emisión del ultrasonido se espera un determinado tiempo a que las vibraciones en el sensor desaparezcan y esté preparado para recibir el eco producido por el obstáculo. Esto implica que existe una distancia mínima *d* (proporcional al tiempo de relajación del transductor) a partir de la cual el sensor mide con precisión. Por lo general, todos los objetos que se encuentren por debajo de esta distancia, *d*, serán interpretados por el sistema como que están a una distancia igual a la distancia mínima. Para evitar esto se ha

escogido un sensor con un transductor de emisor y otro de receptor, y así ganar precisión.

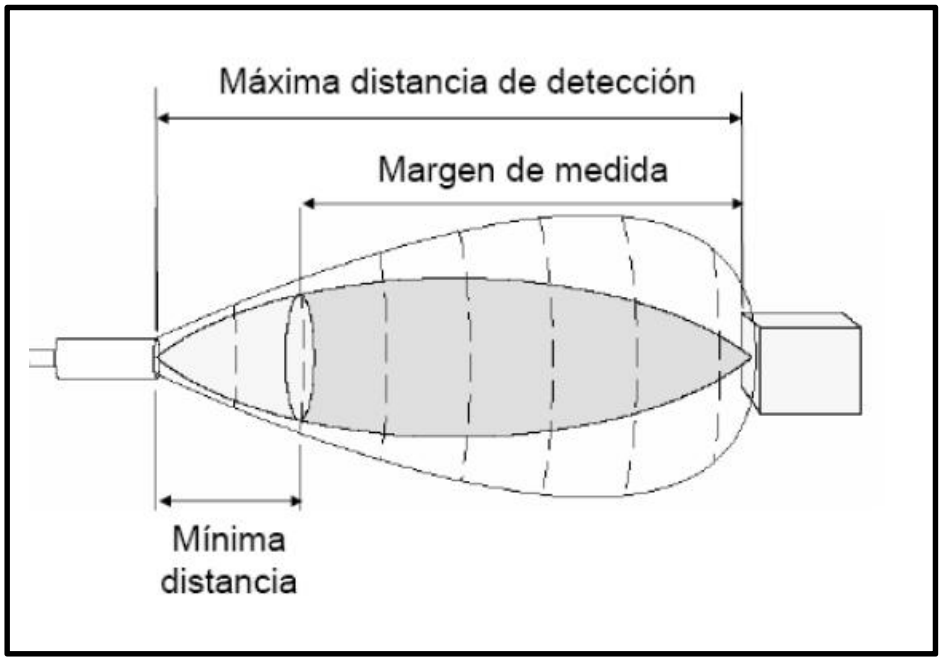


Figura 24. Posibles problemas del ultrasonidos 2

- Un factor de error muy común es el conocido como *falsos ecos*. Estos falsos ecos se pueden producir por razones diferentes: puede darse el caso en que la onda emitida por el transductor se refleje varias veces en diversas superficies antes de que vuelva a incidir en el transductor (si es que incide). Este fenómeno, conocido como reflexiones múltiples, implica que la lectura del sensor evidencia la presencia de un obstáculo a una distancia proporcional al tiempo transcurrido en el viaje de la onda; es decir, una distancia mucho mayor que a la que está en realidad el obstáculo más cercano, que pudo producir la primera reflexión de la onda. Otra fuente más común de *falsos ecos*, conocida como *crosstalk*, se produce cuando se emplea un cinturón de ultrasonidos donde una serie de sensores están trabajando al mismo tiempo. En este caso puede ocurrir (y ocurre con una frecuencia relativamente alta) que un sensor emita un pulso y sea recibido por otro sensor que estuviese esperando el eco del pulso que él había enviado con anterioridad (o viceversa).

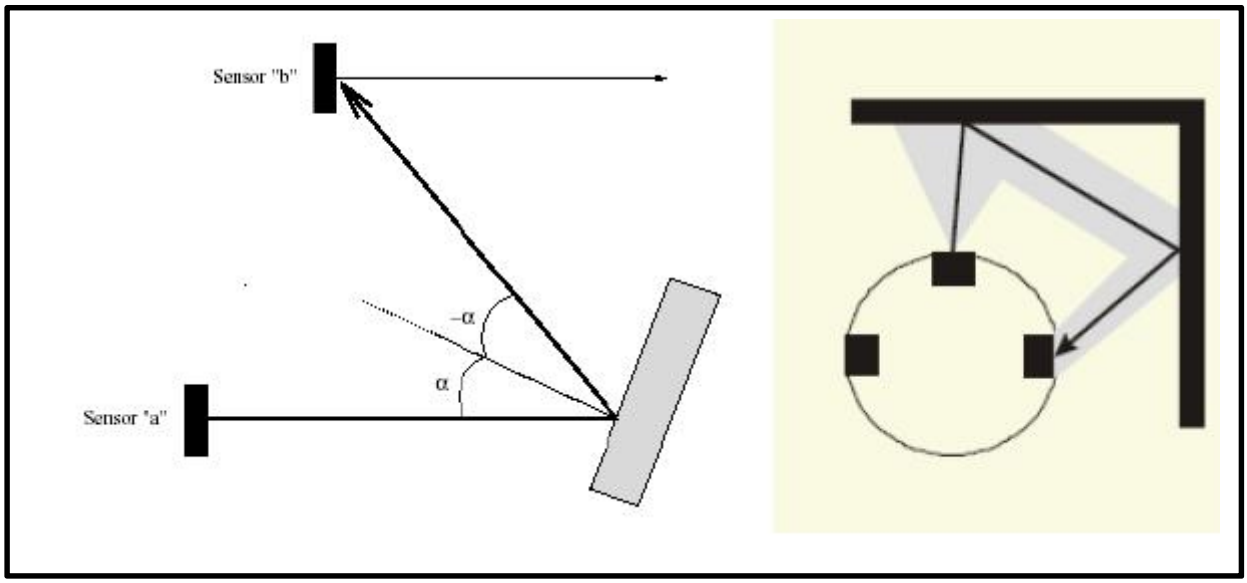


Figura 25. Posibles problemas del ultrasonidos 3

- Las ondas de ultrasonido obedecen a las leyes de reflexión de las ondas, por lo que una onda de ultrasonido tiene el mismo ángulo de incidencia y reflexión respecto a la normal a la superficie. Esto implica que si la orientación relativa de la superficie reflectora con respecto al eje del sensor de ultrasonido es mayor que un cierto umbral, el sensor nunca recibirá el pulso de sonido que emitió.

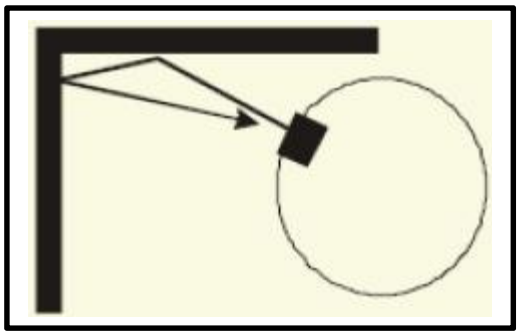


Figura 26.Posibles problemas del ultrasonidos 4

Una vez sabidos todos estos posibles problemas que afectan a la lectura, se escoge el sensor ultrasonidos a utilizar. En el presente proyecto se ha escogido el sensor HC-SR04, cuyas características y funcionamiento se puede observar en la figura 26:

Specifications

Parameters	Specification
Operating Voltage	+5V DC
Operating Current	15mA
Operating Frequency	40KHz
Maximum Distance	400cm
Minimum Distance	2cm
Detect Angle	15 degree
Resolution	0.3cm
Input Trig Signal	>10us TTL pulse
Output Signal	TTL pulse with width representing distance
Weight	
Dimension	45 x 20 x 15 mm

Figura 27. Especificaciones del sensor ultrasonidos

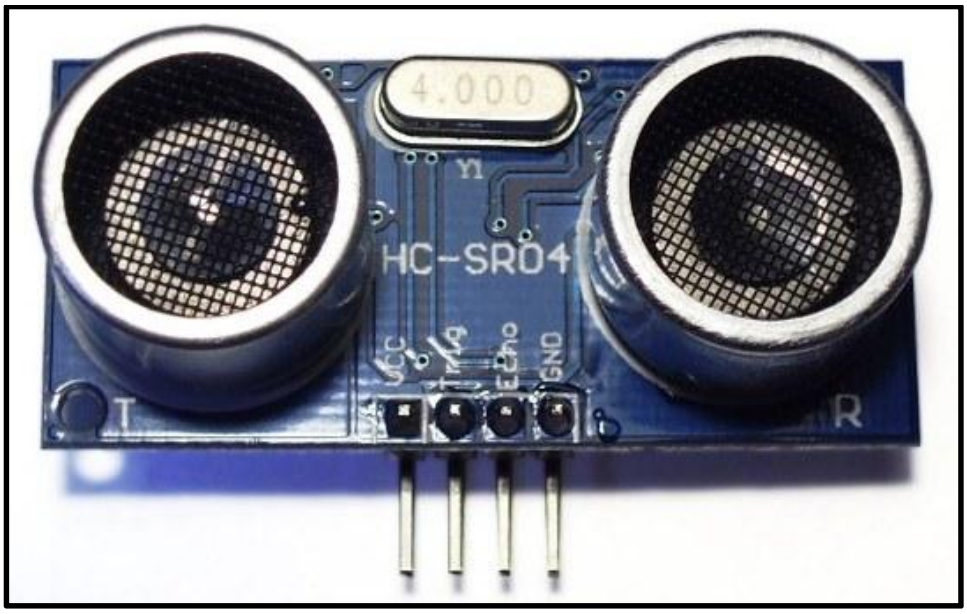


Figura 28. El sensor ultrasonidos

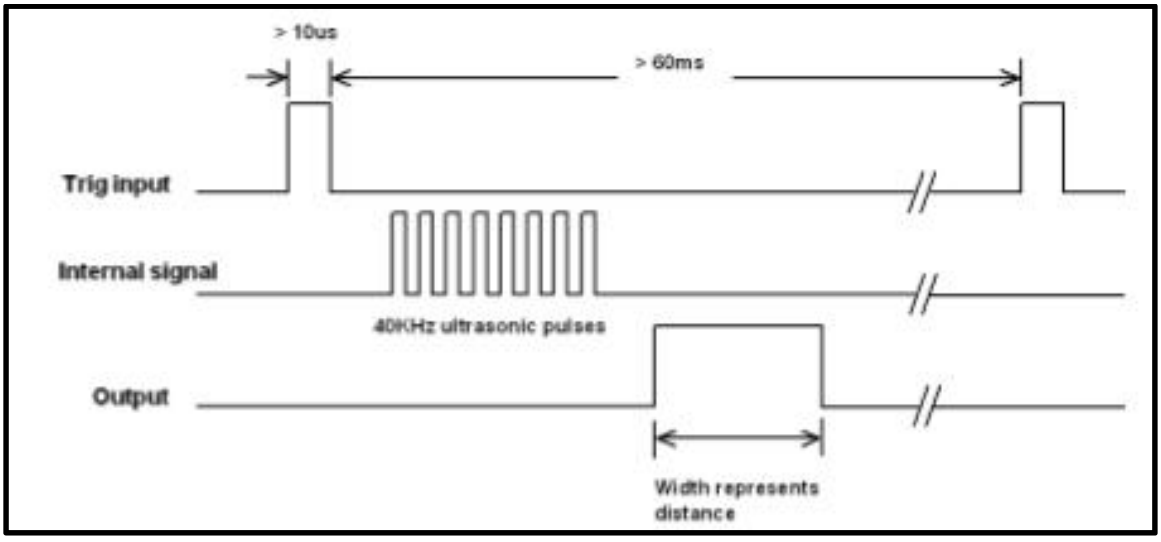


Figura 29. Señales del sensor ultrasonidos

La conexión del sensor de ultrasonidos se hará como en la figura 32, donde se puede observar cómo se conecta el GND del ultrasonidos al GND del Arduino, el Vcc del ultrasonidos a los 5V del Arduino y el *trigger* y el *echo* del sensor ultrasonidos en los pines 4 y 5 del Arduino respectivamente . Una vez conectado, se hace un programa muy simple con Arduino, para comprobar que no hay ningún error de lectura de las distancias. Para mejorar la precisión de la medición, se realizan mediciones periódicas cada 50 ms (con un *delay* en el bucle) y se utiliza un filtro de constante de tiempo 10 segundos. Para hacer el programa se utilizará una de las librerías de Arduino, la *NewPing*, quedando el programa de la siguiente forma (el programa utilizado para medir el nivel del agua es muy parecido al siguiente):

```
/* Autor: Aitor Roig Alemany                                    */
/* Fecha: Agosto 2015                                           */
/* "Sistema de control automático de una planta de laboratorio  */
/* basada en 2 depósitos"                                       */


/* Programa de prueba del sensor ultrasonidos.                  */

#include <NewPing.h>

/* Definimos los pines a los que conectaremos los sensores      */
#define TRIGGER_PIN 4

#define ECHO_PIN 5

#define MAX_DISTANCE 200

#define SOUND_SPEED 0.0171 // La mitad de la velocidad del sonido en el aire, medida en [mm/us]

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

double microsegundos;

double distanciamm, distancia;

const double tfiltro = 10;

double pd;
```

Figura 30. Definición constantes

```
void setup() {

  Serial.begin(115200);

}
void loop() {

  ultrasonidos();
  Serial.println(distancia);
  delay(50);


}


/* Funcion que devuelve el nivel del agua, tiene un filtro          */
/* para que la curva sea más suave y con poco ruido.                */

void ultrasonidos() {

  pd = exp(-0.05 / tfiltro);

  microsegundos = sonar.ping();

  distanciamm = microsegundos * SOUND_SPEED;

  distancia = distancia * pd + distanciamm * (1 - pd);

}
```

Figura 31.Código para el sensor ultrasonidos.

Si al ponerlo en marcha no observamos ningún problema en la lectura de la distancia, la localización del sensor (está localizado en el espacio reservado para ello en el sistema de depósitos) será correcta, de ocurrir lo contrario, se debería modificar la localización del sensor de ultrasonidos en el sistema de depósitos.

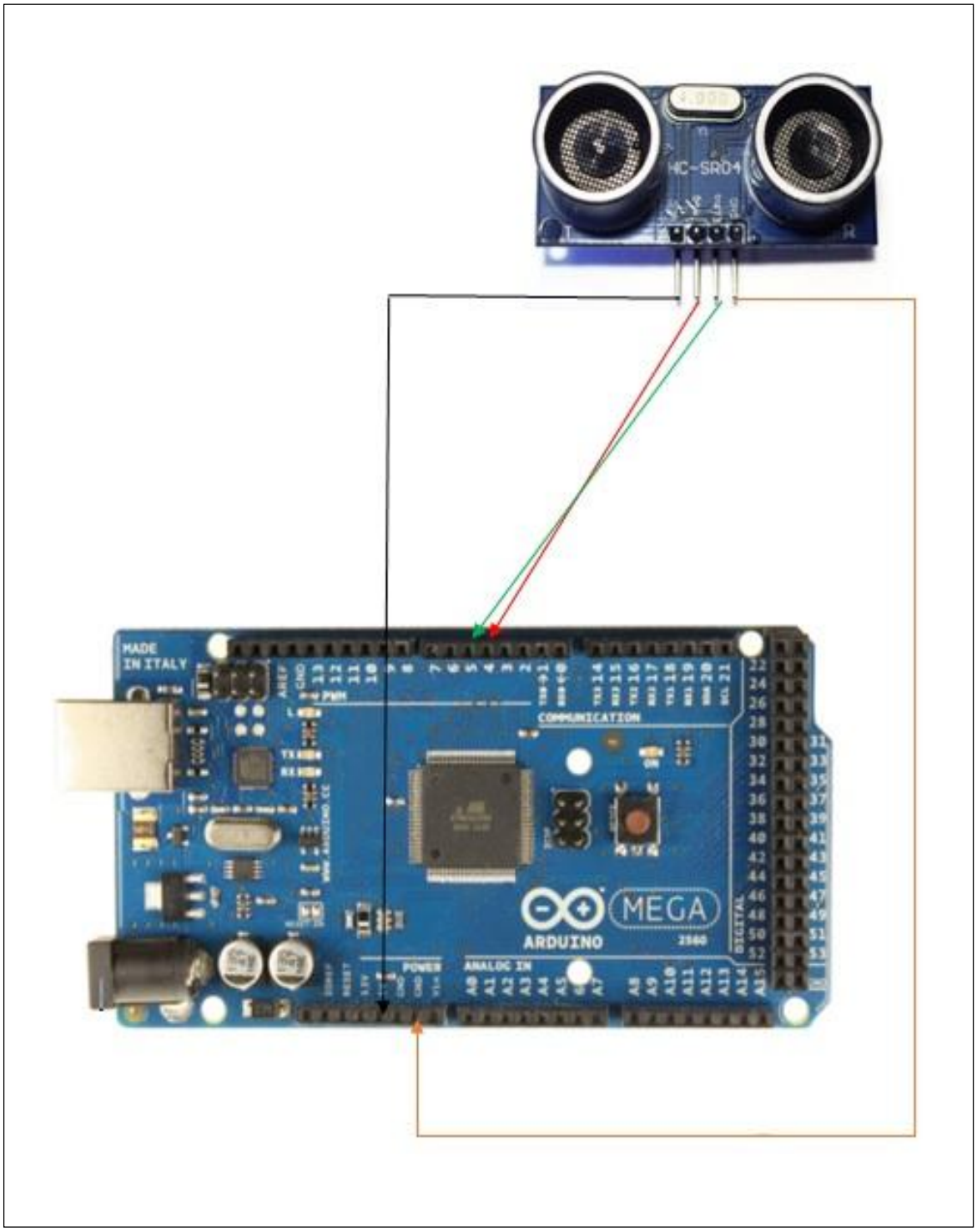


Figura 32. Conexión sensor ultrasonidos al Arduino

1.8. Firmware de la tarjeta Arduino

Cuando se abre el programa de Arduino, la ventana que se observa es igual a la de la figura 33. En ella es donde se programara el firmware del presente proyecto.

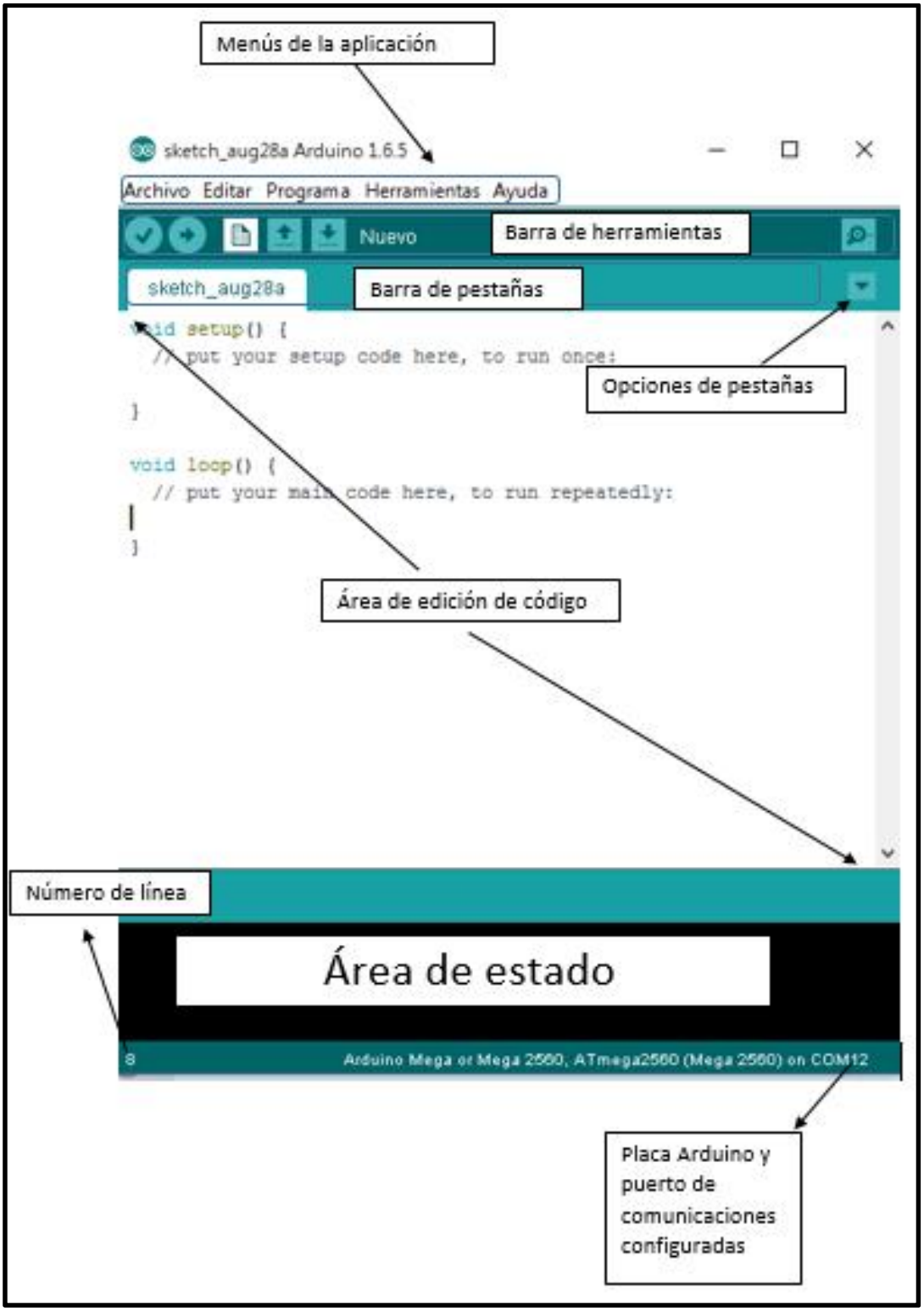


Figura 33. Software Arduino

Aunque antes de empezar a programar es conveniente conocer un poco el entorno. Para ello, lo primero que hay que saber es qué es un *sketch*, que es el nombre que usa Arduino para un programa. Es la unidad de código que se sube y se ejecuta en la placa

Arduino. El concepto de *sketch* sería equivalente a proyecto en otros entornos de programación.

Un *sketch* puede contener múltiples archivos (pestañas). Cuando un *sketch* es compilado, todas las pestañas serán concatenadas juntas para formar el archivo principal del *sketch*. Arduino puede utilizar librerías y código C/C++. Las pestañas *.c* o *.cpp* se compilan por separado y deberán ser incluidas en el *sketch* usando *#include*.

Es importante configurar correctamente la placa Arduino que se va a utilizar, y a través de qué puerto de comunicaciones estará conectada. Para tal efecto, debemos ir al menú *Herramientas*, escoger el submenú *Placa* e indicar la placa que usaremos, en el caso de este proyecto *Arduino Mega*. En el mismo menú *Herramientas*, submenú *Puerto*, se selecciona el puerto correspondiente, en este proyecto es el "COM10".

La barra de herramientas proporciona un acceso rápido a las siguientes funciones:

	Verificar/ Compilar	Se utiliza para compilar y así comprobar que el *sketch* es correcto antes de cargarlo a la placa Arduino.
	Cargar/ Programar	Cargar el *sketch* actual a la placa Arduino. Se debe asegurar que la placa y el puerto seleccionado es correcto antes de cargar el código.
	Nuevo	Creará un nuevo *sketch* para poder empezar a introducir código.
	Abrir	Presenta una lista de *sketch* almacenados en su entorno, así como una lista de *sketch* de ejemplo. Estos se pueden utilizar para comprobar que los periféricos funcionan correctamente.
	Guardar	Guarda el *sketch* de la ventana en el archivo de *sketch*. Una vez finalizado muestra un mensaje en el área de estado, debajo de la zona de edición de código.
	Monitor serie	Abre una ventana con un monitor del bus serie al que está conectado el Arduino. Esta es una herramienta muy útil, especialmente para depurar el código. El monitor muestra datos serie que se envían desde la placa Arduino y también puede enviar datos serie de vuelta a la placa Arduino. En la parte inferior izquierda del monitor serie se puede seleccionar la velocidad de transmisión de datos. La configuración por defecto es 9600 baud, para el proyecto será de 115200 baud.

Tabla 2. Elementos del software de Arduino

Todo programa Arduino debe contener, de manera obligatoria, al menos dos funciones básicas: *setup()* y *loop()*.

La función *setup()* se ejecuta cuando se inicia el programa. Se emplea para iniciar variables, establecer el estado de las entradas y salidas, inicializar librerías, etc. Esta función se ejecutará una única vez después de que se conecte la placa Arduino a la fuente de alimentación, o cuando se pulse el botón de reinicio de la placa.

Después de ejecutar la función *setup()*, la función *loop()* se ejecuta de manera consecutiva e ininterrumpida. Cuando se llega al final de esta función, se vuelve a ejecutar desde el principio hasta que se vuelva a reiniciar la placa. En el interior de esta función es donde se pone el código para controlar de forma activa la placa Arduino.

Los tipos de datos, su uso y declaración, es muy similar a otros lenguajes de programación, como el C o Java *(boolean, char, unsigned char, byte, int, unsigned int, float, double, void, etc.*). Además contiene una serie de constantes predefinidas en el lenguaje de Arduino que se usan para facilitar la lectura de los programas, y éstas son:

- HIGH | LOW → estado de un pin de E/S digital
- INPUT | OUTPUT → comportamiento de un pin de E/S digital
- true | false → estado de un resultado lógico.

Además Arduino tiene muchas funciones ya implementadas para simplificar la tarea de desarrollo para problemas comunes. Muchas de ellas están limitadas a ser usadas en pines concretos. Algunas de las funciones son:

E/S digital

- pinMode(pin, modo) → configura el pin a modo de entrada o salida.
- digitalWrite(pin,valor) → escritura digital.
- digitalRead(pin) → devuelve una lectura digital.

E/S Analógica

- analogReference(tipo) → configura el tipo de referencia analógica.
- analogRead(pin) → devuelve una lectura analógica.
- analogWrite(pin,valor) → "escritura analógica". Generará una onda cuadrada con el ciclo de trabajo que se le indique como parámetro (0-255 implica un ciclo de trabajo de 0 a 100%). La frecuencia de la señal PWM será de 490 Hz por diseño de Arduino.

E/S Avanzada

- tone(pin, frecuencia) → Genera una onda cuadrada de la frecuencia en Hz especificada. Opcionalmente se puede definir la duración del tono en ms.
- noTone(pin) → Deja de generar el tono en el pin especificado.
- shiftOut(pinDatos,pinReloj,ordenBits,valor) → Desplaza un byte de datos bit a bit a través del SPI.
- pulseIn(pin,value) → Devuelve la anchura de un pulso en microsegundos empezando a medir cuando el pin se encuentra al nivel definido en *value* (se suele utilizar con los ultrasonidos de 3 pines).

Tiempo

- millis() → Tiempo desde el arranque en ms.
- micros() → Tiempo desde el arranque en µs
- delay(ms) →Espera activa en ms.
- delayMicroseconds(us) → Espera activa en us(µs)

Cálculo

- map(value,fromLow,fromHigh,toLow,toHigh) →Devuelve el valor de un "re-mapeo" de un rango hacia otro. Por ejemplo, para realizar un cambio de escala de un valor de 0 a 1024 a un rango de 0 a 255.
- pow(base, exponente) → Devuelve el valor de un número elevado a otro número.

Interrupciones

- interrupts() → Habilita las interrupciones.
- noInterrupts() → Desactiva las interrupciones.

Interrupciones externas.

- attachInterrupt(inter,función,modo) → Con el parámetro "inter" se indica qué interrupción externa se va a configurar. Con el parámetro "modo" se indica el motivo que la va a disparar (CHANGE, LOW, RISING o FALLING) y con el parámetro "función", que va a ejecutarse al ser disparada
- detachInterrupt(inter) → Desactiva la interrupción inter.

Comunicación (estas dos son clases, con lo que no se pueden invocar directamente, sino que se debe invocar el método que se necesite).

- Serial
- Stream

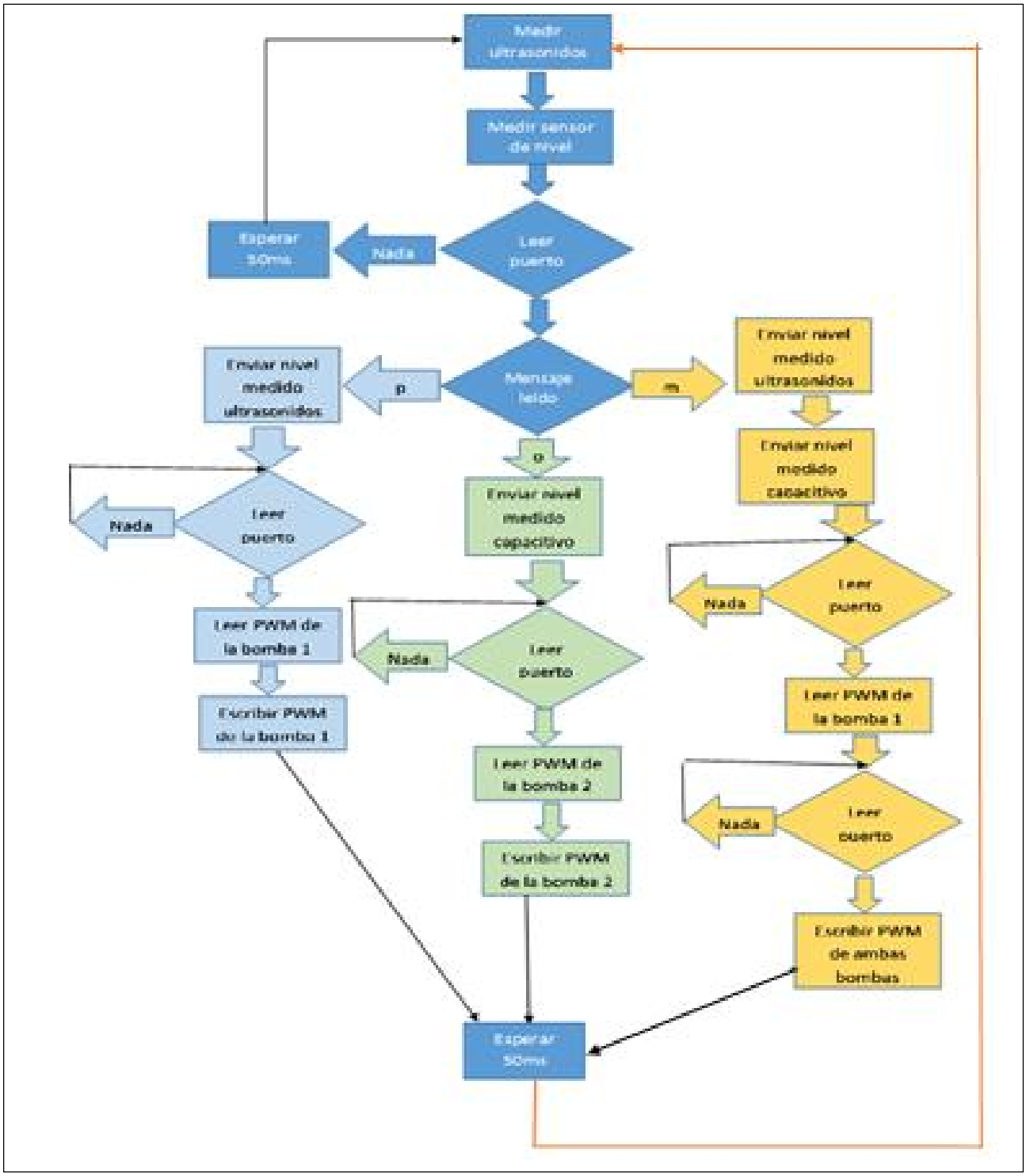


Figura 34. Esquema código del firmware

Una vez aprendido estas nociones básicas del software de Arduino, ya se puede empezar a programar. Aunque también se podría hablar de las librerías pero al haber muchas (de hecho cada día aparece una nueva librería) y no es algo que se use en todos los programas (aunque en el presente proyecto se use la *NewPing*), no se detallará en el presente documento, aunque se puede encontrar información en la página web de Arduino.

Primero se incluye la librería necesaria (*NewPing)*, para que funcionen todos los comandos, operaciones y funciones utilizadas. Se definen todas las variables globales a utilizar según el tipo que sean, y los pines donde se conectan la electrónica para así una mayor comprensión del código.

En la figura 35 está la librería definida, y los pines donde se ha conectado las bombas, el sensor ultrasonidos y el sensor de nivel. También está preparado para conectar dos caudalímetros. Con la última función la *NewPing sonar (TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);* se configura la librería.

```
#include <NewPing.h>

/* Definimos los pines a los que conectaremos los sensores             */
#define TRIGGER_PIN 4

#define ECHO_PIN 5

#define MAX_DISTANCE 200

const int Bomba1 = 11;

const int Bomba2 = 12;

const int Sonda = A0;

//#define CaudalimetroA 2

//#define CaudalimetroB 3

#define SOUND_SPEED 0.0171 // La mitad de la velocidad del sonido en el aire, medida en [mm/us]

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
```

Figura 35. Inicio Firmware

Y en la figura 36 se definen el resto de variables.

```
double microsegundos;

double distanciamm, distancia;

char mp;

unsigned char tension;

double tensionbomba2;

const double tfiltro = 10;

const double tfiltro1 = 5;

const double polofiltro = 0.9;

double pd, pd1;

double SensorValue, SensorValues;

double cauda, caudb, cuentas, cuentasant, caudala, caudalb, cuentasb, cuentasantb;
```

Figura 36. Definición constantes

A continuación, en la función *setup()* se definen los baudios, en este caso 115200.

```
void setup() {

  Serial.begin(115200);

}
```

Figura 37.Setup()

La función *loop()* se ejecuta tras haberse ejecutado la función *setup()*. Solo parará si se produce una interrupción del suministro eléctrico o si se carga un nuevo código sobre la placa. Al iniciar la función *loop()* (figura 38), se llamará a la función *ultrasonidos()* (figura 39), para realizar la medición del sensor ultrasonidos, con su respectivo filtro de 10 segundos, para suavizar el ruido. También al inicio de la función l*oop()* se hará la captura de la señal del sensor de nivel con su respectivo filtro de 5 segundos.

```
void loop() {

  ultrasonidos();

  SensorValue = analogRead(Sonda);

  pd1 = exp(-0.05 / tfiltro1);

  SensorValues = SensorValues * pd1 + (SensorValue / 20.0) * (1 - pd1);
```

Figura 38. Loop())

```
void ultrasonidos() {

  pd = exp(-0.05 / tfiltro);

  microsegundos = sonar.ping();

  distanciamm = microsegundos * SOUND_SPEED;

  distancia = distancia * pd + distanciamm * (1 - pd);

}
```

Figura 39.ultrasonidos()

A continuación, en función de la letra recibida por el puerto serie, hará unas acciones u otras (devolver el valor medido por el sensor ultrasonidos, devolver el valor medido por el sensor de nivel, o devolver ambos). Antes de finalizar el *loop()*, hay un *delay()* de 50 ms, que es para que no haya interferencias en las lecturas de los sensores y lean la señal correcta. Las operaciones que se realizarán en función del valor del puerto se observa en las siguientes 3 figuras.

```
if (mp == 'p') {

  /* Enviamos por el puerto serie el nivel medido en ultrasonidos

  Serial.println(distancia);

  /* Esperamos a tener algun mensaje en el puerto serie y lo leemos

  while (Serial.available() == 0);

  tension = Serial.read();

  /* Escribimos el valor de la tension que le debe llegar a la bomba

  analogWrite(Bomba2, 0);

  analogWrite(Bomba1, tension);

}
```

Figura 40. Depósito del sensor ultrasonidos

```
else if (mp == 'o') {

  /* Enviamos por el puerto serie el nivel medido en ultrasonidos          */

  Serial.println(SensorValues);

  /* Esperamos a tener algun mensaje en el puerto serie y lo leemos        */

  while (Serial.available() == 0);

  tensionbomba2 = Serial.read();

  /* Escribimos el valor de la tension que le debe llegar a la bomba       */

  analogWrite(Bomba1, 0);

  analogWrite(Bomba2, tensionbomba2);

}
```

Figura 41. Depósito del sensor de nivel capacitivo

```
else if (mp == 'm') {

  /* Enviamos por el puerto serie el nivel medido en ultrasonidos          */

  Serial.println(distancia);

  Serial.println(SensorValues);

  /* Esperamos a tener algun mensaje en el puerto serie y lo leemos        */

  while (Serial.available() == 0);

  tension = Serial.read();

  /* Esperamos a tener algun mensaje en el puerto serie y lo leemos        */

  while (Serial.available() == 0);

  tensionbomba2 = Serial.read();

  /* Escribimos el valor de la tension que le debe llegar a la bomba       */

  analogWrite(Bomba1, tension);

  analogWrite(Bomba2, tensionbomba2);

}
```

Figura 42. Ambos depósitos.

1.9. Software de control: interfaz gráfica

Una interfaz gráfica de usuario (GUI), es una interfaz construida a través de objetos gráficos, tales como menús, botones, listas y barras de desplazamiento.

Esos objetos formarán una interfaz en el momento que se defina una acción a realizar cuando se produzca algún cambio o acción en los mismos (una pulsación de ratón, arrastrar un objeto con el puntero del ratón, etc.).

Para diseñar una interfaz gráfica efectiva, se deben escoger los objetos gráficos adecuados que se desea visualizar en la pantalla, distribuirlos y colocarlos de una manera lógica para que la interfaz sea fácil de usar y determinar las acciones que se realizarán al ser activados estos objetos gráficos por el usuario.

Matlab permite desarrollar y definir un conjunto de elementos (botones, menús, ventanas…) que permiten utilizar de manera fácil e intuitiva, programas realizados en este entorno. Los objetos de una GUI en Matlab se dividen en dos clases:

- Controles

- Menús

Los gráficos de Matlab tienen una estructura jerárquica, formada por objetos de distintos tipos.

En la siguiente figura podemos observar la forma que presenta esta jerarquía:

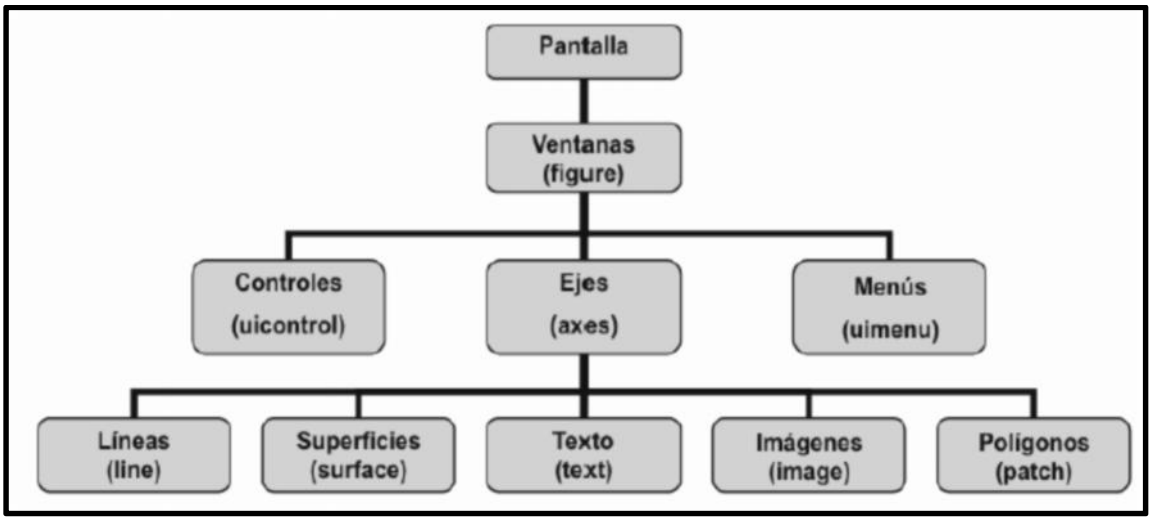


Figura 43. Jerarquía GUIDE Matlab

Tal y como se indica en la figura anterior, la pantalla es el objeto más general y del que dependen todos los demás y únicamente puede haber un objeto pantalla. Una pantalla puede contener una o más ventanas (*figures*). A su vez, cada una de las ventanas puede tener uno o más ejes de coordenadas (*axes*) con los que representar otros objetos de más bajo nivel. Una ventana puede tener también controles (*uicontrols*), tales como botones, barras de desplazamientos, botones de selección o de opción, etc. y menús (*uimenus*). Finalmente, los ejes pueden contener los cinco tipos de elementos gráficos que permite Matlab y que se indican en la figura x (*line, Surface, text, image y patch*).

De todo esto se deduce que en el interface gráfico de Matlab hay objetos padres e hijos. Cuando se borra un objeto de Matlab, automáticamente también se borran todos sus descendientes. El comando a utilizar será: *delete* (*identificador*).

Además, cada uno de los objetos de Matlab tiene un identificador único denominado *handle*. Si una pantalla tiene muchos objetos, cada uno de ellos tendrá asignado un *handle*. El objeto raíz (Pantalla), es siempre único y su identificador es cero. El identificador de las ventanas es un entero, que aparece en la barra de nombre de dicha ventana. Los identificadores de otros elementos gráficos, son número flotantes, que pueden ser obtenidos como valor de retorno y almacenados en variables de Matlab.

Así mismo, Matlab puede tener varias ventanas abiertas, pero siempre hay una única activa. También una ventana puede tener varios ejes, pero sólo uno activo. Matlab dibuja en los ejes activos de la ventana activa.

Se pueden obtener los identificadores de la ventana activa, de los ejes activos y del objeto activos con los siguientes comandos: *gcf (get current figure), gca (get current axes) y gco (get current object).*

La herramienta GUIDE (GUI Development Environment) es una herramienta de Matlab para el desarrollo de aplicaciones GUI bajo un entorno gráfico. GUIDE facilita al desarrollador, un conjunto de herramientas de uso sencillo que simplifican mucho el diseño y la programación de GUIs.

Actualmente, las herramientas que nos ofrece GUIDE, permiten además de diseñar la ventana, generar un archivo de extensión *.m* que contiene el código para controlar el lanzamiento e inicialización de la aplicación GUI.

Se puede abrir GUIDE desde el Lanch Path de Matlab haciendo doble clic sobre GUIDE o tecleando desde la línea de comandos *guide*, abriendo la siguiente pantalla, donde se seleccionará crear un nuevo GUI o abrir uno ya existente.

Para creara una nueva GUI, seleccionaremos la opción “Blank GUI (Default)” Y nos aparecerá una ventana que servirá como editor de nuestro nuevo GUI. En ella podremos poner los objetos gráficos que necesitemos para nuestra aplicación.

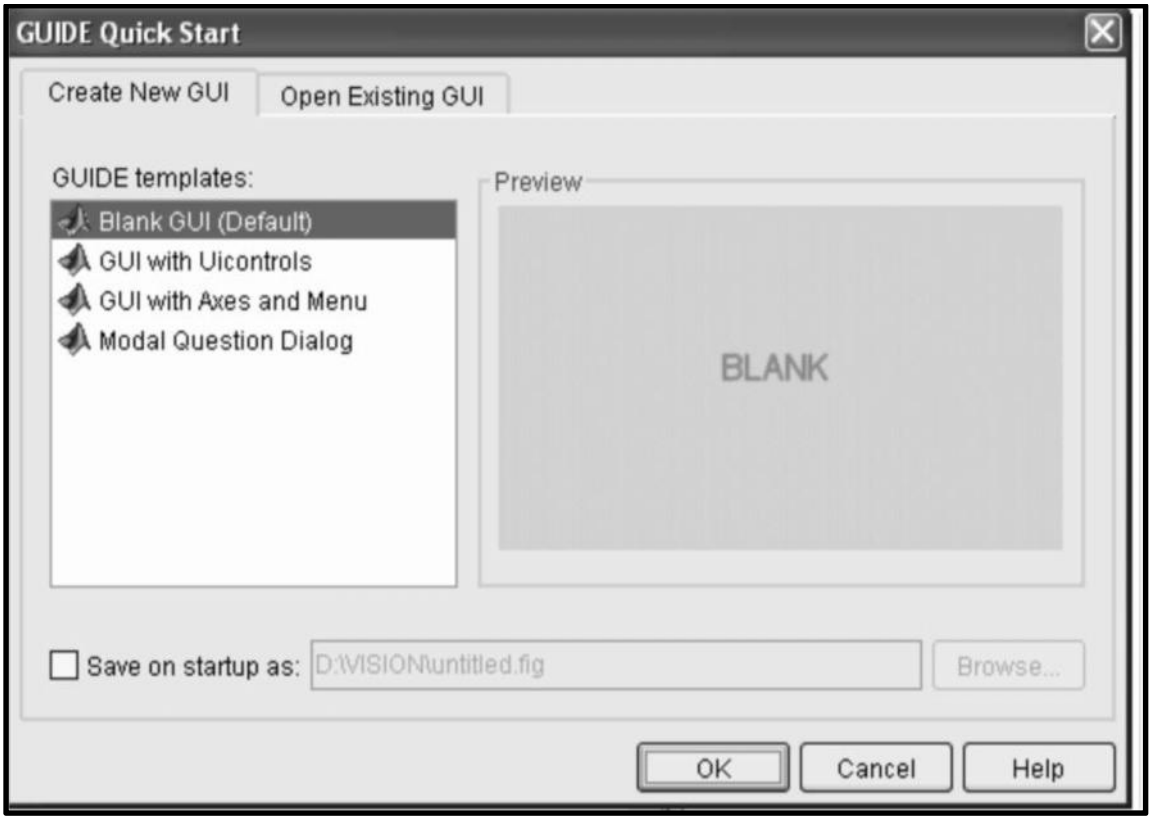


Figura 44.Abrir GUIDE Matlab

El desarrollo de una aplicación GUI, implica diseñar la ventana (colocando cada uno de los objetos gráficos dentro del área de trabajo) y programar los componentes para definir cuáles son las funciones que realizará cada uno de los objetos cuando se actúe sobre ellos.

A continuación, se muestra el formato del área de trabajo donde se va a desarrollar la nueva GUI.

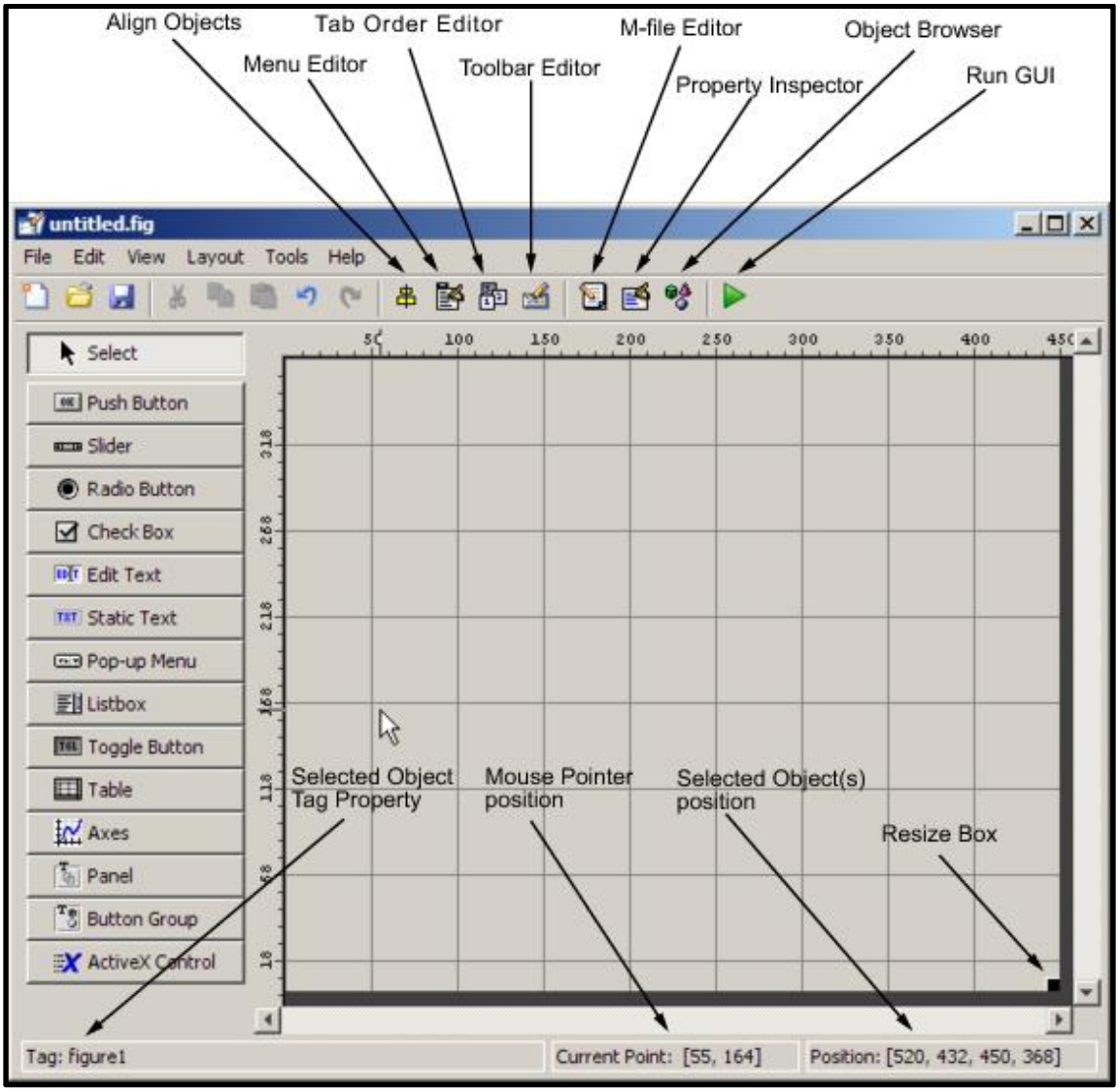


Figura 45. GUIDE

Arrancando la herramienta GUIDE, tenemos la posibilidad de configurar las opciones del GUI en el menú Tools opción GUI Options.

Una vez que se configura la herramienta GUIDE, se procede a crear en el *layout*, los objetos gráficos que compondrán el GUI de muestra aplicación. Se muestran a continuación las opciones más empleadas en el editor de *layout*:

- *Herramienta de Alineación (Alingment Tool)*

Esta herramienta permite ajustar con respecto a una referencia, la posición de un objeto dentro del *layout*. Las operaciones de alineación se realizarán sobre los objetos seleccionados, una vez pulsado "*Apply*". Esta herramienta proporciona dos tipos de alineaciones, tanto en horizontal como en vertical.

- *Buscador de Objetos (Object Browser)*

Esta herramienta permite mostrar el orden que se ha seguido a la hora de ir insertando los distintos controles (uicontrols) u objetos gráficos. Es muy útil cuando se tiene un GUI muy complejo (con muchos objetos insertados). Además, haciendo doble clic con el ratón sobre cualquier elemento, se accede a sus características propias, el *Property Inspector*.

- *Menú Editor*

Esta herramienta permite diseñar los menús que contendrá el GUI cuando se ejecute. Matlab es capaz de crear dos tipos de menús:

  - *Menú bar:* Son los menús típicos de las barras de herramienta (Archivo, Edición, Insertar,…).
  - *Context Menu*: Son los menús que saldrán en cada objeto al pulsar el botón derecho del ratón. La forma de crearlos es igual a los anteriores, salvo que hay que asociar los menús al elemento gráfico en cuestión.

- *Uicontrols*

Los *uicontrols* son los denominados controles de usuarios del Interface (*User Interfaces Controls*). En el área de *Layout* se pueden insertar los siguientes *uicontrols*:

  - *Push Buttons:* Permiten pasar de un estado al siguiente en la aplicación, cuando el usuario pulsa sobre ellos (bien con el ratón o bien con la tecla de tabulador y el *enter*). Son los típicos botones de Activación: Cancelar, OK…

  - *Toggle Buttons:* Generan una acción e indican un estado binario (on/off). Para ejecutar la *callback* asociada a este botón, se necesita leer el valor con el comando: get(gcbo, 'value').

  - *Radio Buttons:* Este tipo de botones sólo tiene dos estados: activado o desactivado.

  - *Edit Text:* Sirve para modificar y escribir cadenas de texto.

- *Static Text:* Sirve para insertar un texto fijo en controles, botones, etc. Dicho texto no se puede modificar en ejecución.

- *Check Boxes:* Cajas de selección que se activan pulsando sobre el cuadrado y aparece un *stick*. Tiene dos posibles valores: seleccionado o no seleccionado.

- *Sliders:* Se suelen emplear para que el usuario de la aplicación selecciones valores numéricos. Aparece un abarra, que puede tener una orientación vertical u horizontal; dicha orientación se establece con el *property inspector*. El *slider* tiene asignadas cuatro variables*: value, max, min y sliderstep.*

- *List Boxes:* Aparece una lista con varias opciones, donde el usuario deberá seleccionar una de ellas.

- *Pop-Up Menús:* Listas desplegables donde se muestran varias opciones para que el usuario de la aplicación pueda seleccionar alguna/s de ellas.

- *Axes y figures:* Permiten insertar ejes y figuras en el GUI

Llegados a este punto se debe mencionar una parte muy importante de la creación de una interfaz GUI que son las funciones *callbacks*.

Cada objeto gráfico y/o figura que se crea en el GUI, lleva asociado un *callback* que permite definir la acción que llevará a cabo la aplicación cuando se actúe sobre dicho objeto.

Existen dos tipos de *callback*, según se trate de objetos gráficos o figuras:

- *Callbacks para objetos gráficos*

Todos los objetos gráficos que se añaden al GUI, tienen una serie de propiedades que permiten al programador definir funciones *callbacks* para asociarlas a ellas.

- *Callbacks para figuras*

Las figuras o ventanas tienen otras propiedades que pueden asociarse con sus respectivas funciones *callbacks* tras las correspondientes acciones del usuario.

Cuando se guarda la aplicación GUI que hemos creado, se generan automáticamente dos archivos: *archivo.FIG y archivo.M*.

El *archivo.FIG* contiene la descripción del GUI que se ha diseñado. En él se guarda la figura con los objetos gráficos insertados en ella, para su posterior modificación o uso. También contiene la información relativa a las propiedades de cada objeto.

El *archivo.M* es un fichero que contiene las funciones necesarias para controlar y ejecutar el GUI y las funciones *callabacks*. En este fichero, aparece una función *callback* para cada objeto gráfico diseñado en el *archivo.FIG*. El programador, dependiendo de lo que quiera que haga dicho objeto, introducirá el código necesario.

Por último queda comentar las estructuras *handles*: concepto importante para el uso de esta herramienta. Como se ha explicado anteriormente, la herramienta GUIDE genera automáticamente código en un fichero Matlab con extensión *.m*, donde se encuentra las funciones que controlan y ejecutan el GUI y las funciones *callabacks.*

Dicho código está basado en la estructura *handles* y en él se puede añadir el código para ejecutar los hilos de GUI. La estructura *handles* es pasada como una entrada a cada *callback.* La estructura *handles* puede usarse para:

- Compartir datos entre *callbacks*
- Acceder la data en el GUI

Una vez aprendido cómo funciona la GUI ya nos disponemos a realizar la interfaz necesaria para el presente proyecto. Se decide que la interfaz tenga una ventana principal, donde se podrá establecer la conexión con Arduino por el puerto serie y deshacerla, y escoger qué operación se quiere realizar, cuando se haya escogido la operación a realizar, se entrara en una de las 4 ventanas secundarias.

En la ventana principal, se encuentran varios de los elementos mencionados anteriormente. Desde dos *push buttoms*, un *edit text*, un *list box*, dos *static text* y un *axes*. En esta ventana se puede escoger a qué subventana acceder, desde el *list box*, según la operación que se desee realizar. Éstas pueden ser: Sistema control bucle abierto, Sistema control bucle cerrado, Multivariable bucle abierto y Multivariable bucle cerrado. Además, desde el botón *Conectar* se establece la comunicación al dispositivo conectado en el puerto serie, el puerto al que se conecte será el indicado en el *edit text* situado justo debajo del botón (hay que escribir "COM" y el número que le corresponda, de poner solo el número, daría un error), en el momento que se establezca la conexión, el botón *Conectar* pasará a tener el fondo verde, y en lugar de *Conectar* pondrá *Conectado.* Finalmente desde el botón *Salir* cerraremos la conexión con el puerto serie, y se cerrará la ventana. Añadir que si entre los archivos no se encuentra el archivo "logo1" la aplicación no se abrirá, o en caso de abrirse produciría un error y no funcionaría. A continuación en la figura 46 se muestra una imagen de la ventana.

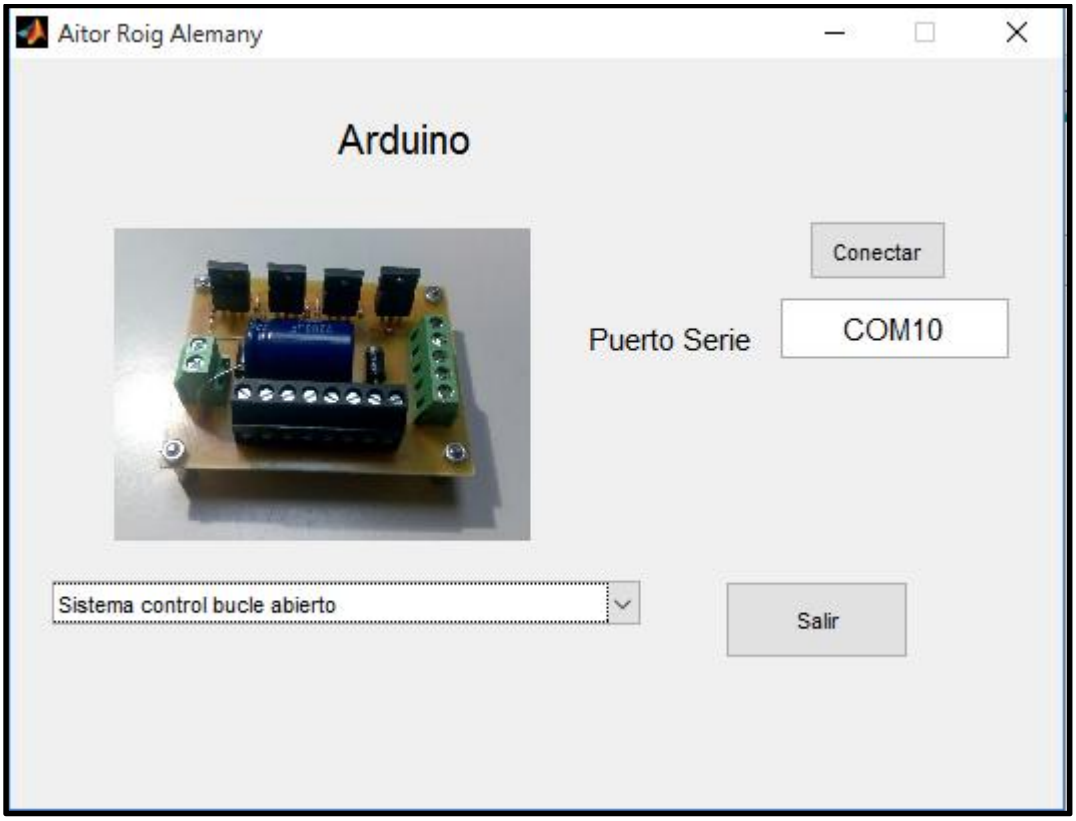


Figura 46. Ventana principal

Si la operación a realizar es la de *sistema de control de bucle abierto* la subventana que aparecerá será la de la figura 47. En esta ventana aparece un elemento que no aparecía en la ventana principal, se trata del *slider,* con el que moviendo la deslizadera, se enviará un valor u otro por el puerto serie, y es el que indicará el valor de la tensión que le llegará a la bomba. Se decidió representar los valores en las unidades del PWM, ya que se pueden conectar bombas cuya tensión sea distinta a la de las actuales, y como el PWM es un valor proporcional a la tensión que le llega, por eso se escogió esta nomenclatura. A modo de ejemplo: si se tiene la placa que controla la bomba conectada a 15V y se le envía un PWM de 190 (valor por defecto en el *slider*), la tensión que le llegará a la bomba será de 11,18V, en este ejemplo, en el que la placa está a 15V, la relación que existe de PWM con la tensión que le llegará a la bomba es: *Valor PWM*0.0588* el resultado de esta operación será la tensión que le llegará a la bomba. Para otras futuras tensiones la tensión que le llegaría a la bomba sería: $\frac{Valor\,PWM\,\cdot\,Tensión\,alimentación}{255}$.

A parte de enviar el valor del PWM por el *slider* también se puede enviar por un *edit box* situado al lado de éste, para así enviar valores exactos. En el *static text* de la derecha del *slider* se verá reflejado el PWM que se enviará por el puerto serie. Luego con los botones *Depósito 1* y *Depósito 2* se escogerá en qué depósito se quiere observar el nivel del líquido. En función de en qué depósito se esté mostrando el nivel del líquido, el botón *Depósito 1* estará en verde, en el caso de observar el nivel de este depósito o en rojo, en caso de no mostrar este nivel, y viceversa.

Finalmente, los 3 botones que aparecen, tratan la adquisición de datos. El botón *Tomar medidas* es el que llama a una función que mediante el *Timer* adquiere los niveles del líquido y los representa (además de enviarle el valor de PWM). Cuando se esté adquiriendo datos, este botón aparecerá en verde y en lugar de *Tomar medidas* pondrá *Adquiriendo datos*. Además, mientras se estén tomando las medidas del nivel, se abrirá automáticamente un *figure* donde se irán representando las curvas, tanto la de nivel del líquido como el valor del PWM enviado a la placa controladora de las bombas. El *Timer* funcionará hasta que se pulse el botón *Finalizar Timer*. Este botón además de finalizar el *Timer*, hará que el botón *Tomar medidas* vuelva a su estado inicial. Y por último, el botón de *Guardar Datos* irá guardando los datos adquiridos y los enviados por el puerto, en cada instante.

El botón *Guardar Datos* se puede pulsar cuando funcione el *Timer* o cuando ya se haya parado éste. El archivo se guardará en la carpeta donde esté el código con el nombre: *sistemacontrol_bucleabierto.txt*. Se recomienda cada vez que se guarde un archivo, ir a la carpeta y hacer una copia de seguridad, por si se requiere guardar datos nuevos, no perder estos datos.

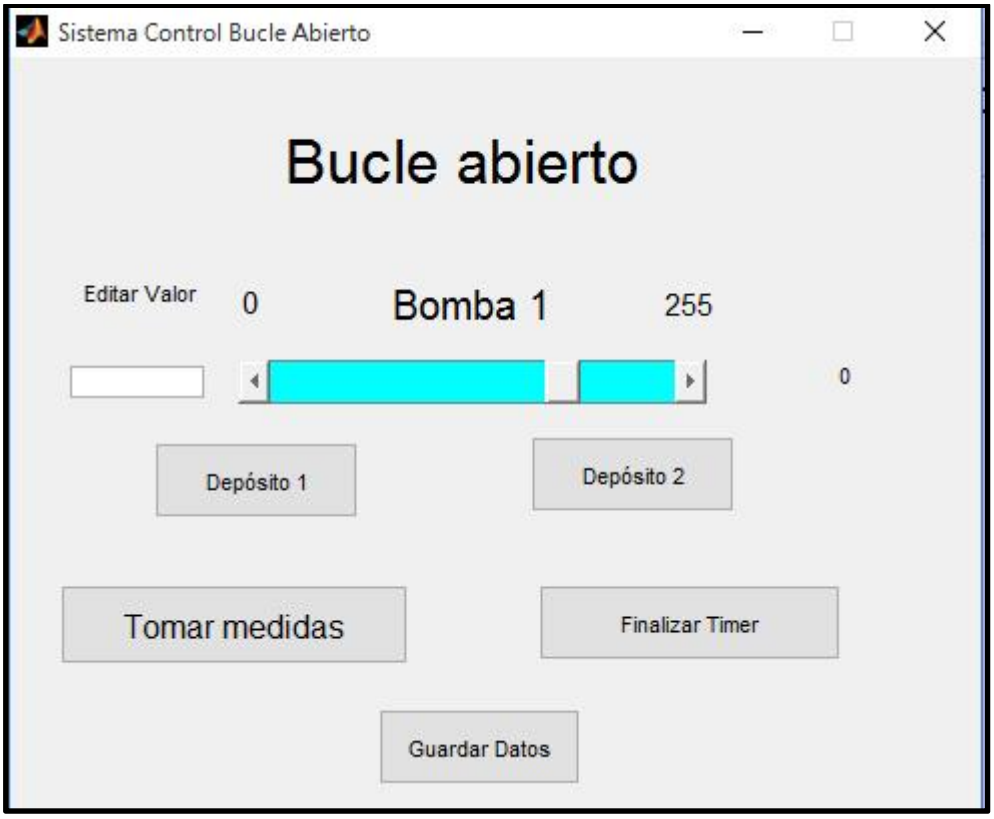


Figura 47.Ventana sistema control bucle abierto

Si la operación a realizar es la de *sistema de control de bucle cerrado* la subventana que aparecerá será la de la figura 48. En esta ventana aparece un *slider* como en la anterior ventana explicada. Pero este *slider*, en lugar de enviar por el puerto serie el valor de PWM, define el nivel al que se quiere que llegue el líquido, y a través de un algoritmo de control PID, envía por el puerto serie el valor de PWM, el cuál irá variando constantemente, para que el nivel del líquido no supere el nivel deseado.

A continuación, viene otra de las diferencias respecto a la anterior ventana, y es que, en este cas,o no se selecciona el depósito del cual queremos observar el nivel, sino que con varios *edit text* se introduce los parámetros para calcular el PID del sistema de control. La aplicación lleva unos datos por defecto a modo de ejemplo, estos datos

corresponden a los resultados obtenidos en el apartado 1.12, sitio donde se explicará de dónde salen estos valores y como calcularlos.

Finalmente, los 3 botones que aparecen, tratan la adquisición de datos y son equivalentes a los de la ventana de *sistema control bucle abierto*. La única diferencia se encuentra en el nombre del archivo que será *ultrasonidos_con_bomba_contrario_control_cerrado.txt* donde se guardaran los datos del nivel del líquido y del valor del PWM. En este caso también se recomienda que cada vez que se guarde un archivo, ir a la carpeta y hacer una copia de seguridad, por si se requiere guardar datos nuevos, no perder estos datos.

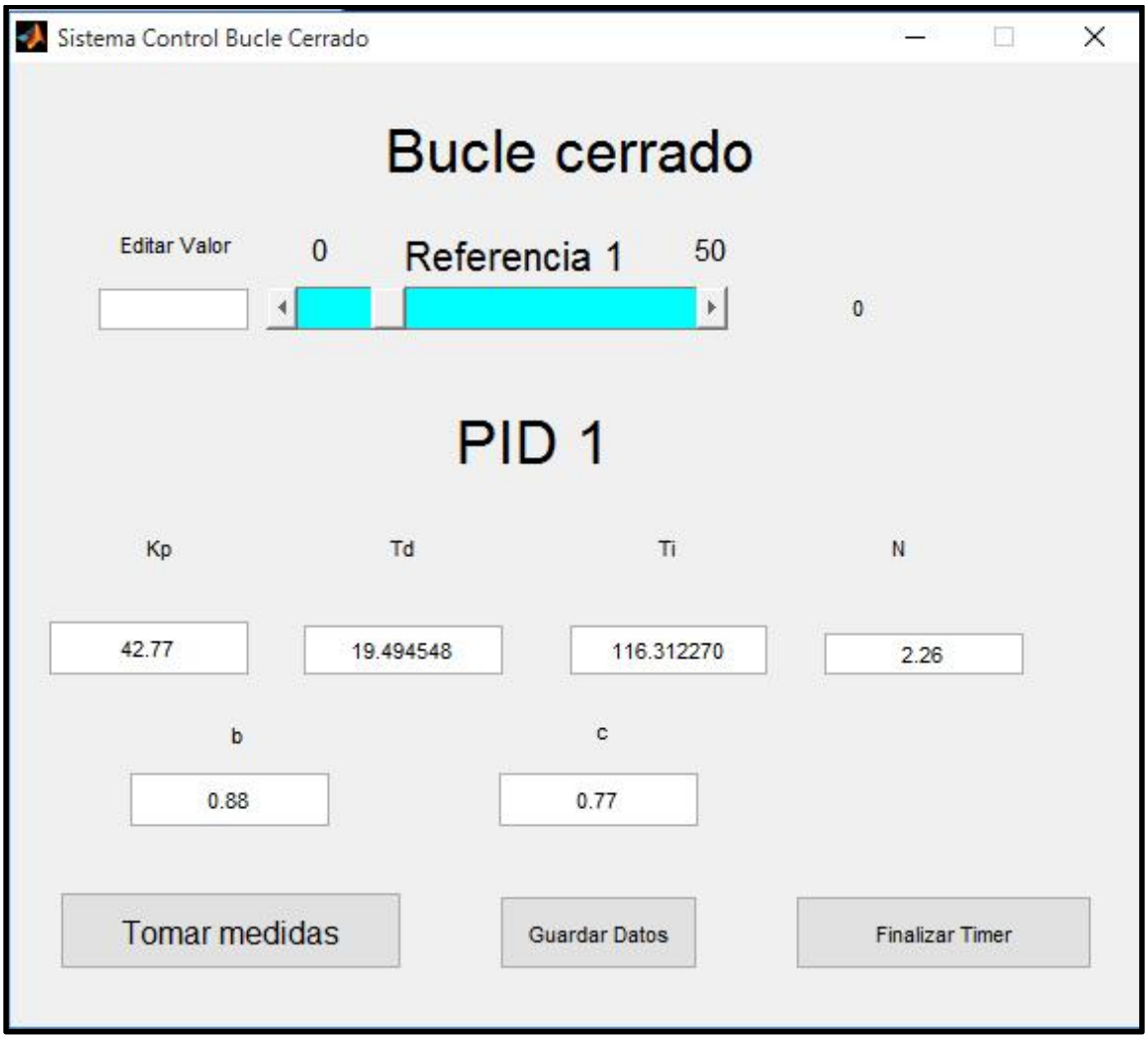


Figura 48.Ventana sistema control bucle cerrado

Para acceder a la siguiente ventana la opción que se debe haber escogido es la *Multivariable de bucle abierto*, al escoger esta opción la ventana que nos aparecerá será como la de la figura 49.

Esta ventana es muy semejante a la de *Sistema de control de bucle abierto* ya que el objetivo de ambas es el mismo, proporcionar un PWM a la placa controladora de las bombas y representar el nivel del líquido. Pero esta ventana, a diferencia de la otra, envía un PWM para cada una de las dos bombas, y se mide el nivel en ambos depósitos a la vez. Para ello, en lugar de utilizar solo un *slider* y un *edit box,* se necesita la ayuda de dos, uno para cada bomba. La *bomba 1* es la del sistema donde se ha colocado el sensor de nivel, y la *bomba 2* del sistema donde se ha colocado el sensor ultrasonidos.

Finalmente, los 3 botones que aparecen, tratan la adquisición de datos y son equivalentes a las ventanas anteriores. La única diferencia se encuentra en el número de archivos y en el nombre de éstos, en este caso, hay 4 archivos, ya que, para hacer el cálculo del valor de PWM correctamente se necesitan los datos de cada bomba con su sensor y con el sensor contrario, (esto se explicará en los puntos 1.10 i 1.11).

El nombre de los archivos será: *ultrasonidos_con_su_bomba.txt* (donde se almacenará el nivel medido por el sensor de ultrasonidos con respecto a su bomba), *ultrasonidos_con_bomba_contrario.txt* (en este caso se almacenará el ultrasonidos con la bomba del otro sistema de depósitos), *sensor_con_su_bomba.txt* (donde se almacenará el nivel medido por el sensor de nivel con respecto a su bomba) y finalmente *sensor_con_bomba_contrario.txt* (en este caso se almacenará el sensor de nivel con la bomba del otro sistema de depósitos) .

En este caso también se recomienda que cada vez que se guarde un archivo, ir a la carpeta y hacer una copia de seguridad, por si se requiere guardar datos nuevos, no perder estos datos.

Figura 49.Ventana multivariable bucle abierto

Para terminar, la última ventana (figura 50), accedemos a ella, cuando se escoge la opción de *Multivariable de bucle cerrado*. Esta ventana se asemeja a la ventana de *Sistema de control de bucle cerrado* pero con unas pequeñas diferencias. Una de las diferencias es que como se actuará sobre las dos bombas, se necesita, la referencia deseada en ambos depósitos, y por tanto, se deberá calcular el sistema de control PID para las dos bombas. Esta ventana, al igual que su semejante, tiene los datos por defecto a modo de ejemplo para calcular los dos PID, estos datos corresponden a los resultados obtenidos en el apartado 1.13, sitio donde se explicará de dónde salen estos valores y cómo calcularlos.

Uno de los nuevos campos que aparecen es los *edit box* para poner los valores de k_{21} y k_{12} valores que si la válvula que une ambos sistemas de depósitos está abierta, influirán en el cálculo del sistema de control. Para decirle al programa si la válvula está abierta o cerrada, es decir, si los sistemas de depósitos están conectados o no, se utiliza uno de los elementos de la *GUIDE* que aún no se había utilizado hasta el momento, el

check box, el cual si está activo querrá decir que está la válvula abierta, por lo tanto, si esta desactivado, significará que está cerrada.

Finalmente los 3 botones que aparecen, tratan la adquisición de datos y son equivalentes a las ventanas anteriores, especialmente de la última, ya que los archivos son los mismos aunque modificando el nombre para que no se confundan con los de la ventana anterior. El nombre de estos es: *ultrasonidos_con_su_bomba_cerrado.txt*, *ultrasonidos_con_bomba_contrario_cerrado.txt*, *sensor_con_su_bomba_cerrado.txt* y finalmente *sensor_con_bomba_contrario_cerrado.txt*. En este caso también se recomienda cada vez que se guarde un archivo, ir a la carpeta y hacer una copia de seguridad, por si se requiere guardar datos nuevos, no perder estos datos.

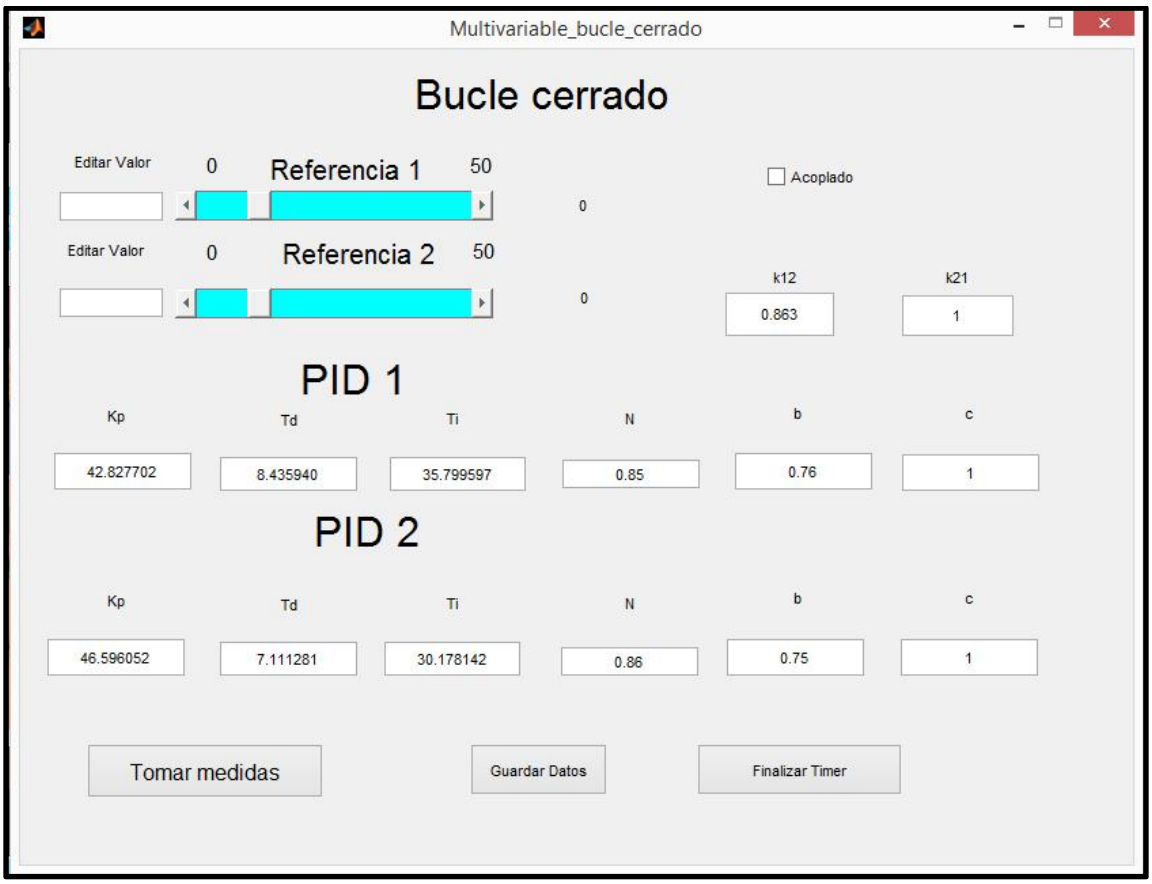


Figura 50.Ventana multivariable bucle cerrado

A continuación se muestra un esquema de cómo está distribuida la interfaz gráfica, con los archivos necesarios.

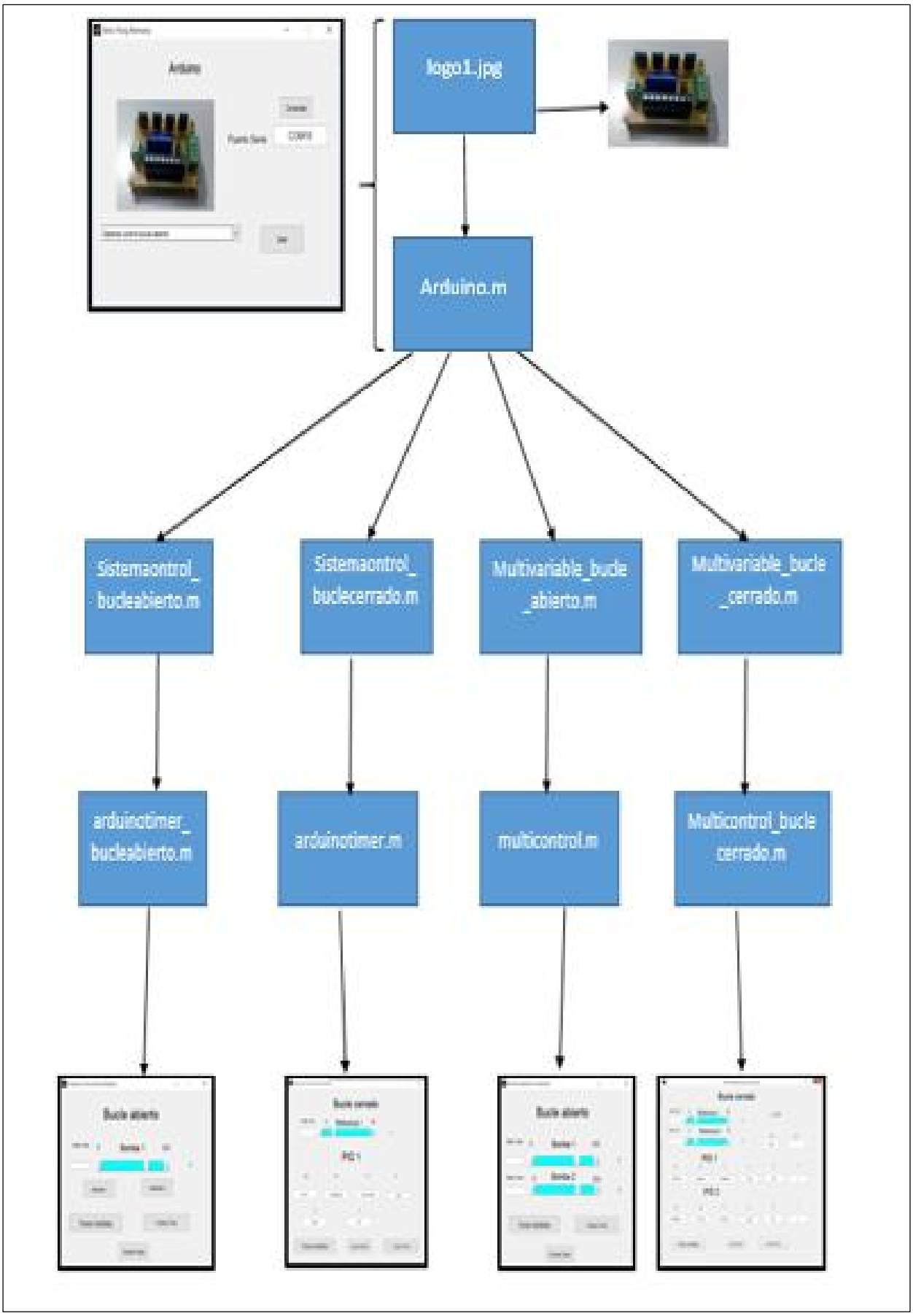


Figura 51. Esquema interfaz gráfica

1.10. Control PID de sistema de una entrada y una salida.

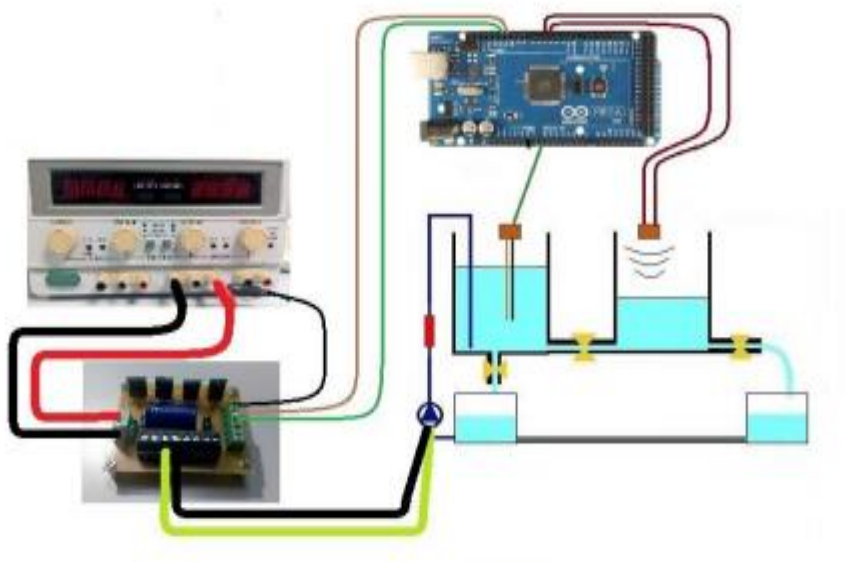


Figura 52. Sistema de una entrada y una salida

En primer lugar se debe hacer un experimento del sistema en bucle abierto, (con la ventana de *Sistema Control de Bucle Abierto*). En la ventana se pondrá primero un PWM de 100, cuando se observe que se ha estabilizado, se hacen dos escalones, el primero a 110, y finalmente cuando se haya estabilizado, el escalón será subir el PWM de 110 a 120, obteniendo un resultado como el de la figura 53.

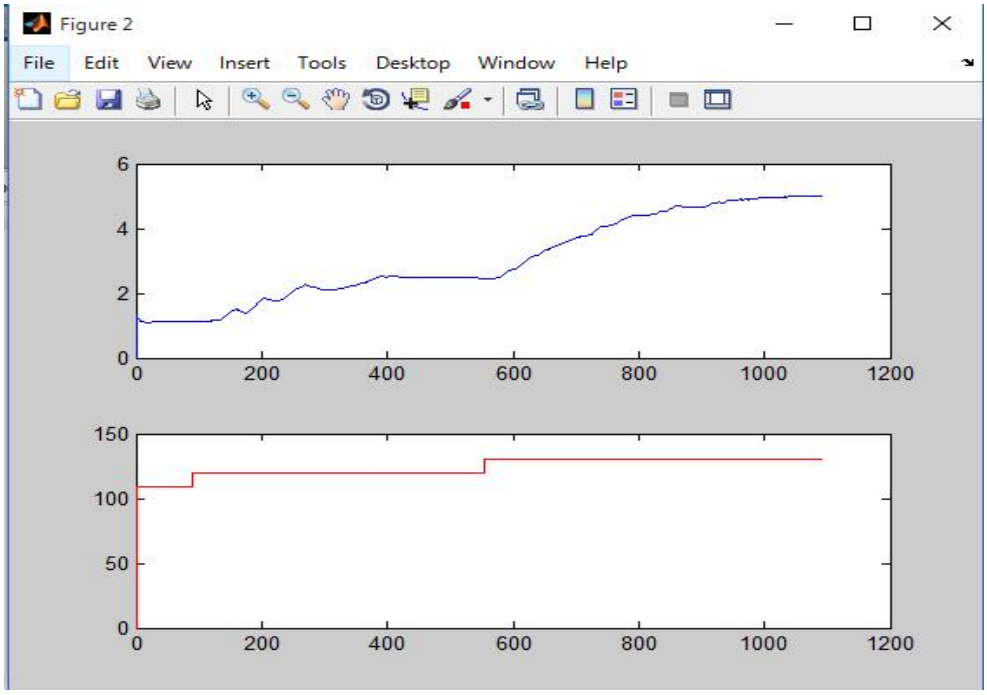


Figura 53. Resultados 1

Una vez obtenida la curva de la figura anterior, se almacenan los datos en un fichero en formato *.txt*. El archivo se abrirá posteriormente con la herramienta *ejs_model_identescalon23.jar* (Aplicación de identificación).

En esta aplicación en la pestaña *Carga_de_datos_experimentales* se cargarán los datos almacenados con la ayuda del botón *Cargar datos experimentales*. Una vez hecho esto se escoge el escalón que se quiera analizar, en este caso el escalón de entre 110 y 120. Para ello se pone la línea azul al principio del escalón y la línea roja, cuando se ha estabilizado.

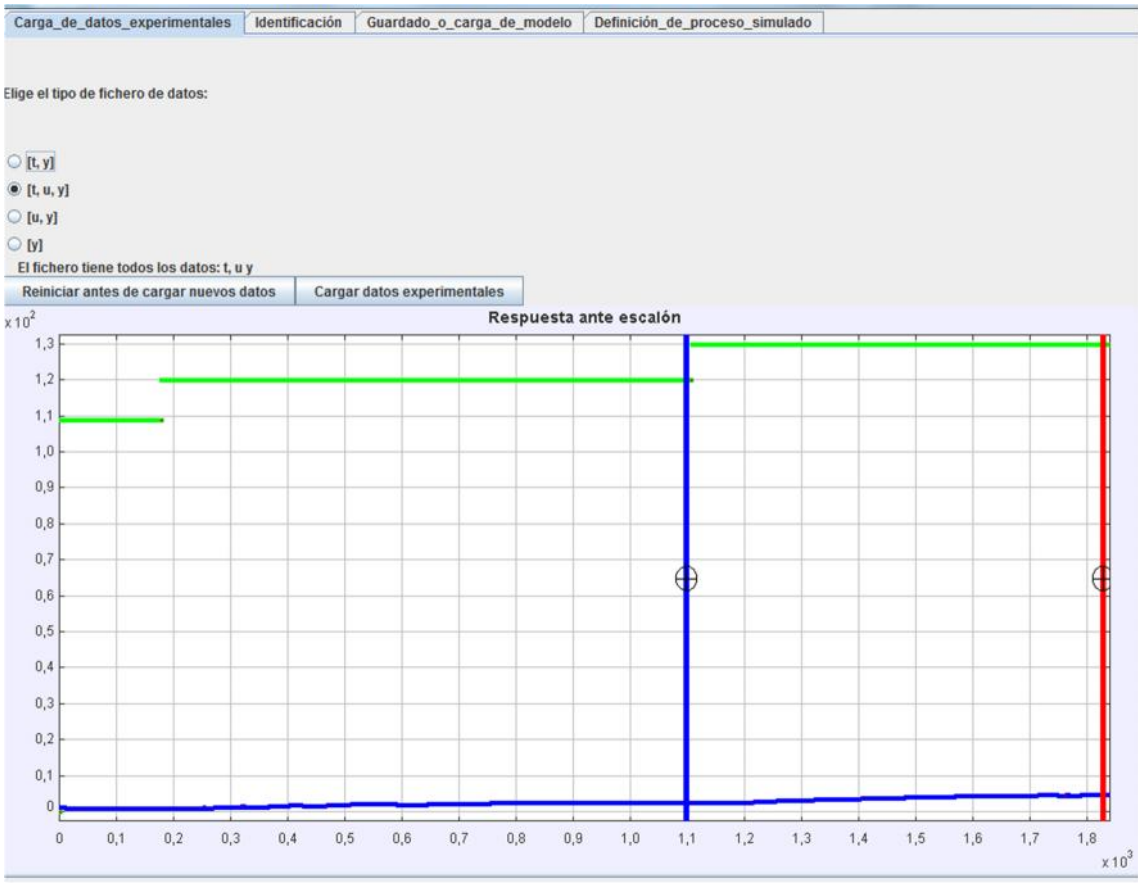


Figura 54. Escalón escogido 1

El siguiente paso, es ir a la pestaña de *Identificación*, en la cual se identificará la función de transferencia. Se activarán los polos que tenga, en este caso 2, y con las deslizaderas de la *K*, la *t1* y la *t2*, hasta que el error sea el mínimo. Para ello el usuario también se puede ayudar del botón *Ajuste 1 iteración*, o el *check box* de *Ajuste n interacciones*. El resultado se puede observar en la figura 55.

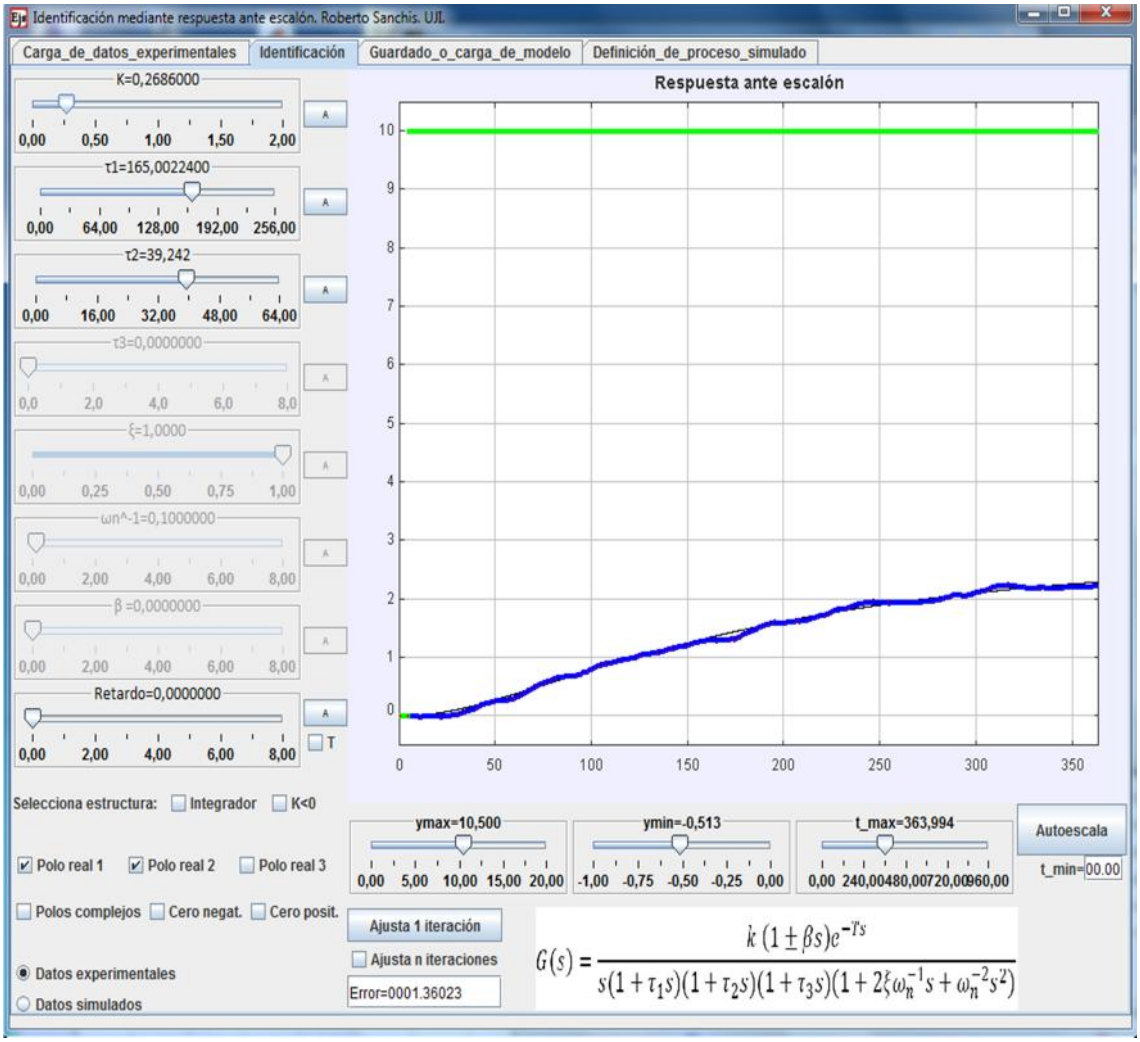


Figura 55. Función de transferencia

La función de transferencia obtenida es la siguiente.

$$G = \frac{0.269}{(1 + 165.004 \cdot s) + (1 + 39.242 \cdot s)}$$

Finalmente, en la pestaña *Guardado_o_carga_de_modelo* se guarda el modelo obtenido, en un fichero para usarlo en el siguiente programa, donde se realizará el diseño del controlador PID.

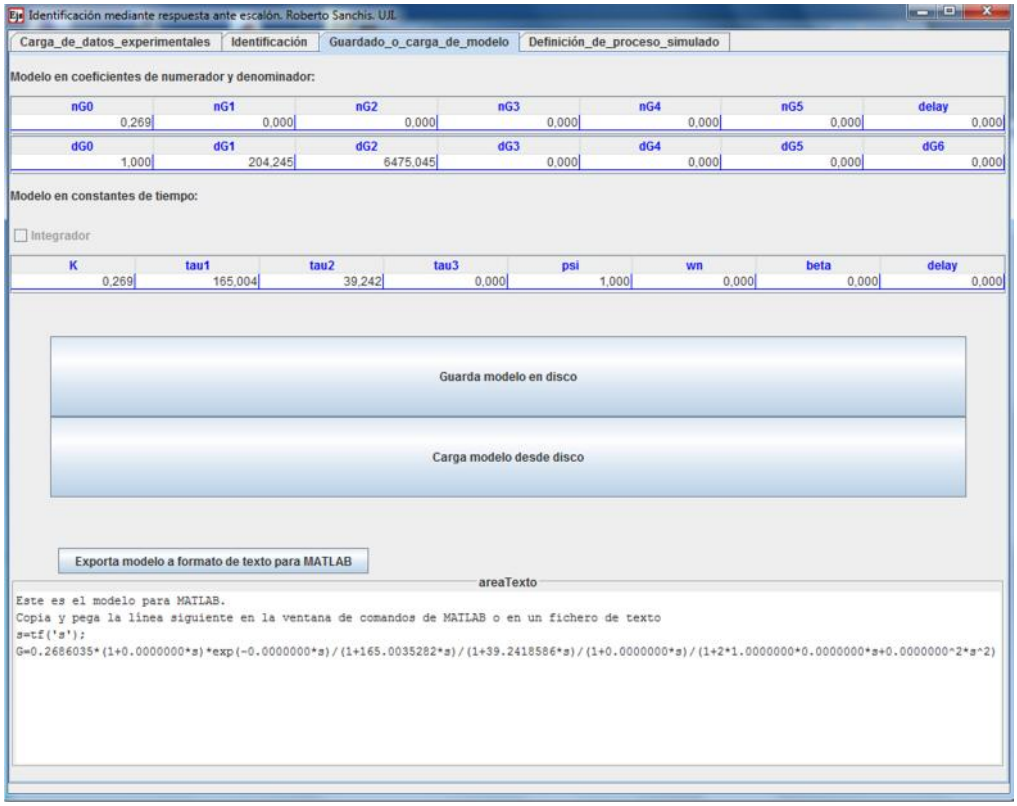


Figura 56.Función transferencia obtenida

El programa a utilizar para hacer el diseño del controlador PID, es la *ejs_model_PIDFREC2047.jar* (Aplicación de diseño PID).

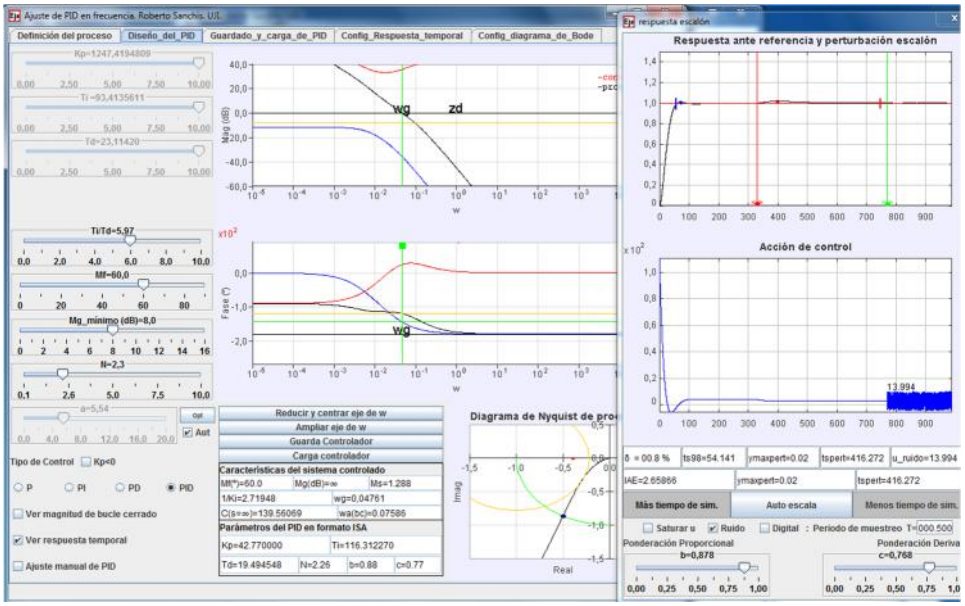


Figura 57. Diseño controlador PID

En la pestaña *Diseño_del_PID* se regula la *N* hasta obtener un ruido pequeño, y se regula el parámetro *b* y *c* hasta obtener un parametro de sobreoscilación menor al 5% ($\delta \leq 5\%$). Hay que tener en cuenta que cuando se modifique el valor de *N* se debe optimizar el valor de *a*, aunque esto se puede hacer de modo automático si se tiene el *check box* de *Aut* activado, al lado del *slider* del parámetro *N*.

Y para finalizar en la pestaña *Guardado_y_carga_de_PID* se observan los valores a introducir en los *edits box* de la aplicación de Matlab *Sistema de Control de Bucle Cerrado*.

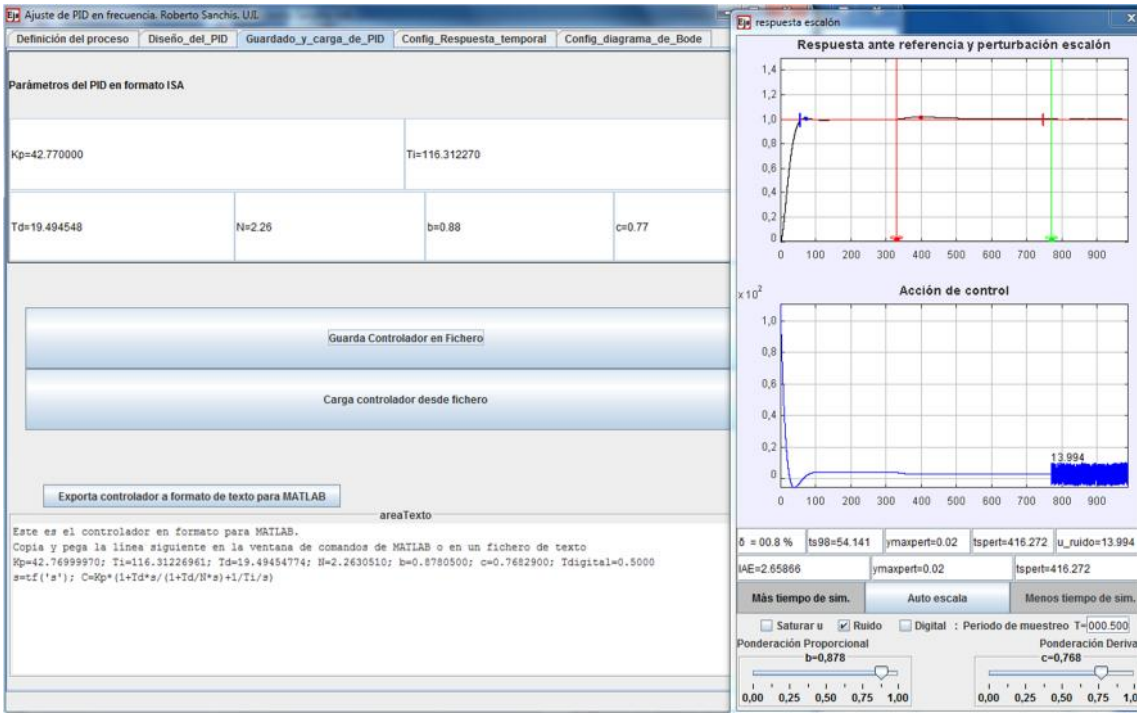


Figura 58.Controlador PID obtenido

Una vez hecho el diseño, se introducen los datos en la subventana de *Sistema de Control Bucle Cerrado*, en los *edits boxs* se introducen las constantes calculadas, y una referencia.

Para el presente experimento, se puso una referencia inicial de 8 cm, y cuando se observó que el nivel del agua, era equivalente a la referencia, se subió el valor de está, a 16 cm, hasta que se observó que el nivel del agua era equivalente a la nueva referencia, dando por bueno el experimento. Siendo el resultado el de la figura 59.

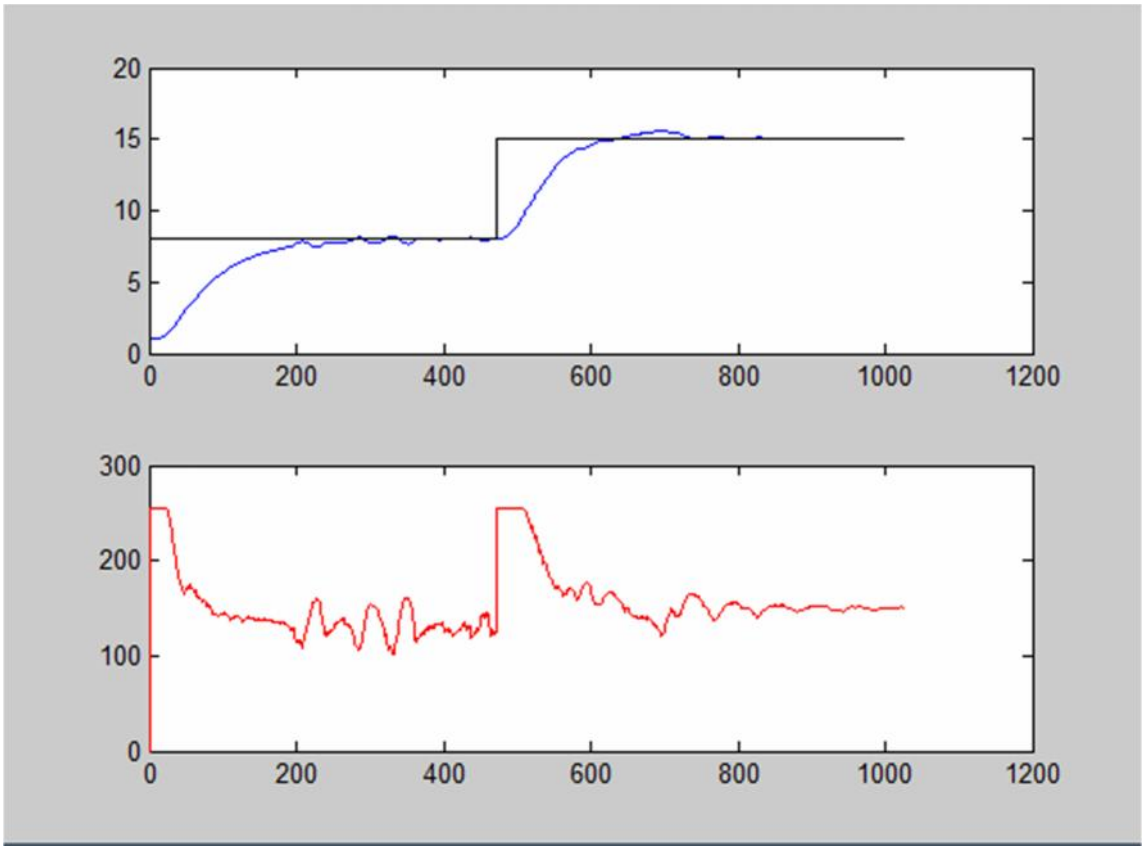


Figura 59. Resultado de aplicar el PID

1.11. Control PID de sistema de dos entradas y dos salidas

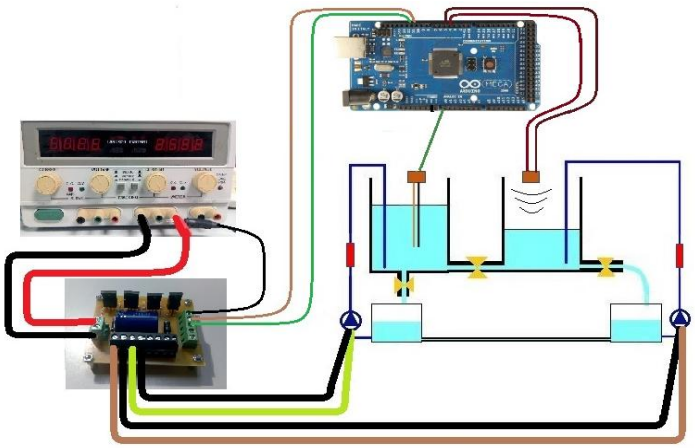


Figura 60. Sistema de dos entradas y dos salidas

Un sistema multivariable es aquel en el cual hay varias entradas de control, y varias salidas que interesa controlar. Normalmente debe haber el mismo número de entradas de control que de variables de salida. El modelo lineal de un sistema multivariable continuo está definido por las funciones de transferencia entre cada entrada y cada una de las salidas. Considérese, por ejemplo, un sistema de 2 entradas y 2 salidas. El modelo del sistema se puede representar como:

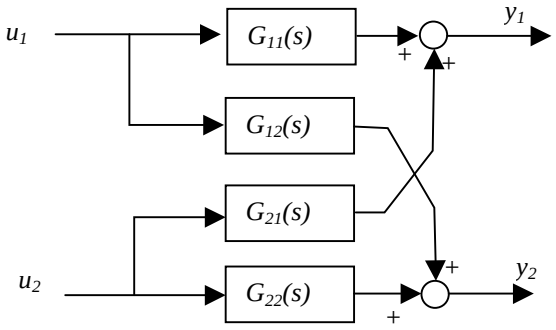


Figura 61. Modelo del sistema multivariable

Que se puede expresar en forma de matriz de transferencia como:

$$\begin{matrix} Y_1\ s \\ Y_2\ s \end{matrix} = \begin{matrix} G_{11}\ s & G_{21}\ s \\ G_{12}\ s & G_{22}\ s \end{matrix} \cdot \begin{matrix} U_1\ s \\ U_2\ s \end{matrix}$$

Un primer paso que hay que realizar en este tipo de sistemas es el **emparejamiento de variables**, o definición de bucles principales. Esto puede no ser trivial, pero en general habría que emparejar cada variable de entrada con la variable de salida sobre la que tiene más influencia. Supóngase que en este caso u_1 tiene su mayor efecto sobre y_1, aunque también afecta a y_2, mientras que u_2 tiene su mayor efecto sobre y_2. Una vez emparejadas, se podría decir que se tienen dos bucles o lazos de control que tienen interacción entre ellos. La interacción se debe a los bloques G_{12} y G_{21}, que hacen que si cambia la entrada de un proceso se vea afectado el otro.

El control de estos sistemas se puede plantear de diversas formas. La posibilidad más sencilla es definir dos bucles de control independientes, uno para cada par entrada-salida. Para el par u_1,y_1 se tendría por ejemplo:

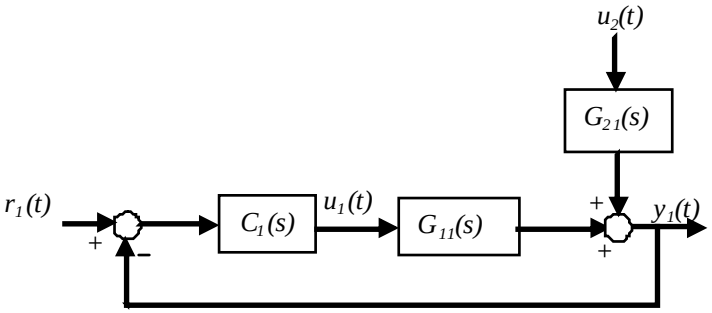


Figura 62. Control el sistema multivariable

Con esta estructura, el efecto de la segunda entrada se considera una perturbación, que será compensada por el controlador $C_1(s)$ para conseguir que el error sea cero.

El carácter especial de esta perturbación (es una entrada de control conocida) hace que se pueda plantear fácilmente un control por prealimentación. En este caso no se necesita medir la perturbación, puesto que es una señal de entrada que se tiene (la fija el controlador del otro bucle). La prealimentación podría tener la siguiente estructura:

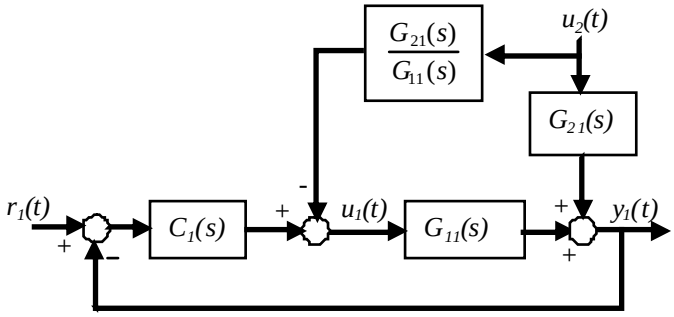


Figura 63. Control el sistema multivariable con prealimentación 1

Siendo exactamente igual para el otro par entrada-salida:

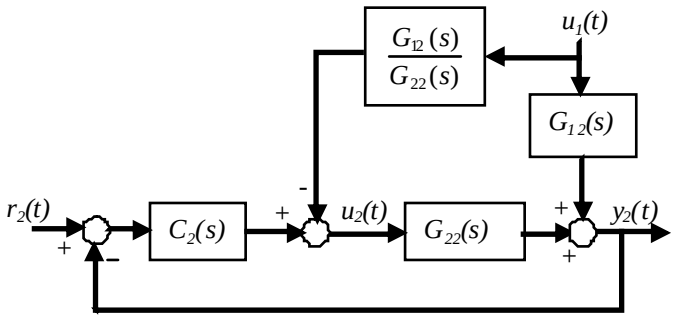


Figura 64 Control el sistema multivariable con prealimentación 2

En los diagramas anteriores de las figuras 63 y 64 u_1 depende de u_2 y éste a su vez de u_1, por lo que la implementación se puede obtener despejando las variables u_1 y u_2 en

función de r_1, r_2, y_1, e y_2. En las siguientes ecuaciones todas las variables son funciones de s (son la transformada de Laplace de las señales, o las funciones de transferencia):

$$u_1 = C_1\ r_1 - y_1\ - u_2 \frac{G_{21}}{G_{11}}$$

$$u_2 = C_2\ r_2 - y_2\ - u_1 \frac{G_{12}}{G_{22}}$$

Despejando se obtiene:

$$u_1 = \frac{G_{11}G_{22}}{G_{11}G_{22} - G_{12}G_{21}} C_1\ r_1 - y_1\ - \frac{G_{21}G_{22}}{G_{11}G_{22} - G_{12}G_{21}} C_2\ r_2 - y_2$$
$$= C_{11}\ r_1 - y_1\ + C_{21}\ r_2 - y_2$$

$$u_2 = \frac{G_{11}G_{22}}{G_{11}G_{22} - G_{12}G_{21}} C_2\ r_2 - y_2\ - \frac{G_{12}G_{11}}{G_{11}G_{22} - G_{12}G_{21}} C_1\ r_1 - y_1$$
$$= C_{12}\ r_1 - y_1\ + C_{22}\ r_2 - y_2$$

Es decir, se tiene en realidad una matriz de transferencia entre el vector de errores y el vector de acciones de control

$$\begin{matrix} u_1\ s \\ u_2\ s \end{matrix} = \begin{matrix} C_{11}\ s & C_{21}\ s \\ C_{12}\ s & C_{22}\ s \end{matrix} \cdot \begin{matrix} R_1\ s\ - Y_1(s) \\ R_2\ s\ - Y_2(s) \end{matrix}$$

El procedimiento práctico sería el siguiente: Se calcula de forma independiente C_1 para el bucle 1 y C_2 para el bucle 2 (por ejemplo un PI o un PID). Después se calculan las funciones de transferencia C_{11}, C_{12}, C_{21}, C_{22} a partir de las expresiones anteriores. Estas funciones de transferencia son las que finalmente se implementan.

Otra opción aproximada más sencilla para implementar esta estrategia es calcular en cada periodo de control, la acción de control del primer bucle, como un PID normal, y restarle el término debido a la otra acción de control calculada en el periodo anterior. Si se aproximan las funciones de transferencia $\frac{G_{21}}{G_{11}}$ y $\frac{G_{12}}{G_{22}}$ por sus ganancias estáticas, k_{21} y k_{12}, el algoritmo quedaría:

y10=entrada analógica 1;
y20= entrada analógica 2;
r10=variable interna;

r20=variable interna;

D1=pd1*D1+qd1(c1*r10-c1*r11-y10+y11);

I1=I1+qi1*(r10-y10);

u1=Kp1*(b1*r10-y10)+D1+I1-k21*u2;

if ((u1<u1min)&(e1<0))|((u1>u1max)&(e1>0)) then I1=I1-qi1*(r10-y10);

if (u1<u1min) u1=u1min;

if (u1>u1max) u1=u1max;

D2=pd2*D2+qd2(c2*r20-c2*r21-y20+y21);

I2=I2+qi2*(r20-y20);

u2=Kp2*(b2*r20-y20)+D2+I2-k12*u1;

if ((u2<u2min)&(e2<0))|((u2>u2max)&(e2>0)) then I2=I2-qi2*(r20-y20);

if (u2<u2min) u2=u2min;

if (u2>u2max) u2=u2max;

salida analógica 1=u1;

salida analógica 2=u2;

r11=r10;

y11=y10;

r21=r20;

y21=y20;

La estrategia de control anterior da como resultado (si los modelos fueran exactos) que los dos bucles de control funcionen de forma independiente (como dos bucles sencillos de una entrada y una salida). Por esta razón se llama **control por desacoplamiento** (se ha cancelado el acoplamiento entre los dos bucles, es decir, el efecto de cada entrada sobre la salida del otro bucle).

La estrategia anterior puede tener problemas importantes.

- La función de transferencia resultante puede ser no realizable (el orden del numerador mayor que el del denominador). Esto sucede si $G_{12}(s)/G_{22}(s)$ ó $G_{21}(s)/G_{11}(s)$ son no realizables. En ese caso se puede optar por compensar solo la ganancia estática en la prealimentación (utilizar $G_{ij}(0)$ en lugar de $G_{ij}(s)$). Otra posibilidad es utilizar funciones de transferencia aproximadas (por ejemplo con los polos y ceros

dominantes) de forma que los cocientes anteriores sean realizables.

- Si la función de transferencia $\frac{G_{12}G_{21}}{G_{11}G_{22}-G_{12}G_{21}}$ tiene algún polo inestable, el controlador anterior trata de cancelarlo, lo que resulta en un sistema inestable en bucle cerrado. Para evitar esto se podría compensar solo la ganancia estática (utilizar $G_{ij}(0)$ en lugar de $G_{ij}(s)$ en la prealimentación), o utilizar funciones de transferencia aproximadas para que la función de transferencia anterior no tuviera polos inestables.

Una vez explicado esto, se realizará el mismo proceso que en el apartado anterior, pero adaptado para el modelo multivariable. Para ello se obtendrán 4 funciones de transferencia, estas serán: la que relaciona el sensor de nivel capacitivo con su bomba, y otra función de transferencia, la que relaciona este sensor con la bomba del otro sistema de depósitos, y la misma operación con el sensor ultrasonido.

Para obtener estas funciones de transferencia lo primero que hay que realizar es una serie de experimentos en bucle abierto, con la ayuda de la aplicación *Multivariable bucle abierto*. Siendo los datos obtenidos los representados en la figura x, en la gráfica superior están representados los datos del sensor de ultrasonidos, en la que está justo debajo de esta primera está representado el valor PWM de la tensión que le llega a la bomba, en la tercera representación se observan los datos del sensor de nivel capacitivo y en la última gráfica el valor PWM de la tensión que le llega a la bomba de este sistema de depósitos. Los escalones se explican en la siguiente tabla.

Timer(s)	0	200	380	950	1200
Bomba ultrasonidos	100	100	110	110	120
Bomba sensor capacitivo	100	110	110	120	120

Tabla 3. PWM aplicados

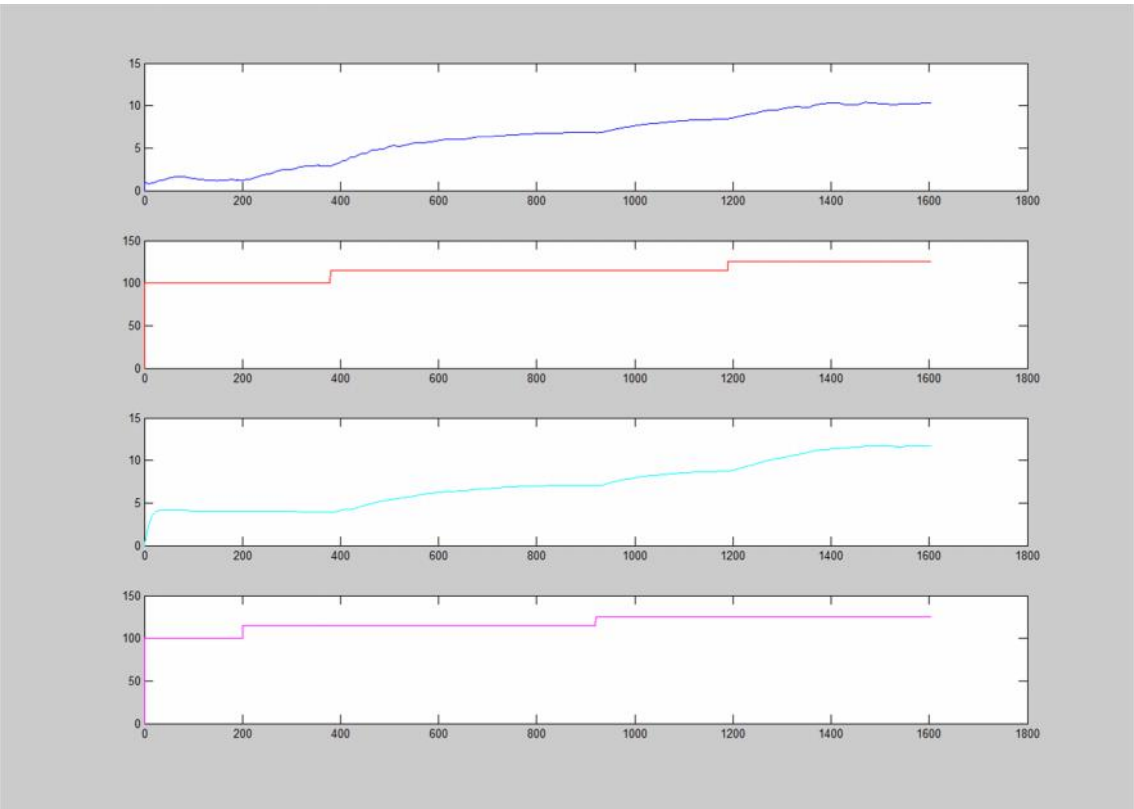


Figura 65. Resultados obtenidos de poner los PWM de la Tabla 2

A continuación se obtienen las 4 funciones de transferencia, del mismo modo que en el apartado 1.10, dos de las cuales servirán para calcular el PID, y las otras dos para calcular el factor de corrección explicado en la teoría, del sistema multivariable. El escalón escogido es el de 110 a 120.

La primera función de transferencia obtenida, la que relaciona, el sensor de nivel capacitivo con la bomba de su sistema, es la siguiente.

$$G_{11} = \frac{0.185}{(1 + 102.944 \cdot s) \cdot (1 + 10.548 \cdot s)}$$

En las siguientes figuras, se muestra el proceso para el cálculo de dicha función de transferencia.

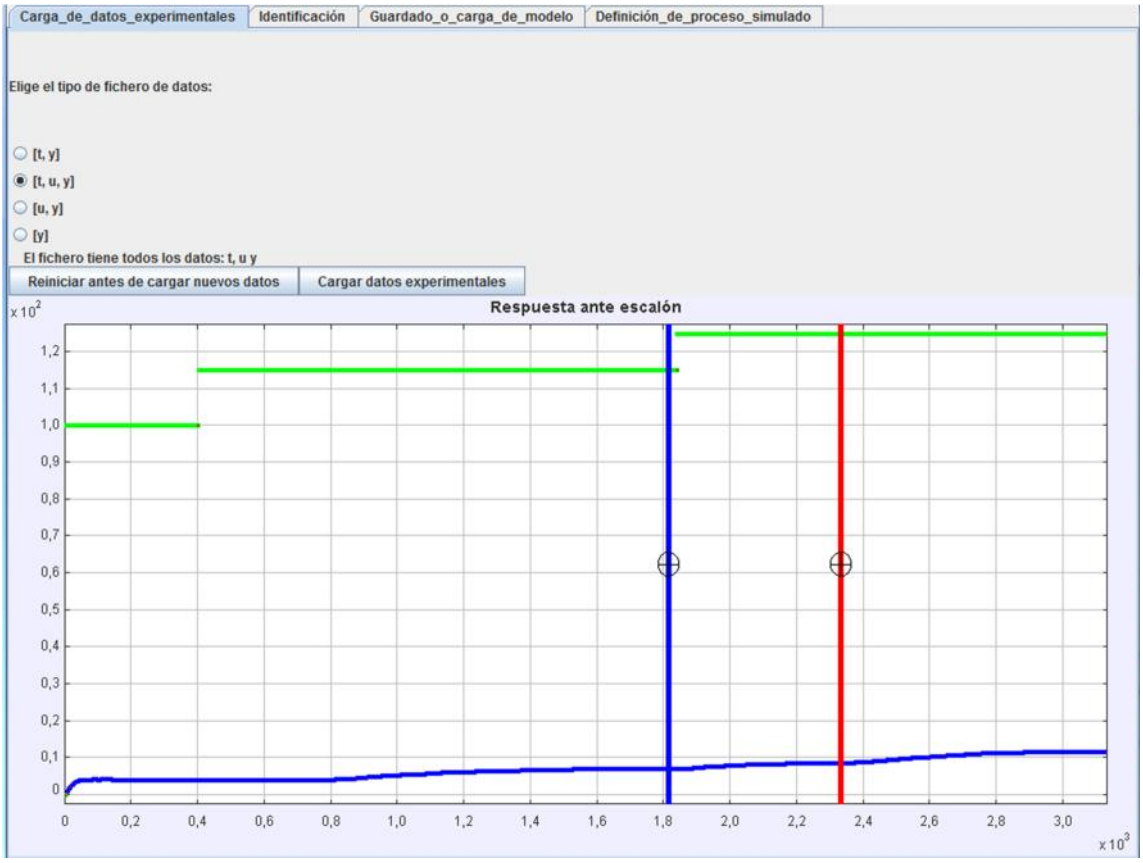


Figura 66. Escalón escogido 2

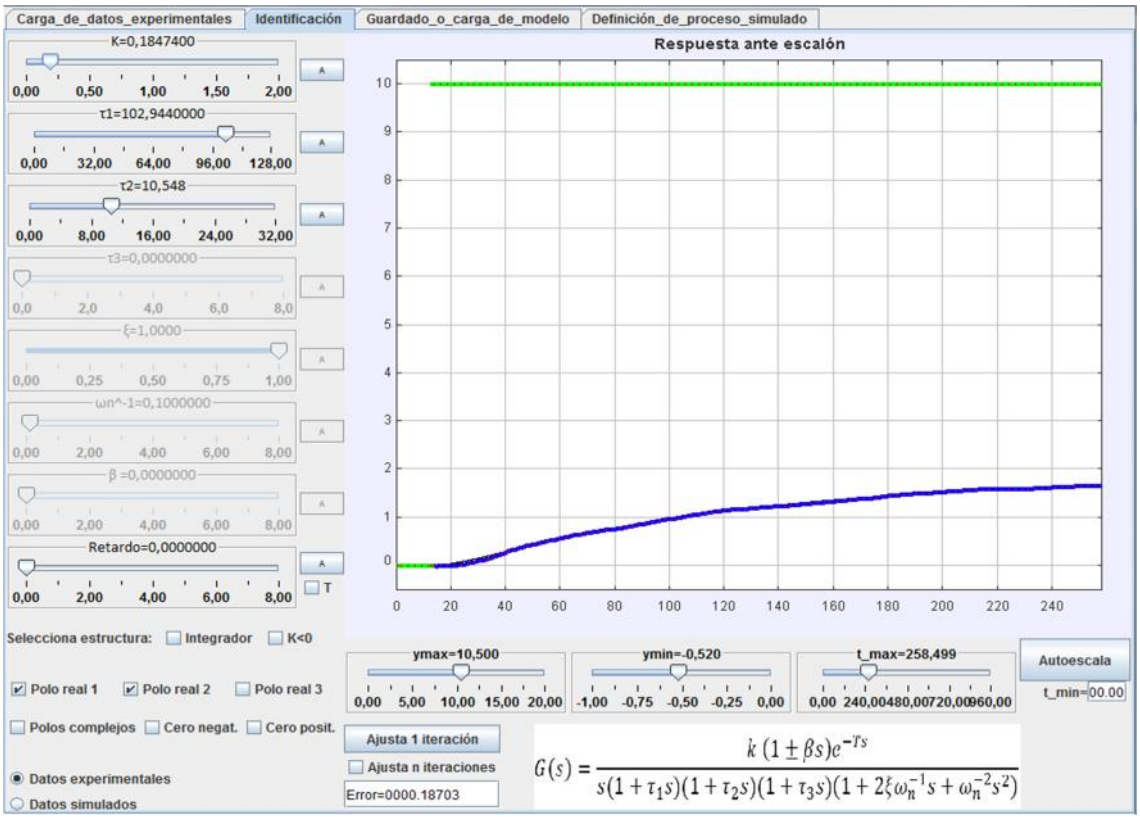


Figura 67. Función de transferencia 2

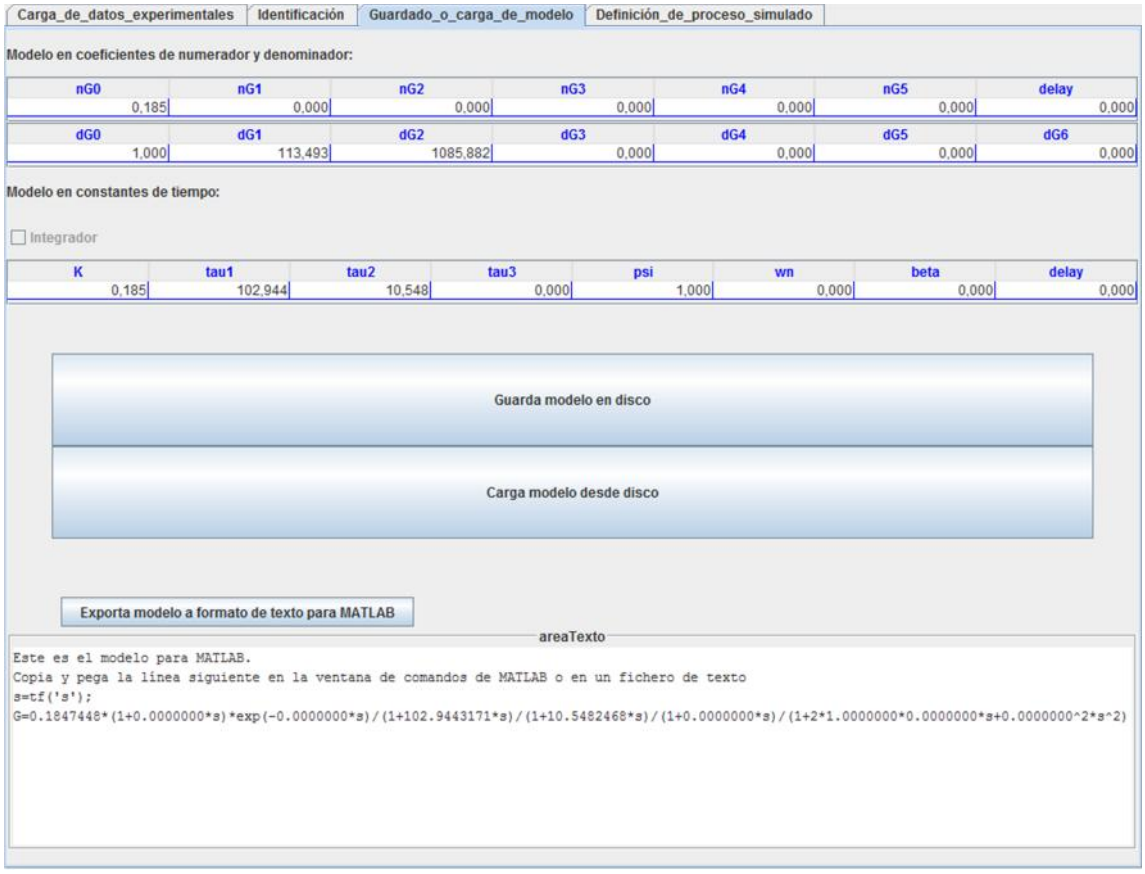


Figura 68. Función de transferencia obtenida 2

Una vez obtenida la función de transferencia se cambia de herramienta, y se calcula el controlador PID. Los datos de este PID, serán introducidos en la zona de *PID1* de la subventana de *Multicontrol de bucle cerrado.*

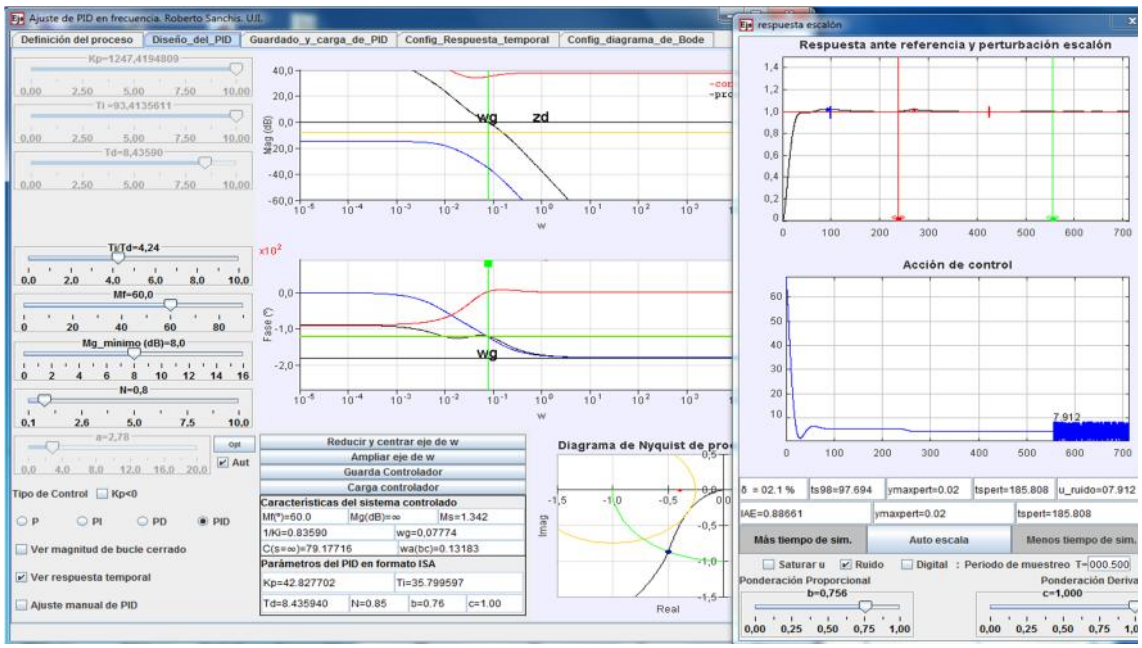


Figura 69. Diseño controlador PID 2

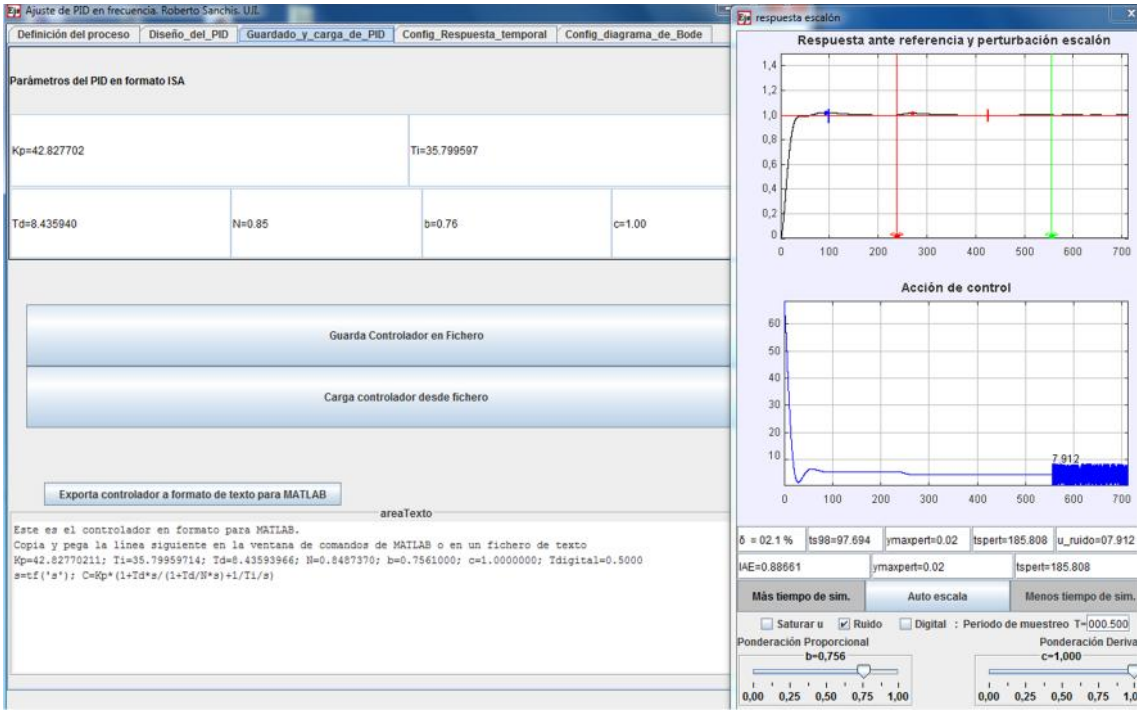


Figura 70. Controlador PID obtenido 2

Una vez obtenido este controlador, se procede a calcular la función de transferencia, la cual relaciona el sensor de nivel capacitivo con la bomba del sistema contrario. El escalón escogido es el de 110 a 120.

La función de transferencia de éste es:

$$G_{12} = \frac{0.195}{(1 + 99.588 \cdot s) \cdot (1 + 8.548 \cdot s)}$$

Una vez calculada se pondrá que *s*=0, tanto en la G_{12} cómo para la G_{11}, se sacará el valor de la k_{12} tal y como se explica en la teoria del sistema multivariable. El valor obtenido es: 0.863.

En las siguientes figuras, se muestra el proceso para el cálculo de dicha función de transferencia.

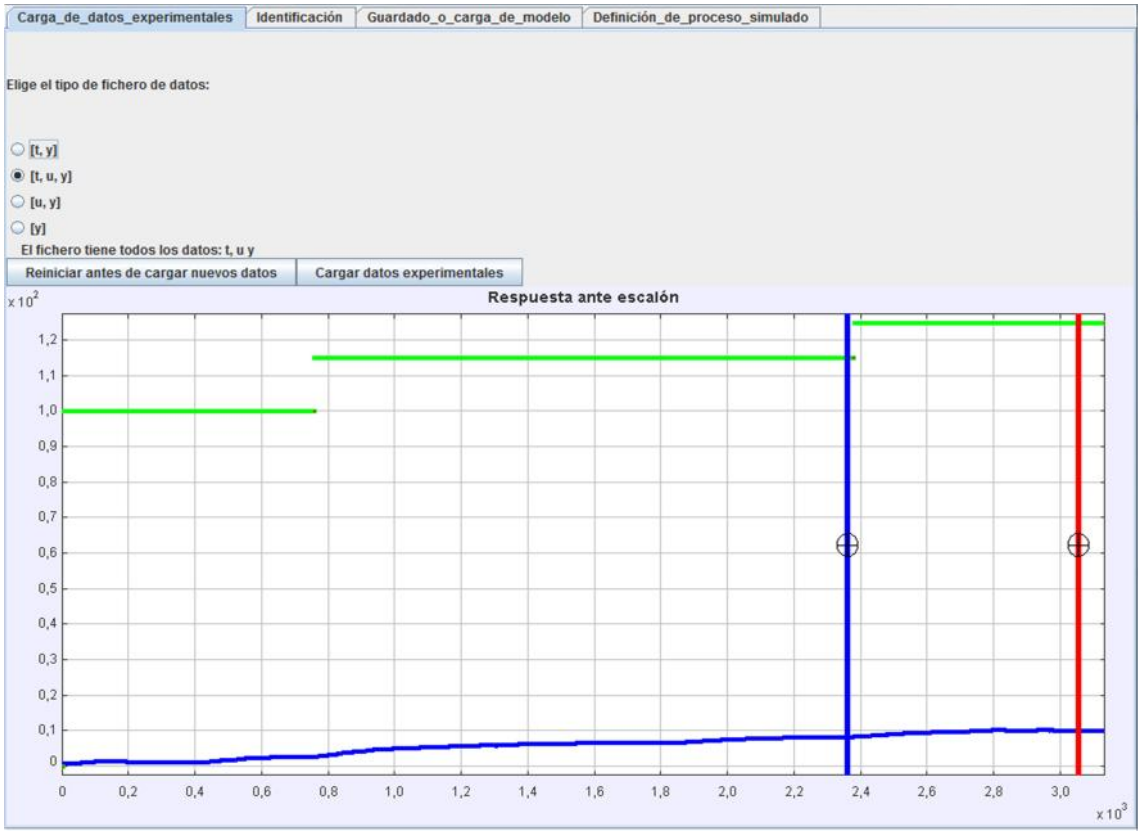


Figura 71. Escalón escogido 3

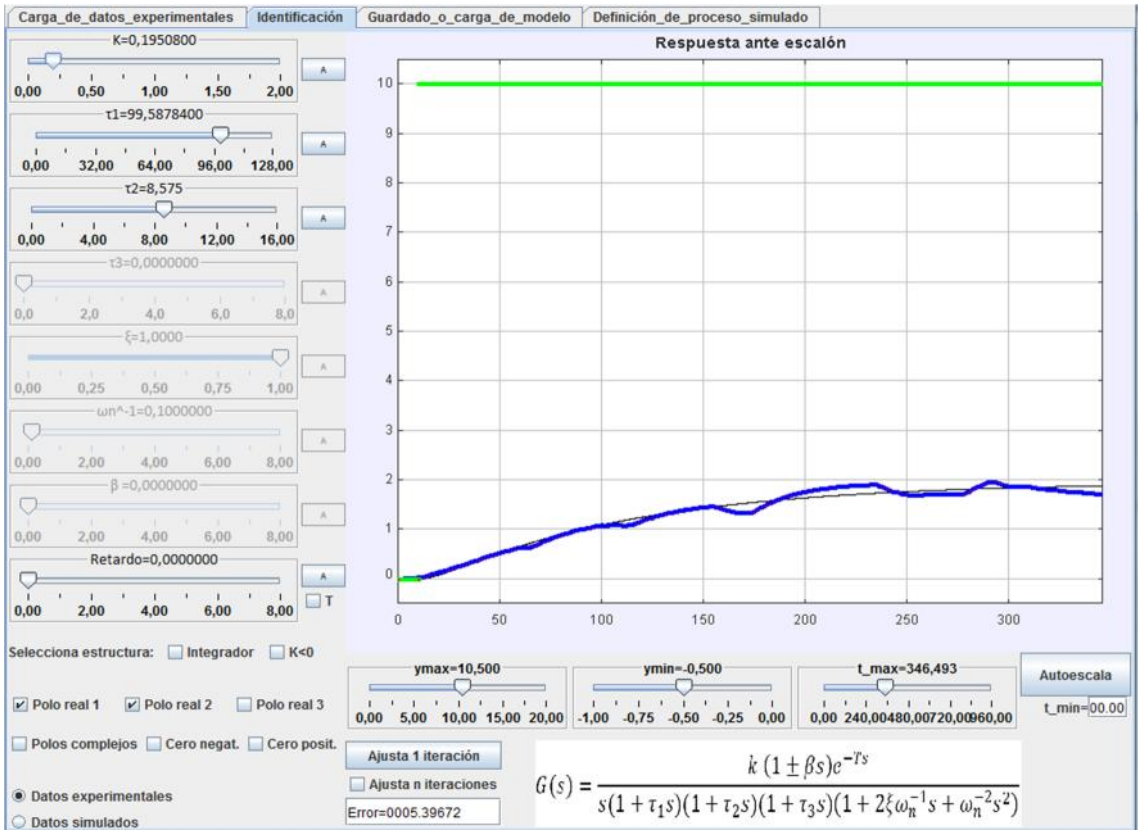


Figura 72. Función de transferencia 3

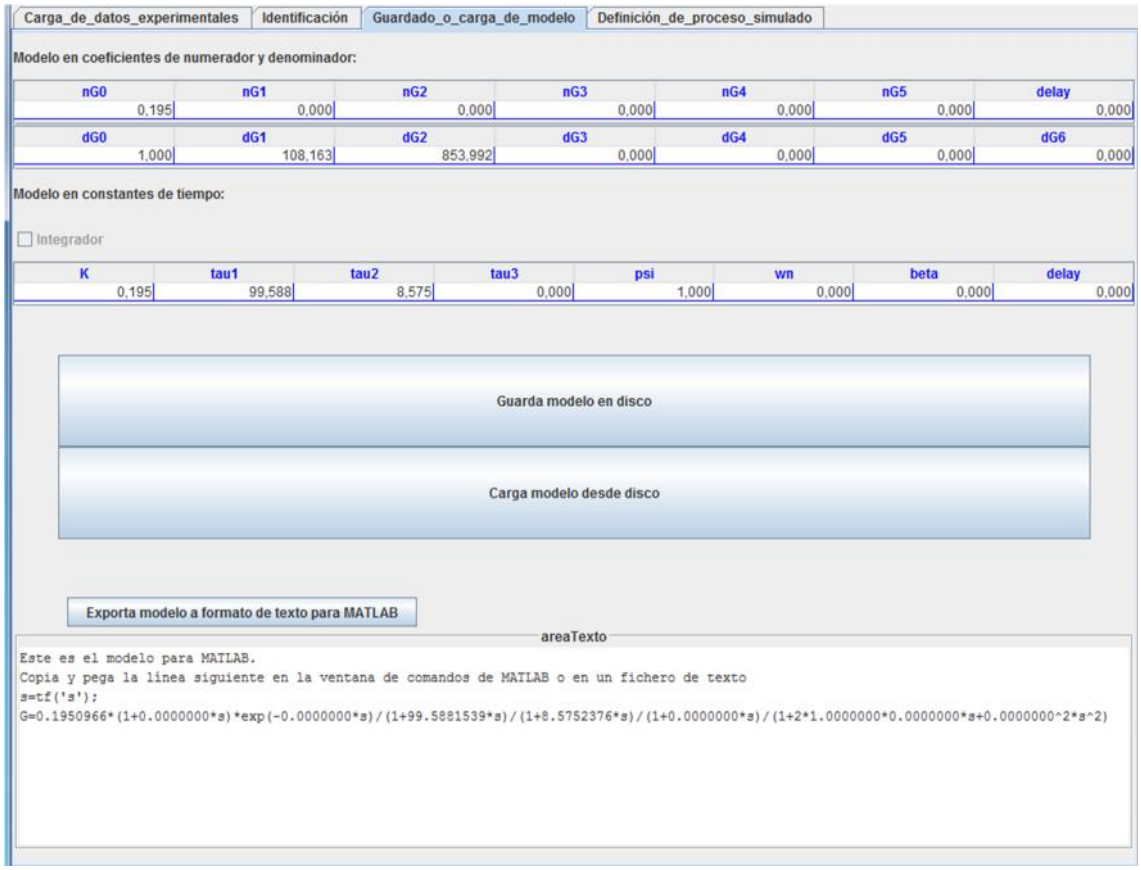


Figura 73. Función de transferencia obtenida 3

Una vez obtenidas las funciones de transferencia para el sensor de nivel capacitivo, y el controlador PID para éste, se procede a calcular el siguiente controlador, el cual, usará la función de transferencia que relaciona, el sensor ultrasonidos con la bomba de su sistema. El escalón escogido es el de 110 a 120.

La función de transferencia de éste es:

$$G_{22} = \frac{0.226}{(1 + 176.937 \cdot s) \cdot (1 + 26.054 \cdot s)}$$

En las figuras, se muestra el proceso para el cálculo de dicha función de transferencia.

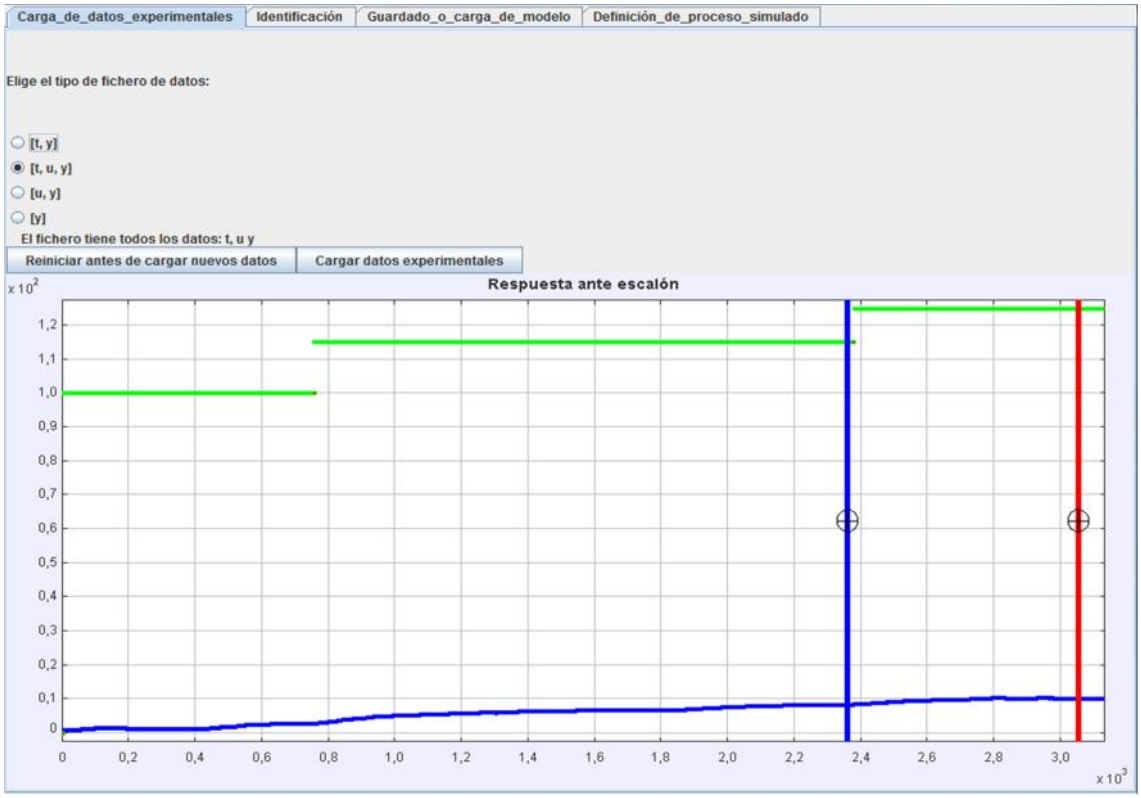


Figura 74. Escalón escogido 4

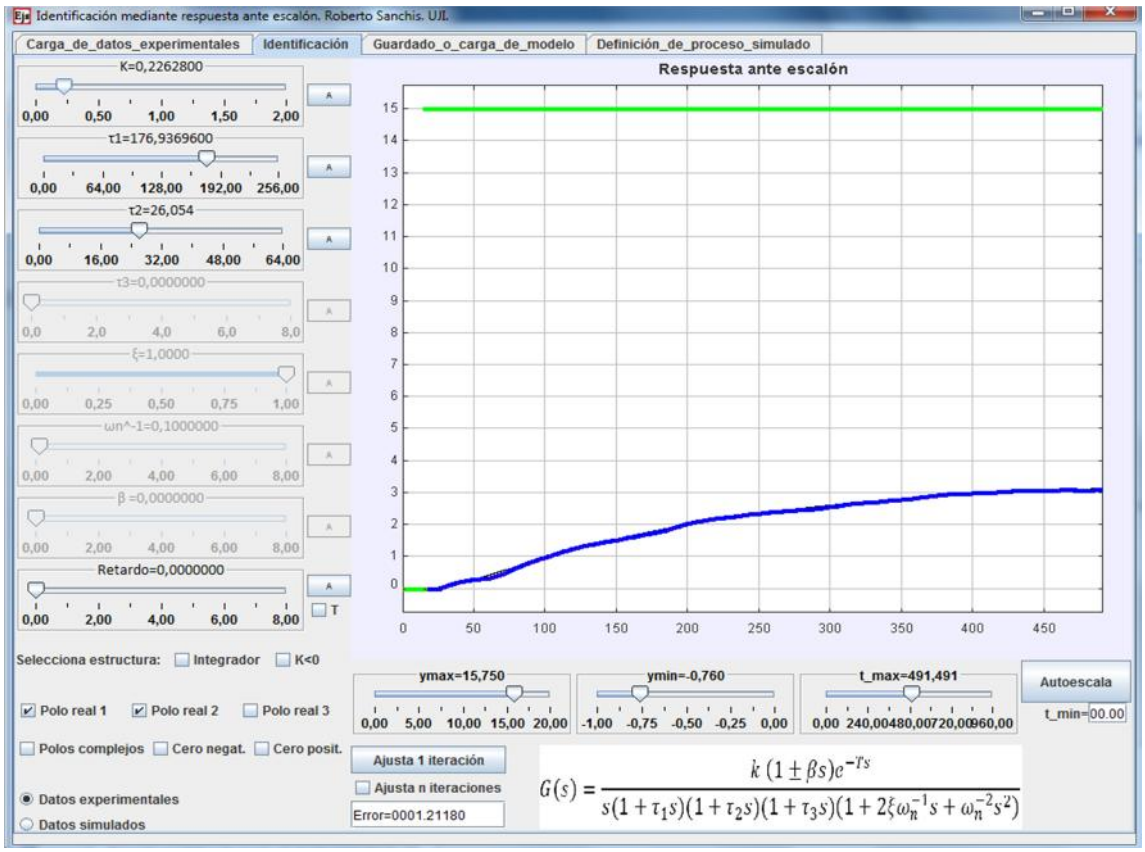


Figura 75. Función de transferencia 4

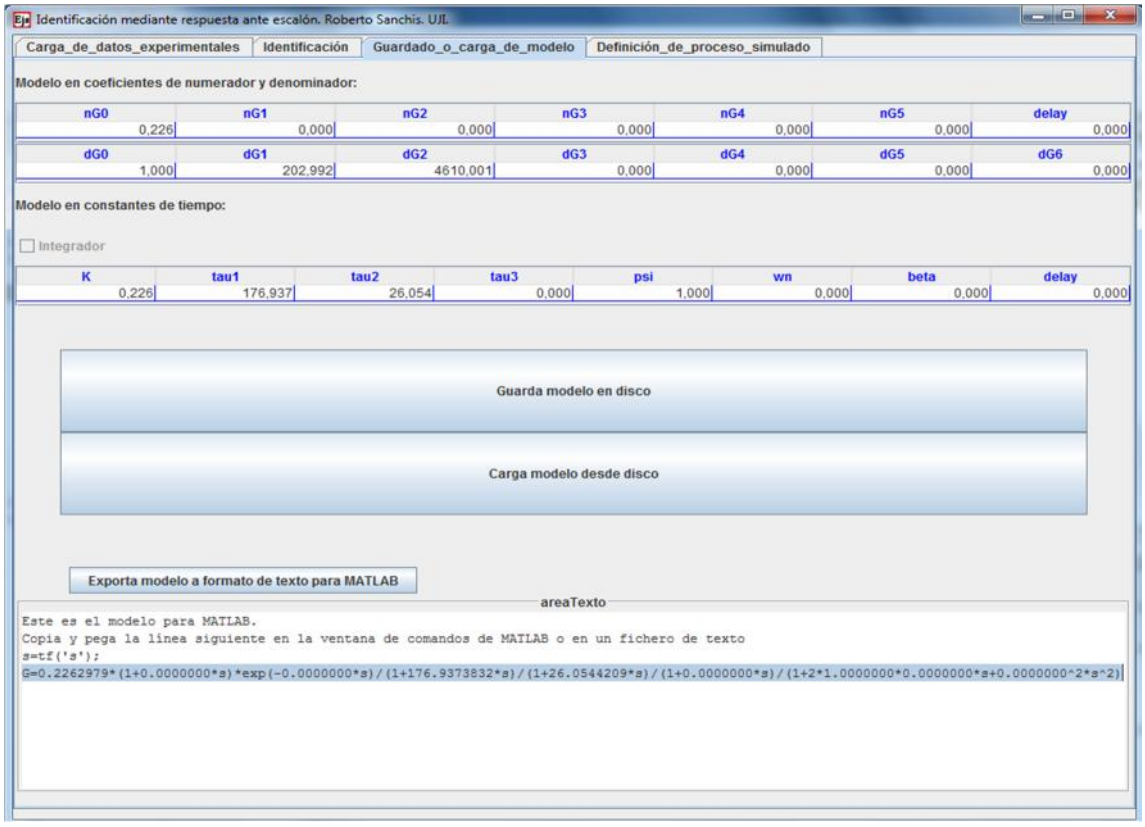


Figura 76. Función de transferencia obtenida 4

Una vez obtenida la función de transferencia se cambia de herramienta, y se calcula el controlador PID. Los datos de este PID, serán introducidos en la zona de *PID2* de la subventana de *Multicontrol de bucle cerrado.*

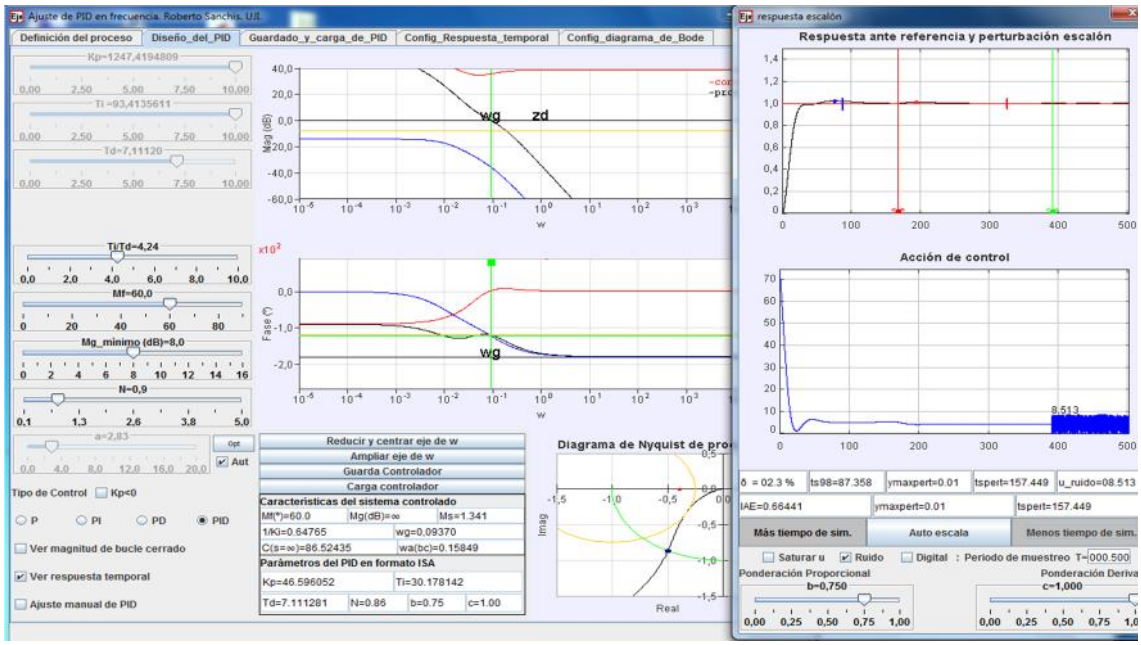


Figura 77. Diseño PID 3

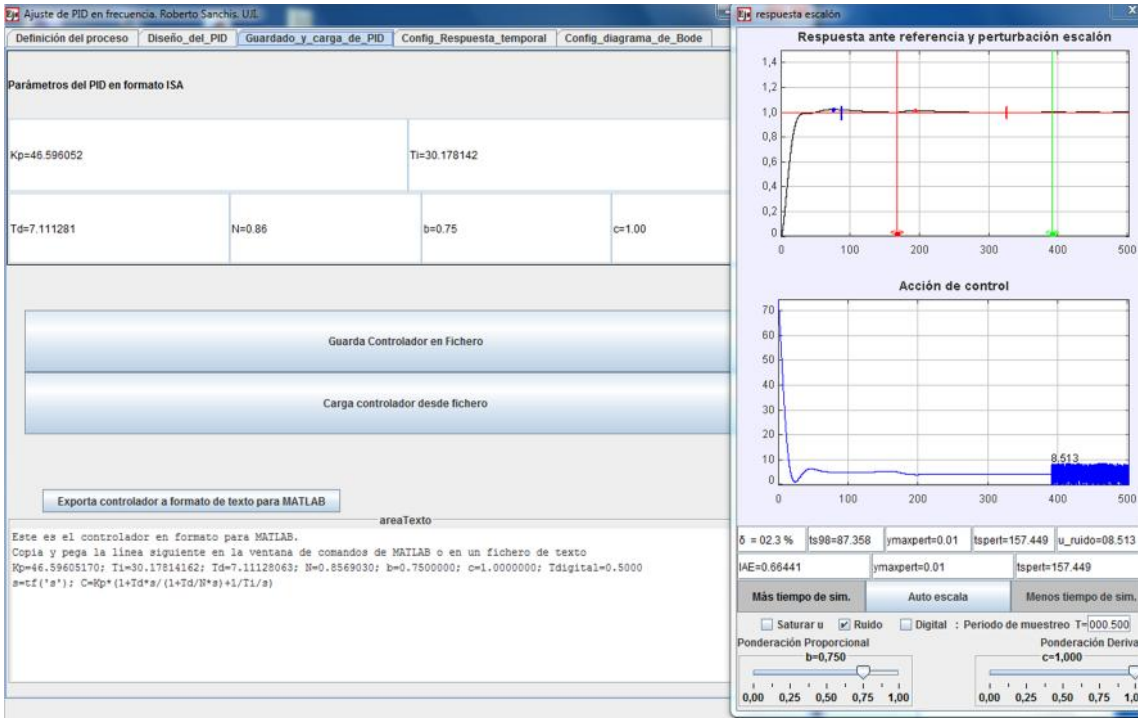


Figura 78. Controlador PID obtenido 3

Una vez obtenido este controlador, se procede a calcular la última función de transferencia, la cual relaciona el sensor de ultrasonidos con la bomba del sistema contrario. El escalón escogido es el de 110 a 120.

La función de transferencia de este es:

$$G_{21} = \frac{0.185}{(1 + 116.544 \cdot s) \cdot (1 + 10.544 \cdot s)}$$

Una vez calculada se pondrá que *s*=0, tanto en la G_{21} cómo para la G_{22}, se sacará el valor de la k_{12} tal y como se explica en la teoria del sistema multivariable. El valor obtenido es: 1.

En las figuras, se muestra el proceso para el cálculo de dicha función de transferencia.

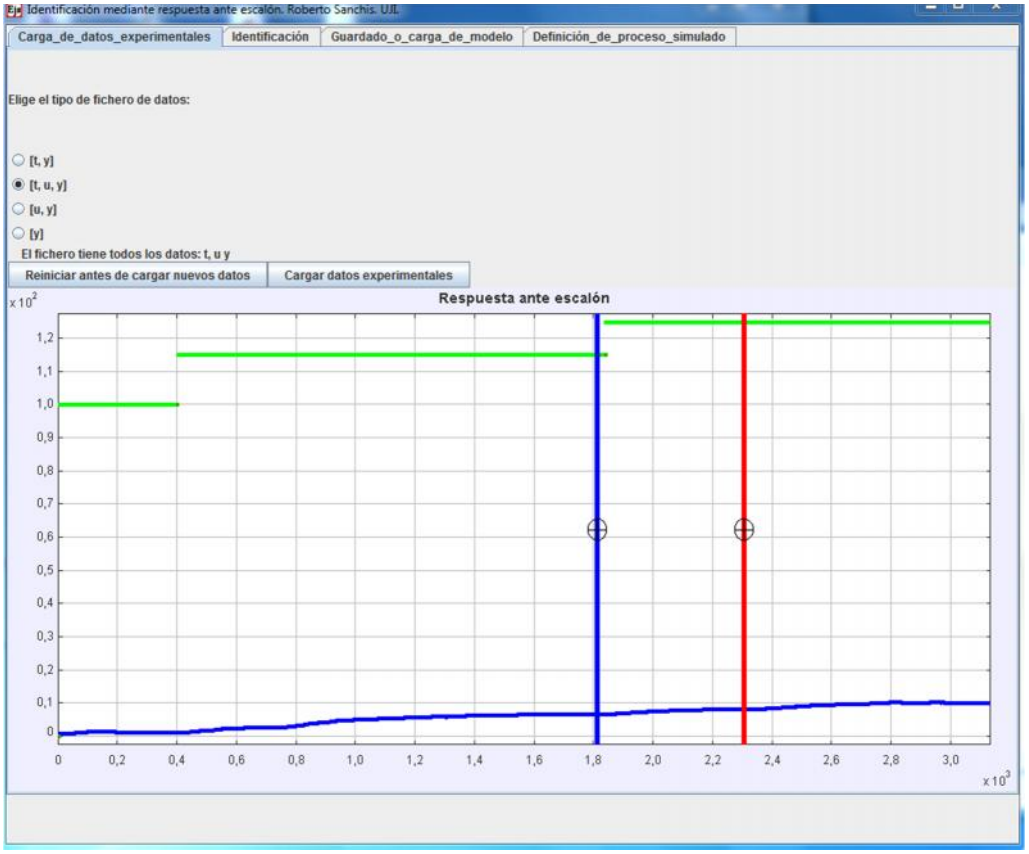


Figura 79. Escalón escogido 5

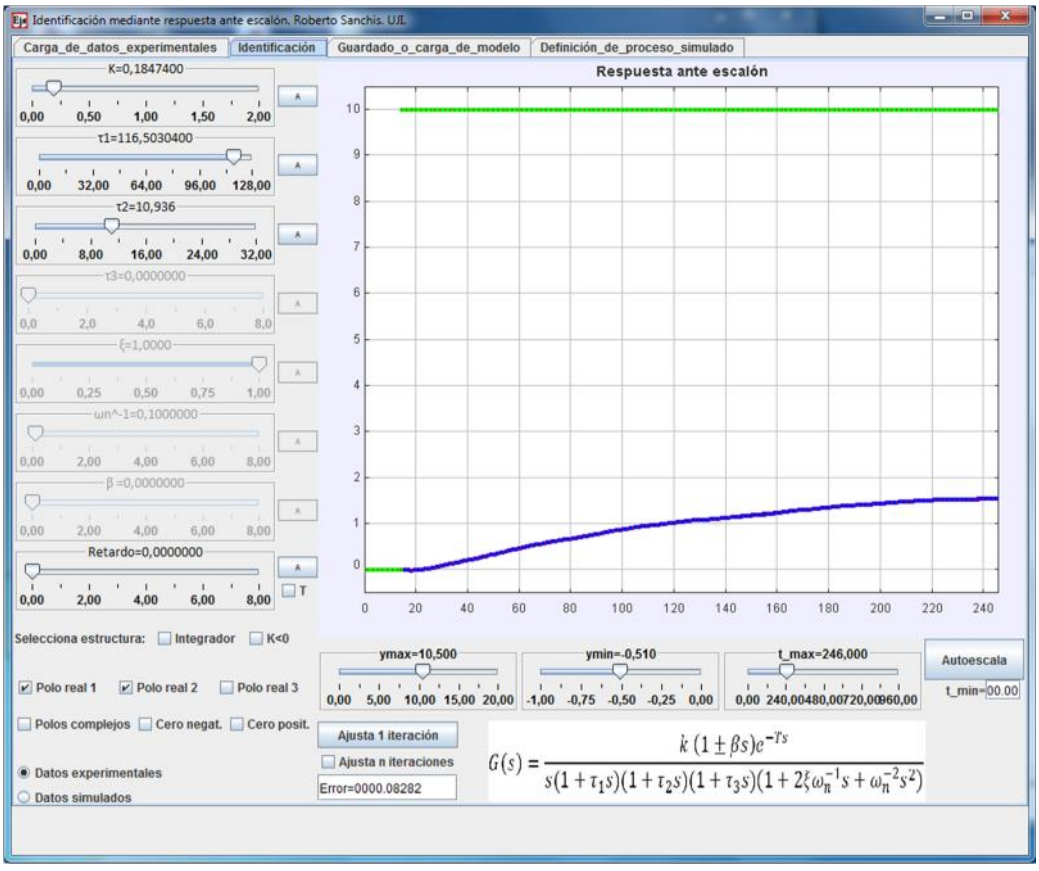


Figura 80.Función de transferencia 5

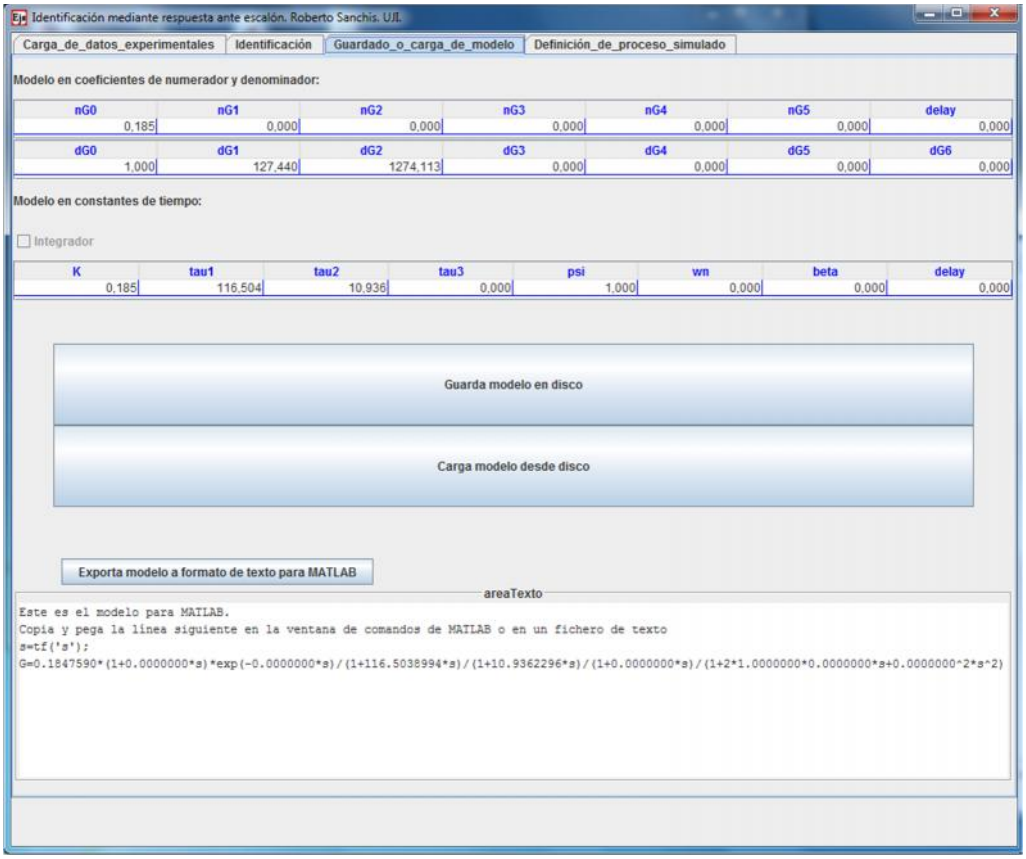


Figura 81.Función de transferencia obtenida 5

Cabe recordar el archivo que se ha usado para el cálculo de cada función de transferencia, lo cual se puede observar en la siguiente tabla.

	Archivo
Sensor nivel con su bomba	*sensor_con_su_bomba.txt*
Sensor nivel bomba contraria	*sensor_con_bomba_contrario.txt*
Ultrasonidos nivel con su bomba	*ultrasonidos_con_su_bomba.txt*
Ultrasonidos nivel bomba contraria	*ultrasonidos_con_bomba_contrario.txt*

Tabla 4. Contenido de los archivos guardados

En el caso 1 y 3, los archivos se guardan con el nombre que desee el usuario, y ese archivo se usarán en la aplicación *ejs_model_identescalon23.jar* (Aplicación de identificación).

Una vez introducidos todos los datos a los campos correspondientes, se selecciona si se desea que el algoritmo de control incluya los términos de desacoplamiento o no, y se empieza el experimento. En la figura 82 se muestran los resultados sin el desacoplamiento, y en la figura 83 se muestra el resultado con los términos de

desacoplamiento, para una mayor comparación se pone en la k_{12} el valor calculado y en la k_{21} se iguala a 0, así en una gráfica se puede detectar los efectos del desacoplamiento.

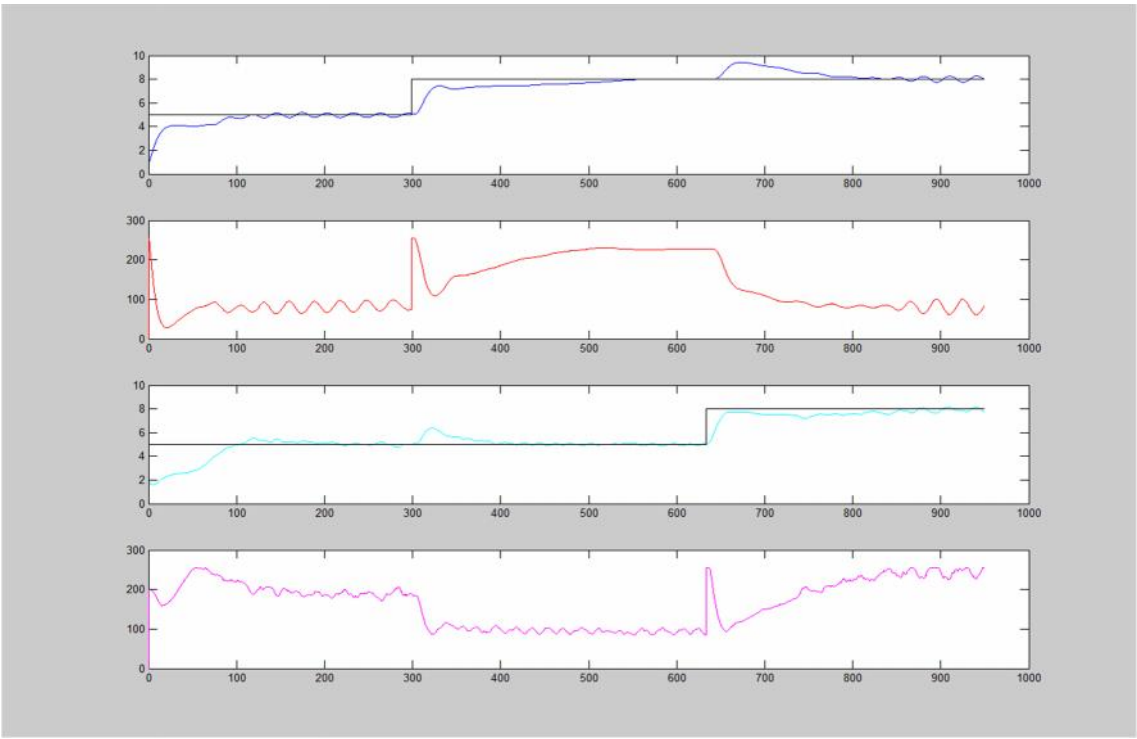


Figura 82. Resultado sin el acoplamiento

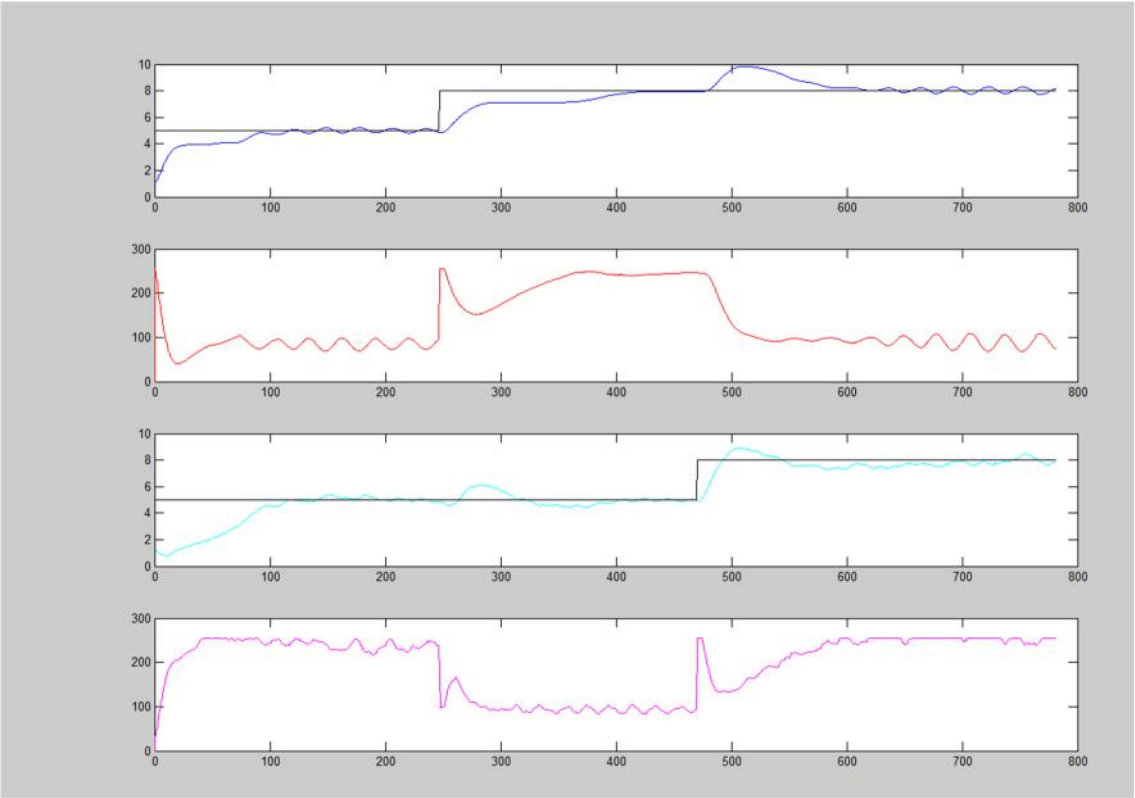


Figura 83. Resultado con acoplamiento, diferenciándose compensación y sin compnesación

Para el presente experimento, se puso una referencia inicial de 5 cm en ambos depósitos, y cuando se observó que el nivel del agua, era equivalente a la referencia en ambos sitios, se subió la referencia en uno de los depósitos, a 8 cm, hasta que se observó que el nivel del agua era equivalente a la nueva referencia. Cuando esto ocurrió se subió la referencia del otro depósito al mismo valor, y se dejó el experimento hasta que este se estabilizara.

En las figuras 82 y 83, se puede observar que, cuando se tienen en cuenta los parámetros de acoplamiento, el tiempo de establecimiento frente a la perturbación provocada por la bomba contraria, y el valor máximo de esta respuesta frente a esa perturbación es mucho menor que cuando no están los parámetros de acoplamiento.

1.12. Viabilidad

1.12.1. Viabilidad técnica.

Todos los dispositivos y/o software requeridos para la ejecución de las soluciones presentadas en el presente proyecto, están disponibles en el mercado. Por tanto, el proyecto se considera viable

1.12.2. Viabilidad económica.

Para el estudio de la viabilidad económica se va a comparar el presupuesto necesario, con el precio de la compra de un sistema de doble depósito acoplado.

Para ello, lo primero que se hace es buscar el precio de un sistema de doble depósito acoplado. Los precios encontrados en el mercado, están entre 30.000 y 40.000€.

Una vez sabido los precios de compra de un sistema de doble depósito acoplado comercial, se comprueba cuánto puede costar realizar el proyecto y comprobar así si es rentable.

Sistemas de depósitos

Cada sistema de depósitos cuesta 2500€, por lo tanto, al tener 2, el precio será de 5000€. En el precio, está incluido el sensor de nivel capacitivo.

Software

El único software que tiene coste es el MATLAB, ya que los otros programas son gratuitos. El precio de éste de 2500€.

Fabricación PCB

Fabricar una PCB cuesta 20€. Si se decidiera realizar más, el precio bajaría, siendo el coste de 10€ para la fabricación de 10.000 PCB.

Componentes PCB

Componente	Referencia	Precio (€)	Unidades
MOSFET	IRLIZ44GPbF	2.05	4
Diodo		0.16	4
Resistencia 1K		0.03	4
Resistencia 100K		0.04	4
Conector 8 pines		1.23	1
Conector 5 pines		1.31	1
Conector 2 pines		0.93	1
Condensador 220nF		1.12	1

Tabla 5. Componentes electrónicos

Otros

Objeto	Precio (€)	Unidades
Arduino MEGA	19	1
Sensor Ultrasonidos	3	1
Cables	4	1

Tabla 6. Material necesario

En el precio de los cables se incluyen todos los cables utilizados.

Horas personal

Profesional	Precio (€)	Horas
Operario	25	2
Ingeniero Hardware	40	50
Ingeniero Software	40	125

Tabla 7.Horas del personal

Precio final

El precio final es la suma de todos los apartados anteriores, el cual es de 14.609,71€.

Con estos datos, se puede observar, que solo con la inversión inicial, con la que se tienen en cuenta las horas de investigación y desarrollo del software y hardware, el precio ya es menor que comprar un sistema de doble depósito. Con lo que se considera que el proyecto será viable.

1.13. Bibliografía

Roberto Sanchis. Problemas resueltos de teoría de sistemas. Publicacions de la Universitat Jaume I.

K. Ogata. Ingeniería de control moderna. Ed. Prentice-Hall, 1993.

J. Bondia i A. Sala. Comportamiento Dinámico de Sistemas. UPV. 2000.

A. Sala, P. Albertos. Problemas resueltos. Comportamiento dinámico de sistemas. SPUPV. 1995. (Código SPUPV-95.061).

Blasco, F.X. et al. Sistemas automáticos. Editorial UPV. 2000.4186.

Antonio Barrientos, Ricardo Sanz, Fernando Matía i Ernesto Gambao. Control de sistemas continuos. Problemas resueltos. McGraw-Hill. 1996.

https://sites.google.com/a/uji.es/freepidtools/home

https://www.arduino.cc

http://es.mathworks.com/

Anexos

1.14.1 Código Arduino

```
/* Autor: Aitor Roig Alemany                                          */
/* Fecha: Agosto 2015                                                 */
/* "Sistema de control automático de una planta de laboratorio        */
/* basada en 2 depósitos"                                             */


/* Programa de adquisicion de datos de los diversos sensores, y los   */
/* envia por el puerto serie, para que el programa correspondiente    */
/* haga lo que deba hacer.                                            */
/*                                                                    */
/* Ademas envia la tensión que debe llegarle a la bomba               */

#include <NewPing.h>

/* Definimos los pines a los que conectaremos los sensores            */
#define TRIGGER_PIN 4

#define ECHO_PIN 5

#define MAX_DISTANCE 200

const int Bomba1 = 11;

const int Bomba2 = 12;

const int Sonda = A0;

#define CaudalimetroA 2

#define CaudalimetroB 3

#define SOUND_SPEED 0.0171    // La mitad de la velocidad del sonido en el
aire, medida en [mm/us]

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

double microsegundos;

double distanciamm, distancia;

char mp;

unsigned char tension;

double tensionbomba2;

const double tfiltro = 10;
```

```
const double tfiltro1 = 5;

const double polofiltro = 0.9;

double pd, pd1;

double SensorValue, SensorValues;

double cauda, caudb, cuentas, cuentasant, caudala, caudalb, cuentasb,
cuentasantb;

void setup() {

 Serial.begin(115200);

  distancia = 51.56;

}
void loop() {

  ultrasonidos();

  SensorValue = analogRead(Sonda);

  pd1 = exp(-0.05 / tfiltro1);

  SensorValues = SensorValues * pd1 + (SensorValue / 20.0) * (1 - pd1);

  mp = Serial.read();

  if (mp == 'p') {

    /* Enviamos por el puerto serie el nivel medido en ultrasonidos
*/

    Serial.println(distancia);

    /* Esperamos a tener algun mensaje en el puerto serie y lo leemos
*/

    while (Serial.available() == 0);

    tension = Serial.read();

    /* Escribimos el valor de la tension que le debe llegar a la bomba
*/

    analogWrite(Bomba2, 0);
```

```
    analogWrite(Bomba1, tension);

  }

  else if (mp == 'o') {

    /* Enviamos por el puerto serie el nivel medido en ultrasonidos
*/

    Serial.println(SensorValues);

    /* Esperamos a tener algun mensaje en el puerto serie y lo leemos
*/

    while (Serial.available() == 0);

    tensionbomba2 = Serial.read();

    /* Escribimos el valor de la tension que le debe llegar a la bomba
*/

    analogWrite(Bomba1, 0);

    analogWrite(Bomba2, tensionbomba2);

  }
  else if (mp == 'm') {

    /* Enviamos por el puerto serie el nivel medido en ultrasonidos
*/

    Serial.println(distancia);

    Serial.println(SensorValues);

    /* Esperamos a tener algun mensaje en el puerto serie y lo leemos
*/

    while (Serial.available() == 0);

    tension = Serial.read();

    /* Esperamos a tener algun mensaje en el puerto serie y lo leemos
*/

    while (Serial.available() == 0);
```

```
    tensionbomba2 = Serial.read();

    /* Escribimos el valor de la tension que le debe llegar a la bomba
*/

    analogWrite(Bomba1, tension);

    analogWrite(Bomba2, tensionbomba2);

  }

  /*Codigo preparado por si se decide añadir caudalimetros.              */

  /* else if (mp = 'c') {

    attachInterrupt(0, DoCaudalimetroA, RISING);

    Serial.println(caudala);
   }

   else if (mp = 'v') {

    attachInterrupt(0, DoCaudalimetroB, RISING);

    Serial.println(caudalb);

   }*/


  delay(50);


}


/* Funcion que devuelve el nivel del agua, tiene un filtro             */
/* para que la curva sea más suave y con poco ruido.                   */

void ultrasonidos() {

  pd = exp(-0.05 / tfiltro);

  microsegundos = sonar.ping();

  distanciamm = microsegundos * SOUND_SPEED;

  distancia = distancia * pd + distanciamm * (1 - pd);
```

```
}
/* Funciones que devuelven el caudal del agua.       */

/*
void DoCaudalimetroA()

{
  cuentasant = cuentas;

  cuentas = micros();

  cauda = (2.331e7) / (cuentas - cuentasant); //2.331*10^7 constante que
relaciona 6l/min con 140Hz

  caudala = caudala * polofiltro + cauda * (1 - polofiltro);

}
void DoCaudalimetroB()

{
  cuentasantb = cuentasb;

  cuentasb = micros();

  caudb = (2.331e7) / (cuentasb - cuentasantb); //2.331*10^7 constante que
relaciona 6l/min con 140Hz

  caudalb = caudalb * polofiltro + caudb * (1 - polofiltro);

}
*/
```

1.14.2 Código Matlab

```
27/08/15 22:58  C:\Users\Usuario\Desktop\TFG\Arduino.m      1 of 3

% Autor: Aitor Roig Alemany
% Fecha: Agosto 2015
% "Sistema de control automático de una planta de laboratorio
% basada en 2 depósitos"


% Aplicación inicial, en la cuál escogemos que sistema analizar
% (sistema control bucle abierto, sistema control bucle cerrado,
% Multivariable bucle abierto, Multivariable bucle cerrado) y
% también escogemos el puerto donde se conecta el Arduino, establece
% la conexión con este y la cierra.



function varargout = Arduino(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Arduino_OpeningFcn, ...
                   'gui_OutputFcn',  @Arduino_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

%Función que abre la aplicación
function Arduino_OpeningFcn(hObject, ~, handles, varargin)


%con la siguiente instrucción se carga la imagen que sale al inicio.
x=imread('logo1.jpg');

handles.axes2=imshow(x);

handles.output = hObject;

guidata(hObject, handles);


function varargout = Arduino_OutputFcn(~, eventdata, handles)

varargout{1} = handles.output;

% Función donde se edita el puerto al que hemos conectado el Arduino

function Editar_Puerto_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');
end

% Función donde se establece la conexión con el puerto serie.

function Conexion_Callback(hObject, eventdata, handles)

   global arduino1

   % Almacenamos el Puerto donde hemos conectado el Arduino

   Puerto=get(handles.Editar_Puerto,'String');

   % Definimos la conexión serie

   arduino1=serial(Puerto,'BaudRate',115200','Terminator','CR/LF');

   % Abrimos la conexión serie

   fopen(arduino1);

   % Canviamos el color del boton e imformamos que nos hemos conectado

   set(handles.Conexion,'BackgroundColor','g');
   set(handles.Conexion,'String','Conectado');

% Función donde se cierra la conexión con el puerto serie, y la aplicación.

function Salir_Callback(hObject, eventdata, handles)

global arduino1

% Cerrar y eliminar el puerto serie.

fclose(arduino1);

delete(arduino1);

% Cerrar aplicación

close(gcbf)

%Selección de la ventana a abrir a continuación.

function escoger_ventana_Callback(hObject, eventdata, handles)

% En función del valor obtenido se abre una ventana u otra.
choose=get(handles.escoger_ventana,'Value');

switch choose
```

```
    case 1

        Sistemacontrol_bucleabierto;

    case 2

        Sistemacontrol_cerrado;

    case 3

        Multivariable_bucle_abierto;

    case 4

        Multivariable_bucle_cerrado;

end


function escoger_ventana_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end
```

27/08/15 23:10 C:\Users...\Sistemacontrol_bucleabierto.m 1 of 5

```
% Autor: Aitor Roig Alemany
% Fecha: Agosto 2015
% "Sistema de control automático de una planta de laboratorio
% basada en 2 depósitos"


% Aplicación en la que se escoge la tensión que le llega a la bomba,
% mediante el PWM del Arduino, ademas se puede escoger en que deposito
% medir el nivel del agua.

function varargout = Sistemacontrol_bucleabierto(varargin)

gui_Singleton = 1;

gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Sistemacontrol_bucleabierto_OpeningFcn, ...
                   'gui_OutputFcn',  @Sistemacontrol_bucleabierto_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);

if nargin && ischar(varargin{1})

    gui_State.gui_Callback = str2func(varargin{1});

end

if nargout

    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});

else

    gui_mainfcn(gui_State, varargin{:});

end


function Sistemacontrol_bucleabierto_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

guidata(hObject, handles);


function varargout = Sistemacontrol_bucleabierto_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;


function Bombal_Callback(hObject, eventdata, handles)

global  bombaval
```

```
bombaval=get(handles.Bomba1,'Value');

set(handles.ValorSlider,'String',bombaval);


function Bomba1_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor',[.9 .9 .9]);

end


function bombaval_Callback(hObject, eventdata, handles)

global bombaval

bombaval=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico


% Deteccion de errores.

if isnan(bombaval)

errordlg('El valor debe ser numérico','ERROR');

set(handles.bombaval,'String','');

return


end
%identificando rango

if(bombaval<-0 || bombaval>255)

errordlg('El Rango  incorrecto (0,255)','ERROR');

set(handles.bombaval,'String','');

return

end

set(handles.ValorSlider,'String',bombaval);

set(handles.Bomba1,'Value',bombaval);
```

```
function bombaval_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end


function boton_empezar_Callback(hObject, eventdata, handles)

    global peri
    global m_1 m_2
    global t
    global u_1 u_2
    global z x z_2 x_2
    global a_1 a_2
    global r0_1 y0_1 y1_1 r1_1 r0_2 y0_2 y1_2 r1_2
    global dt
    global bucle_abierto

    dt=0.5;         %Periodo que se ejecutara el timer
    set(handles.boton_empezar,'BackgroundColor','g');
    set(handles.boton_empezar,'String','Adquiriendo datos');



    a_1=51.56;
    a_2=0;
    r0_1=0;
    y0_1=0;
    y1_1=0;
    r1_1=0;
    r0_2=0;
    y0_2=0;
    y1_2=0;
    r1_2=0;
    z=0;
    u_2=0;
    m_2=0;
    z_2=0;
    x_2=0;
    t=0;
    m_1=0;
    peri=0;
    u_1=0;
    x=0;


    % Se espera 2 segundos, por si se inicia la aplicación demasiado
    % rapido.

    pause(2)
```

```
    % Se crea el Timer

    bucle_abierto=timer('TimerFcn',@arduinotimer_bucleabierto,...
              'ExecutionMode', 'fixedRate', ...
              'StopFcn','disp(''Timer has stopped.'')',...
              'Period',dt);

     % Crea una figura donde se refresentaran los plots.

    figure;

    % Se inicializa el timer

    start(bucle_abierto);
function guardarvar_Callback(hObject, eventdata, handles)

global t

global z

global m_1

% Se crea una matriz la cual se guardara en el fichero correspondiente

control=[t' z' m_1'];

% Se guarda en un fichero la matriz anterior.

save sistemacontrol_bucleabierto.txt control -ascii -tabs


function stop_Callback(hObject, eventdata, handles)

global bucle_abierto

% Para y elimina el timer

stop(bucle_abierto);

delete(bucle_abierto);

set(handles.boton_empezar,'BackgroundColor','w');

set(handles.boton_empezar,'String','Tomar Medidas');


% Funcion para mostrar el valor de la tensión de la bomba en PWM.

function ValorSlider_CreateFcn(hObject, eventdata, handles)
```

```
function Deposit1_Callback(hObject, eventdata, handles)

 global choose_deposito

 % Declaramos la instruccion a enviar a Arduino
 % a cerca de que deposito queremos ver el nivel

 choose_deposito='p';


 % Si se ha escogido este deposito se pone dicho boton en verde
 % y el contrario en rojo

 if( choose_deposito=='p')

     set(handles.Deposit1,'BackgroundColor','g');

     set(handles.Deposit2,'BackgroundColor','r');

 end

function Deposit2_Callback(hObject, eventdata, handles)

 global choose_deposito


 % Declaramos la instruccion a enviar a Arduino
 % a cerca de que deposito queremos ver el nivel


 choose_deposito='o';

 % Si se ha escogido este deposito se pone dicho boton en verde
 % y el contrario en rojo

 if( choose_deposito=='o')

    set(handles.Deposit2,'BackgroundColor','g');

    set(handles.Deposit1,'BackgroundColor','r');

 end
```

27/08/15 23:13 C:\Users\U...\arduinotimer_bucleabierto.m 1 of 2

```
% Autor: Aitor Roig Alemany
% Fecha: Agosto 2015
% "Sistema de control automático de una planta de laboratorio
% basada en 2 depósitos"


% Función la cual llamamos desde la aplicación del sistema de control de
% bucle abierto, y en la cuál se envia la tensión que se quiera que tenga
% la bomba, además de escoger en que deposito observar el nivel.


function arduinotimer_bucleabierto(source,event)


        global arduino1
        global dt
        global peri
        global m_1
        global t z
        global a_1
        global y0_1

        global bombaval
        global choose_deposito


       % Escribimos que queremos leer del puerto

       fwrite(arduino1,choose_deposito,'uchar');

       % Leemos del puerto serie

       a_1=fscanf(arduino1,'%f')

       if (choose_deposito=='o')
           y0_1=a_1;
       else
           y0_1=51.56-a_1;
       end


       m_1=[m_1,y0_1];
       peri=peri+dt;
       t=[t,peri];

       % Escribimos en el puerto serie el valor de la tensión de la bomba.

       fwrite(arduino1,bombaval,'uint8');


       z=[z,bombaval];

       %Con el subplot apareceran varios plots en la misma figura.

       subplot(2,1,1);
```

```
        plot(t,m_1,'b');
        subplot(2,1,2);
        plot(t,z,'r');


    end
```

27/08/15 23:14 C:\Users\Usua...\Sistemacontrol_cerrado.m 1 of 8

```
% Autor: Aitor Roig Alemany
% Fecha: Agosto 2015
% "Sistema de control automático de una planta de laboratorio
% basada en 2 depósitos"


% Aplicación en la que se escoge la referencia del sistema de control y en
% la que se añaden los datos necesarios para hacer el sistema de control
% de bucle cerrado.



function varargout = Sistemacontrol_cerrado(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Sistemacontrol_cerrado_OpeningFcn, ...
                   'gui_OutputFcn',  @Sistemacontrol_cerrado_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);

if nargin && ischar(varargin{1})

    gui_State.gui_Callback = str2func(varargin{1});

end

if nargout

    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});

else

    gui_mainfcn(gui_State, varargin{:});

end
function Sistemacontrol_cerrado_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

guidata(hObject, handles);

function varargout = Sistemacontrol_cerrado_OutputFcn(hObject, eventdata,handles)

varargout{1} = handles.output;


function boton_empezar_Callback(hObject, eventdata, handles)

    global peri
    global m_1
    global t
    global u_1
    global z x
```

27/08/15 23:14 C:\Users\Usua...\Sistemacontrol_cerrado.m 2 of 8

```
    global D_1 I_1
    global a_1
    global r0_1 y0_1 y1_1 r1_1
    global dt
    global ultrasounds


    dt=0.5;      %Periodo que se ejecutara el timer

    set(handles.boton_empezar,'BackgroundColor','g');

    set(handles.boton_empezar,'String','Adquiriendo datos');



    a_1=51.56;
    D_1=0;
    I_1=0;
    r0_1=0;
    y0_1=0;
    y1_1=0;
    r1_1=0;
    z=0;
    t=0;
    m_1=0;
    peri=0;
    u_1=0;
    x=0;
    dt=0.5;

    % Se espera 2 segundos, por si se inicia la aplicación demasiado
    % rapido.

    pause(2)

    % Se crea el Timer

    ultrasounds=timer('TimerFcn',@arduinotimer,...
              'ExecutionMode', 'fixedRate', ...
              'Period',dt);
    % Crea una figura donde se refresentaran los plots.
    figure;

    % Se inicializa el timer
    start(ultrasounds);


function guardarvar_Callback(hObject, eventdata, handles)

global t
global z
global m_1

% Se crea una matriz la cual se guardara en el fichero correspondiente
```

```
ultra_bomba_contraria=[t' z' m_1'];

% Se guarda en un fichero la matriz anterior.

save ultrasonidos_con_bomba_contrario_control_cerrado.txt ultra_bomba_contraria -ascii
-tabs



function kp_Callback(hObject, eventdata, handles)

global Kp

Kp=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(Kp)

errordlg('El valor debe ser numérico','ERROR');

set(handles.kp,'String','');

return

end

function kp_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end



function td_Callback(hObject, eventdata, handles)

global Td


Td=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(Td)
```

```
errordlg('El valor debe ser numérico','ERROR');

set(handles.td,'String','');

return

end


function td_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end



function ti_Callback(hObject, eventdata, handles)

global Ti


Ti=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(Ti)

errordlg('El valor debe ser numérico','ERROR');

set(handles.ti,'String','');

return

end


function ti_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');
end



function n_Callback(hObject, eventdata, handles)
```

27/08/15 23:14 C:\Users\Usua...\Sistemacontrol_cerrado.m 5 of 8

```
global N


N=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(N)

errordlg('El valor debe ser numérico','ERROR');

set(handles.n,'String','');

return

end

function n_CreateFcn(hObject, eventdata, handles)


if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');
end



function referencia_Callback(hObject, eventdata, handles)

global  ref_1;


ref_1=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(ref_1)

errordlg('El valor debe ser numérico','ERROR');

set(handles.referencia,'String','');

return

end
%identificando rango
```

```
if(ref_1<-0 || ref_1>50)

errordlg('El Rango  incorrecto (0,50)','ERROR');

set(handles.referencia,'String','');

return

end


set(handles.sliderreferencia1,'Value',ref_1);

set(handles.valsliderreferencia1,'String',ref_1);


function referencia_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end

% Escoger la referencia

function sliderreferencia1_Callback(hObject, eventdata, handles)

global  ref

ref=get(handles.sliderreferencia1,'Value');

set(handles.valsliderreferencia1,'String',ref);


function sliderreferencia1_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor',[.9 .9 .9]);

end


function B_Callback(hObject, eventdata, handles)

global b


b=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
```

27/08/15 23:14 C:\Users\Usua...\Sistemacontrol_cerrado.m 7 of 8

```
%identifica valor no numerico

% Deteccion de errores.

if isnan(b)

errordlg('El valor debe ser numérico','ERROR');

set(handles.B,'String','');

return

end

function B_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end


function C_Callback(hObject, eventdata, handles)

global c

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

c=str2double(get(hObject,'String'));

% Deteccion de errores.

if isnan(c)

errordlg('El valor debe ser numérico','ERROR');

set(handles.C,'String','');

return

end

function C_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');
end
```

27/08/15 23:14 C:\Users\Usua...\Sistemacontrol_cerrado.m 8 of 8

```
function stop_Callback(hObject, eventdata, handles)
global ultrasounds
% Para y elimina el timer
stop(ultrasounds);
delete(ultrasounds);
set(handles.boton_empezar,'BackgroundColor','w');
set(handles.boton_empezar,'String','Tomar Medidas');
% Funcion para mostrar el valor de la referencia.
function valsliderreferencia1_CreateFcn(hObject, eventdata, handles)
```

27/08/15 23:14 C:\Users\Usuario\Deskto...\arduinotimer.m 1 of 2

```
% Autor: Aitor Roig Alemany
% Fecha: Agosto 2015
% "Sistema de control automático de una planta de laboratorio
% basada en 2 depósitos"


% Función la cual llamamos desde la aplicación del sistema de control de
% bucle cerrado, y en la cuál se hacen los calculos correspondientes
% para lograr un buen sistema de control.


function arduinotimer(source,event)


        global arduino1
        global dt
        global peri
        global m_1
        global t z x
        global a_1
        global r0_1 y0_1 y1_1 r1_1
        global u_1
        global pd_1 qd_1 qi_1 D_1 I_1
        global choose_deposito
        global Kp Ti Td N b c ref_1

        umin_1=0;
        umax_1=255;

        % Escribimos que queremos leer del puerto

        fwrite(arduino1,choose_deposito,'uchar');

        % Leemos del puerto serie

        a_1=fscanf(arduino1,'%f');

        if (choose_deposito=='o')
            y0_1=a_1;
        else
            y0_1=51.56-a_1;
        end

        r0_1=ref_1

        m_1=[m_1,y0_1];

        peri=peri+dt;

        t=[t,peri];

        % Algoritmo de control.

        pd_1=Td/(Td+N*dt);
```

```
        qd_1=(Kp*Td*N)/(Td+N*dt);

        qi_1=Kp*dt/Ti;

        D_1=pd_1*D_1+qd_1*(c*r0_1-c*r1_1-y0_1+y1_1);

        I_1=I_1+qi_1*(r0_1-y0_1);

        u_1=Kp*(b*r0_1-y0_1)+D_1+I_1;

         if ((u_1<umin_1)&(ref_1<y0_1))|((u_1>umax_1)&(ref_1>y0_1))
             I_1=I_1-qi_1*(r0_1-y0_1);
         end

         if (u_1<umin_1)
             u_1=umin_1;
         end


         if (u_1>umax_1)
             u_1=umax_1;
         end


         % Escribimos en el puerto serie el valor de la tensión de la bomba.

         fwrite(arduino1,u_1,'uint8');


         x=[x,ref_1];
         z=[z,u_1];

         %Con el subplot apareceran varios plots en la misma figura.

         subplot(2,1,1);

         plot(t,m_1,'b',t,x,'k');

         subplot(2,1,2);

         plot(t,z,'r');

         y1_1=y0_1;
         r1_1=r0_1;


    end
```

27/08/15 23:16 C:\Users...\Multivariable_bucle_abierto.m 1 of 6

```
% Autor: Aitor Roig Alemany
% Fecha: Agosto 2015
% "Sistema de control automático de una planta de laboratorio
% basada en 2 depósitos"


% Aplicación en la que se escoge la tensión que les llega a las bombas,
% mediante el PWM del Arduino.


function varargout = Multivariable_bucle_abierto(varargin)

gui_Singleton = 1;

gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Multivariable_bucle_abierto_OpeningFcn, ...
                   'gui_OutputFcn',  @Multivariable_bucle_abierto_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);

if nargin && ischar(varargin{1})

    gui_State.gui_Callback = str2func(varargin{1});

end

if nargout

    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});

else

    gui_mainfcn(gui_State, varargin{:});

end


function Multivariable_bucle_abierto_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

guidata(hObject, handles);



function varargout = Multivariable_bucle_abierto_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;
```

27/08/15 23:16 C:\Users...\Multivariable bucle abierto.m 2 of 6

```
% Editar el valor de la bomba en PWM

function Bombal_Callback(hObject, eventdata, handles)

global  bombaval

bombaval=get(handles.Bombal,'Value');

set(handles.ValorSlider,'String',bombaval);



function Bombal_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor',[.9 .9 .9]);

end



function bombaval_Callback(hObject, eventdata, handles)

global bombaval

bombaval=str2double(get(hObject,'String'))

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores

if isnan(bombaval)

    errordlg('El valor debe ser numérico','ERROR');

set(handles.bombaval,'String','');

return

end

%identificando rango

if(bombaval<-0 || bombaval>255)

    errordlg('El Rango  incorrecto (0,255)','ERROR');

set(handles.bombaval,'String','');

return
```

```
end

set(handles.ValorSlider,'String',bombaval);

set(handles.Bomba1,'Value',bombaval);


function bombaval_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');
end



function bomba2_Callback(hObject, eventdata, handles)

global  bombaval_2

bombaval_2=get(handles.bomba2,'Value');

set(handles.ValorSlider2,'String',bombaval_2);

function bomba2_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


function bombaval_2_Callback(hObject, eventdata, handles)

global bombaval_2

bombaval_2=str2double(get(hObject,'String'))

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores

if isnan(bombaval_2)

    errordlg('El valor debe ser numérico','ERROR');

set(handles.bombaval_2,'String','');

return

end
```

27/08/15 23:16 C:\Users...\Multivariable bucle abierto.m 4 of 6

```
%identificando rango

if(bombaval_2<-0 || bombaval_2>255)

    errordlg('El Rango  incorrecto (0,255)','ERROR');

set(handles.bombaval_2,'String','');

return

end

set(handles.ValorSlider2,'String',bombaval_2);

set(handles.bomba2,'Value',bombaval_2);

function bombaval_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');
end


% Funcion para mostrar el valor de la tensión de la bomba en PWM.


function ValorSlider2_CreateFcn(hObject, eventdata, handles)




function boton_empezar_Callback(hObject, eventdata, handles)

    global peri
    global m_1 m_2
    global t
    global u_1 u_2
    global z x z_2 x_2
    global a_1 a_2
    global r0_1 y0_1 y1_1 r1_1 r0_2 y0_2 y1_2 r1_2
    global dt
    global control_multi


    dt=0.5;     %Periodo que se ejecutara el timer

    set(handles.boton_empezar,'BackgroundColor','g');

    set(handles.boton_empezar,'String','Adquiriendo datos');
```

```
    a_1=51.56;
    a_2=0;
    r0_1=0;
    y0_1=0;
    y1_1=0;
    r1_1=0;
    r0_2=0;
    y0_2=0;
    y1_2=0;
    r1_2=0;
    z=0;
    u_2=0;
    m_2=0;
    z_2=0;
    x_2=0;
    t=0;
    m_1=0;
    peri=0;
    u_1=0;
    x=0;

    % Se espera 2 segundos, por si se inicia la aplicación demasiado
    % rapido.

    pause(2)

    % Se crea el Timer

    control_multi=timer('TimerFcn',@multicontrol,...
              'ExecutionMode', 'fixedRate', ...
              'StopFcn','disp(''Timer has stopped.'')',...
              'Period',dt);


    % Crea una figura donde se refresentaran los plots.

    figure;

    % Se inicializa el timer

    start(control_multi);

function guardarvar_Callback(hObject, eventdata, handles)

global t
global z z_2
global m_1 m_2


% Se crean matrices las cuales se guardaran en los ficheros
% correspondientes

ultra_su_bomba=[t' z_2' m_1'];
```

```
ultra_bomba_contraria=[t' z' m_1'];

sensor_su_bomba=[t' z' m_2'];

sensor_bomba_contraria=[t' z_2' m_2'];

% Se guarda en  ficheros las matrices anteriores.

save ultrasonidos_con_su_bomba.txt ultra_su_bomba -ascii -tabs

save ultrasonidos_con_bomba_contrario.txt ultra_bomba_contraria -ascii -tabs

save sensor_con_su_bomba.txt sensor_su_bomba -ascii -tabs

save sensor_con_bomba_contrario.txt sensor_bomba_contraria -ascii -tabs


% Funcion para mostrar el valor de la tensión de la bomba en PWM.

function ValorSlider_CreateFcn(hObject, eventdata, handles)


function stop_Callback(hObject, eventdata, handles)

global control_multi


% Para y elimina el timer

stop(control_multi);

delete(control_multi);

set(handles.boton_empezar,'BackgroundColor','w');

set(handles.boton_empezar,'String','Tomar Medidas');
```

27/08/15 23:16 C:\Users\Usuario\Deskto...\multicontrol.m 1 of 2

```
% Autor: Aitor Roig Alemany
% Fecha: Agosto 2015
% "Sistema de control automático de una planta de laboratorio
% basada en 2 depósitos"


% Función la cual llamamos desde la aplicación del multivariable de bucle
% abierto, y en la cuál se envia la tensión que se quiera que tenga
% la bomba, además de escoger en que deposito observar el nivel.

function multicontrol(source,event)



        global arduino1
        global dt
        global peri
        global m_1 m_2
        global t z  z_2
        global a_1 a_2
        global  y0_1
        global  y0_2
        global bombaval bombaval_2

        % Escribimos que queremos leer del puerto

        fwrite(arduino1,'m','uchar');

        umin_1=0;
        umax_1=255;
        umin_2=0;
        umax_2=255;

        % Leemos del puerto serie

        a_1=fscanf(arduino1,'%f');
        a_2=fscanf(arduino1,'%f');

        y0_1=51.56-a_1;
        y0_2=a_2;

        m_1=[m_1,y0_1];
        m_2=[m_2,y0_2];
        peri=peri+dt;
        t=[t,peri];

        % Escribimos en el puerto serie el valor de la tensión de la bomba.

        fwrite(arduino1,bombaval,'uint8');
        fwrite(arduino1,bombaval_2,'uint8');

        % Almacenamos en un vector los datos.

        z=[z,bombaval];
```

27/08/15 23:16 C:\Users\Usuario\Deskto...\multicontrol.m 2 of 2

```
        z_2=[z_2,bombaval_2];

        %Con el subplot apareceran varios plots en la misma figura.

        subplot(4,1,1);
        plot(t,m_1,'b');

        subplot(4,1,2);
        plot(t,z_2,'r');

        subplot(4,1,3);
        plot(t,m_2,'c');

        subplot(4,1,4);
        plot(t,z,'m');


    end
```

27/08/15 23:18 C:\Use...\Multivariable_bucle_cerrado.m 1 of 15

```
% Autor: Aitor Roig Alemany
% Fecha: Agosto 2015
% "Sistema de control automático de una planta de laboratorio
% basada en 2 depósitos"


% Aplicación en la que se escoge la referencia del sistema de control y en
% la que se añaden los datos necesarios para hacer el sistema de control
% multivariable de bucle cerrado.


function varargout = Multivariable_bucle_cerrado(varargin)

gui_Singleton = 1;

gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Multivariable_bucle_cerrado_OpeningFcn, ...
                   'gui_OutputFcn',  @Multivariable_bucle_cerrado_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);

if nargin && ischar(varargin{1})

    gui_State.gui_Callback = str2func(varargin{1});

end

if nargout

    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else

    gui_mainfcn(gui_State, varargin{:});
end

function Multivariable_bucle_cerrado_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;


guidata(hObject, handles);


function varargout = Multivariable_bucle_cerrado_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

function boton_empezar_Callback(hObject, eventdata, handles)

    global ultrasounds
    global dt
    global peri
    global m_1 m_2
```

```
global t z x z_2 x_2
global a_1 a_2
global r0_1 y0_1 y1_1 r1_1
global r0_2 y0_2 y1_2 r1_2
global u_1 u_2
global pd_1 qd_1 qi_1 D_1 I_1 D_2 I_2 pd_2 qd_2 qi_2
global k21 k12
global Kp Ti Td N b c ref_1 ref_2 Kp_2 Ti_2 Td_2 N_2 b_2 c_2

dt=0.5;      %Periodo que se ejecutara el timer

set(handles.boton_empezar,'BackgroundColor','g');
set(handles.boton_empezar,'String','Adquiriendo datos');

a_1=51.56;;
a_2=0;
r0_1=0;
y0_1=0;
y1_1=0;
r1_1=0;
r0_2=0;
y0_2=0;
y1_2=0;
r1_2=0;
z=0;
z_2=0;
x=0;
x_2=0;
u_1=0;
u_2=0;
dt=0.5;
t=0;
m_1=0;
m_2=0;
peri=0;
D_1=0;
I_1=0;
D_2=0;
I_2=0;
k21=0;
k12=0;
pd_1=0;
qd_1=0;
qi_1=0;
pd_2=0;
qd_2=0;
qi_2=0;
Kp=0;
Ti=0;
Td=0;
N=0;
b=0;
c=0;
ref_1=0;
ref_2=0;
```

```
    Kp_2=0;
    Ti_2=0;
    Td_2=0;
    N_2=0;
    b_2=0;
    c_2=0;


    % Se espera 2 segundos, por si se inicia la aplicación demasiado
    % rapido.
    pause(2)

    % Se crea el Timer

    ultrasounds=timer('TimerFcn',@multicontrol_buclecerrado,...
              'ExecutionMode', 'fixedRate', ...
              'Period',dt);

    % Crea una figura donde se refresentaran los plots.
    figure;

    % Se inicializa el timer
    start(ultrasounds);


function guardarvar_Callback(hObject, eventdata, handles)

global t
global z z_2
global m_1 m_2

% Se crean las matrices que se guardaran en los ficheros correspondientes

ultra_su_bomba=[t' z_2' m_1'];

ultra_bomba_contraria=[t' z' m_1'];

sensor_su_bomba=[t' z' m_2'];

sensor_bomba_contraria=[t' z_2' m_2'];

% Se guarda en ficheros las matrices anteriores.

save ultrasonidos_con_su_bomba_cerrado.txt ultra_su_bomba -ascii -tabs

save ultrasonidos_con_bomba_contrario_cerrado.txt ultra_bomba_contraria -ascii -tabs

save sensor_con_su_bomba_cerrado.txt sensor_su_bomba -ascii -tabs

save sensor_con_bomba_contrario_cerrado.txt sensor_bomba_contraria -ascii -tabs


function kp_Callback(hObject, eventdata, handles)
```

```
global Kp

Kp=str2double(get(hObject,'String'));
%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(Kp)

errordlg('El valor debe ser numérico','ERROR');

set(handles.kp,'String','');

return

end


function kp_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');
end



function td_Callback(hObject, eventdata, handles)

global Td


Td=str2double(get(hObject,'String'));
%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(Td)

errordlg('El valor debe ser numérico','ERROR');

set(handles.td,'String','');

return

end


function td_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');

end


function ti_Callback(hObject, eventdata, handles)

global Ti


Ti=str2double(get(hObject,'String'));
%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(Ti)

errordlg('El valor debe ser numérico','ERROR');

set(handles.ti,'String','');

return

end

function ti_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end


function n_Callback(hObject, eventdata, handles)


global N


N=str2double(get(hObject,'String'));
%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(N)

errordlg('El valor debe ser numérico','ERROR');
```

```
set(handles.n,'String','');

return

end


function n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end



function referencia_Callback(hObject, eventdata, handles)

global  ref_1;


ref_1=str2double(get(hObject,'String'));
%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(ref_1)

errordlg('El valor debe ser numérico','ERROR');

set(handles.referencia,'String','');

return


end

%identificando rango

if(ref_1<-0 || ref_1>50)

errordlg('El Rango  incorrecto (0,50)','ERROR');

set(handles.referencia,'String','');

return

end

set(handles.sliderreferencial,'Value',ref_1);
```

```
set(handles.valsliderreferencia1,'String',ref_1);


function referencia_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end

% Escoger la referencia

function sliderreferencia1_Callback(hObject, eventdata, handles)

global  ref

ref=get(handles.sliderreferencia1,'Value');

set(handles.valsliderreferencia1,'String',ref);

function sliderreferencia1_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor',[.9 .9 .9]);

end


function B_Callback(hObject, eventdata, handles)

global b


b=str2double(get(hObject,'String'));
%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(b)

errordlg('El valor debe ser numérico','ERROR');

set(handles.B,'String','');

return

end

function B_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');
end


function C_Callback(hObject, eventdata, handles)

global c


c=str2double(get(hObject,'String'));
%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.
if isnan(c)

errordlg('El valor debe ser numérico','ERROR');

set(handles.C,'String','');

return

end

function C_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end


function referencia_2_Callback(hObject, eventdata, handles)

global  ref_2;


ref_2=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(ref_2)

errordlg('El valor debe ser numérico','ERROR');
```

```
set(handles.referencia_2,'String','');

return

end

%identificando rango

if(ref_2<-0 || ref_2>50)

errordlg('El Rango  incorrecto (0,50)','ERROR');

set(handles.referencia_2,'String','');

return

end

set(handles.sliderreferencia2,'Value',ref_2);

set(handles.valsliderreferencia2,'String',ref_2);


function referencia_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end

% Escoger la referencia

function sliderreferencia2_Callback(hObject, eventdata, handles)

global  ref_2

ref_2=get(handles.sliderreferencia2,'Value');

set(handles.valsliderreferencia2,'String',ref_2);

function sliderreferencia2_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor',[.9 .9 .9]);

end


function kp_2_Callback(hObject, eventdata, handles)
```

27/08/15 23:18 C:\Use...\Multivariable bucle cerrado.m 10 of 15

```
global Kp_2

Kp_2=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(Kp_2)

errordlg('El valor debe ser numérico','ERROR');

set(handles.kp_2,'String','');

return

end


function kp_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end



function td_2_Callback(hObject, eventdata, handles)

global Td_2


Td_2=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.
if isnan(Td_2)

errordlg('El valor debe ser numérico','ERROR');

set(handles.td_2,'String','');

return

end

function td_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
```

```
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end


function ti_2_Callback(hObject, eventdata, handles)

global Ti_2


Ti_2=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.
if isnan(Ti_2)

errordlg('El valor debe ser numérico','ERROR');

set(handles.ti_2,'String','');

return

end

function ti_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end


function n_2_Callback(hObject, eventdata, handles)

global N_2


N_2=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(N_2)

errordlg('El valor debe ser numérico','ERROR');
```

```
set(handles.n_2,'String','');

return

end

function n_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end


function B_2_Callback(hObject, eventdata, handles)

global B_2

B_2=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(B_2)

errordlg('El valor debe ser numérico','ERROR');

set(handles.b_2,'String','');

return

end


function B_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end


function C_2_Callback(hObject, eventdata, handles)

global c_2
```

```
c_2=str2double(get(hObject,'String'));

%Almacenar valor ingresado y Transformar a formato double
%identifica valor no numerico

% Deteccion de errores.

if isnan(c_2)

errordlg('El valor debe ser numérico','ERROR');

set(handles.C_2,'String','');

return

end

function C_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end


function stop_Callback(hObject, eventdata, handles)

global ultrasounds

% Para y elimina el timer

stop(ultrasounds);

delete(ultrasounds);

set(handles.boton_empezar,'BackgroundColor','w');

set(handles.boton_empezar,'String','Tomar Medidas');


function k12_Callback(hObject, eventdata, handles)

global k12

k12=str2double(get(hObject,'String'));
% Almacenar valor ingresado y Transformar a formato double identifica valor
% no numerico

% Deteccion de errores.
if isnan(k12)
```

```
errordlg('El valor debe ser numérico','ERROR');

set(handles.k12,'String','');

return

end

function k12_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end

% Función que relaciona ambos sistemas.

function k21_Callback(hObject, eventdata, handles)

global k21


k21=str2double(get(hObject,'String'));

% Almacenar valor ingresado y Transformar a formato double identifica valor
% no numerico

% Deteccion de errores.
if isnan(k21)

errordlg('El valor debe ser numérico','ERROR');

set(handles.k21,'String','');

return

end

function k21_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Funciones para mostrar el valor de la referencia.

function valsliderreferencial_CreateFcn(hObject, eventdata, handles)

function valsliderreferencia2_CreateFcn(hObject, eventdata, handles)

% En esta función guardamos el valor del check y en función si esta pulsado
% el check o no sabremos si estan los dos sistema acoplados o no
```

27/08/15 23:18 C:\Use...\Multivariable bucle cerrado.m 15 of 15

```
function Acoplado_Callback(hObject, eventdata, handles)

global acoplado

acoplado=get(hObject,'Vaule');
```

27/08/15 23:18 C:\Users\U...\multicontrol_buclecerrado.m 1 of 3

```
% Autor: Aitor Roig Alemany
% Fecha: Agosto 2015
% "Sistema de control automático de una planta de laboratorio
% basada en 2 depósitos"


% Función la cual llamamos desde la aplicación del sistema multivariable
% bucle cerrado, y en la cuál se hacen los calculos correspondientes
% para lograr un buen sistema de control.


function multicontrol_buclecerrado(source,event)



        global arduino1
        global dt
        global peri
        global m_1 m_2
        global t z x z_2 x_2
        global a_1 a_2
        global r0_1 y0_1 y1_1 r1_1
        global r0_2 y0_2 y1_2 r1_2
        global u_1 u_2
        global pd_1 qd_1 qi_1 D_1 I_1 D_2 I_2 pd_2 qd_2 qi_2
        global k21 k12
        global Kp Ti Td N b c ref_1 ref_2 Kp_2 Ti_2 Td_2 N_2 b_2 c_2
        global acoplado

        % Escribimos que queremos leer del puerto

        fwrite(arduino1,'m','uchar');

        umin_1=0;
        umax_1=255;
        umin_2=0;
        umax_2=255;

       % Leemos del puerto serie

        a_1=fscanf(arduino1,'%f');
        a_2=fscanf(arduino1,'%f');

        y0_2=51.56-a_1;
        y0_1=a_2;

        r0_1=ref_1;
        r0_2=ref_2;

        m_1=[m_1,y0_1];
        m_2=[m_2,y0_2];

        peri=peri+dt;
        t=[t,peri];
```

27/08/15 23:18 C:\Users\U...\multicontrol buclecerrado.m 2 of 3

```
%Algoritmo de control.

pd_1=Td/(Td+N*dt);

qd_1=(Kp*Td*N)/(Td+N*dt);

qi_1=Kp*dt/Ti;

D_1=pd_1*D_1+qd_1*(c*r0_1-c*r1_1-y0_2+y1_2);

I_1=I_1+qi_1*(r0_1-y0_2);

u_1=Kp*(b*r0_1-y0_1)+D_1+I_1-k21*u_2*acoplado;


if ((u_1<umin_1)&(ref_1<y0_1))|((u_1>umax_1)&(ref_1>y0_1))

     I_1=I_1-qi_1*(r0_1-y0_1);

end

if (u_1<umin_1)
    u_1=umin_1;
end


if (u_1>umax_1)
    u_1=umax_1;
end

pd_2=Td_2/(Td_2+N_2*dt);

qd_2=(Kp_2*Td_2*N_2)/(Td_2+N_2*dt);

qi_2=Kp_2*dt/Ti_2;

D_2=pd_2*D_2+qd_2*(c_2*r0_2-c*r1_2-y0_1+y1_1);

I_2=I_2+qi_2*(r0_2-y0_1);

u_2=Kp*(b_2*r0_2-y0_2)+D_2+I_2-k12*u_1*acoplado;

if ((u_2<umin_2)&(ref_2<y0_2))|((u_2>umax_2)&(ref_2>y0_2))

    I_2=I_2-qi_1*(r0_2-y0_2);
end

if (u_2<umin_2)
   u_2=umin_2;
end


if (u_2>umax_2)
   u_2=umax_2;
```

27/08/15 23:18 C:\Users\U...\multicontrol buclecerrado.m 3 of 3

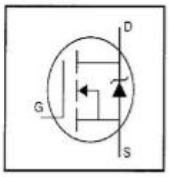
```
        end

      % Escribimos en el puerto serie el valor de la tensión de la bomba 1.
        fwrite(arduino1,u_1,'uint8');
        x=[x,ref_1];
        z=[z,u_1];
      % Escribimos en el puerto serie el valor de la tensión de la bomba 2.
        fwrite(arduino1,u_2,'uint8');
        x_2=[x_2,ref_2];
        z_2=[z_2,u_2];


        %Con el subplot apareceran varios plots en la misma figura.

        subplot(4,1,1);
        plot(t,m_1,'b',t,x,'k');

        subplot(4,1,2);
        plot(t,z,'r');

        subplot(4,1,3);
        plot(t,m_2,'c',t,x_2,'k');

        subplot(4,1,4);
        plot(t,z_2,'m');

        y1_1=y0_1;
        r1_1=r0_1;
        y1_2=y0_2;
        r1_2=r0_2;


    end
```

1.14.3 Datasheet MOSFET IRLIZ44GPbF

International
IOR Rectifier
HEXFET® Power MOSFET

PD-95754

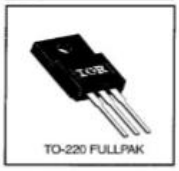
IRLIZ44GPbF



V_{DSS} = 60V

$R_{DS(on)}$ = 0.028Ω

I_D = 30A



- Isolated Package
- High Voltage Isolation= 2.5KVRMS ⑤
- Sink to Lead Creepage Dist.= 4.8mm
- Logic-Level Gate Drive
- $R_{DS(on)}$ Specified at V_{GS}=4V & 5V
- Fast Switching
- Ease of Paralleling
- Lead-Free

Description

Third Generation HEXFETs from International Rectifier provide the designer with the best combination of fast switching, ruggedized device design, low on-resistance and cost-effectiveness.

The TO-220 Fullpak eliminates the need for additional insulating hardware in commercial-industrial applications. The moulding compound used provides a high isolation capability and a low thermal resistance between the tab and external heatsink. This isolation is equivalent to using a 100 micron mica barrier with standard TO-220 product. The Fullpak is mounted to a heatsink using a single clip or by a single screw fixing.

Absolute Maximum Ratings

	Parameter	Max.	Units
I_D @ T_C = 25°C	Continuous Drain Current, V_{GS} @ 5.0 V	30	A
I_D @ T_C = 100°C	Continuous Drain Current, V_{GS} @ 5.0 V	21	
I_{DM}	Pulsed Drain Current ①	120	
P_D @ T_C = 25°C	Power Dissipation	48	W
	Linear Derating Factor	0.32	W/°C
V_{GS}	Gate-to-Source Voltage	±10	V
E_{AS}	Single Pulse Avalanche Energy ②	400	mJ
dv/dt	Peak Diode Recovery dv/dt ③	4.5	V/ns
T_J T_{STG}	Operating Junction and Storage Temperature Range	-55 to +175	°C
	Soldering Temperature, for 10 seconds	300 (1.6mm from case)	
	Mounting Torque, 6-32 or M3 screw	10 lbf•in (1.1 N•m)	

Thermal Resistance

	Parameter	Min.	Typ.	Max.	Units
$R_{\theta JC}$	Junction-to-Case	—	—	3.1	°C/W
$R_{\theta JA}$	Junction-to-Ambient	—	—	65	

www.irf.com 1

8/23/04

IRLIZ44GPbF

International
IOR Rectifier

Electrical Characteristics @ T_J = 25°C (unless otherwise specified)

	Parameter	Min.	Typ.	Max.	Units	Test Conditions
$V_{(BR)DSS}$	Drain-to-Source Breakdown Voltage	60	—	—	V	V_{GS}=0V, I_D= 250µA
$\Delta V_{(BR)DSS}/\Delta T_J$	Breakdown Voltage Temp. Coefficient	—	0.070	—	V/°C	Reference to 25°C, I_D= 1mA
$R_{DS(on)}$	Static Drain-to-Source On-Resistance	—	—	0.028	Ω	V_{GS}=5.0V, I_D=18A ④
		—	—	0.039		V_{GS}=4.0V, I_D=15A ④
$V_{GS(th)}$	Gate Threshold Voltage	1.0	—	2.0	V	V_{DS}=V_{GS}, I_D= 250µA
g_{fs}	Forward Transconductance	22	—	—	S	V_{DS}=25V, I_D=18A ④
I_{DSS}	Drain-to-Source Leakage Current	—	—	25	µA	V_{DS}=60V, V_{GS}=0V
		—	—	250		V_{DS}=48V, V_{GS}=0V, T_J=150°C
I_{GSS}	Gate-to-Source Forward Leakage	—	—	100	nA	V_{GS}=10V
	Gate-to-Source Reverse Leakage	—	—	-100		V_{GS}=-10V
Q_g	Total Gate Charge	—	—	66	nC	I_D=51A
Q_{gs}	Gate-to-Source Charge	—	—	12		V_{DS}=48V
Q_{gd}	Gate-to-Drain ("Miller") Charge	—	—	43		V_{GS}=5.0V See Fig. 6 and 13 ④
$t_{d(on)}$	Turn-On Delay Time	—	17	—	ns	V_{DD}=30V
t_r	Rise Time	—	230	—		I_D=51A
$t_{d(off)}$	Turn-Off Delay Time	—	42	—		R_G=4.6Ω
t_f	Fall Time	—	110	—		R_D=0.56Ω See Figure 10 ④
L_D	Internal Drain Inductance	—	4.5	—	nH	Between lead, 6 mm (0.25in.) from package and center of die contact
L_S	Internal Source Inductance	—	7.5	—		
C_{iss}	Input Capacitance	—	3300	—	pF	V_{GS}=0V
C_{oss}	Output Capacitance	—	1200	—		V_{DS}= 25V
C_{rss}	Reverse Transfer Capacitance	—	200	—		f=1.0MHz See Figure 5
C	Drain to Sink Capacitance	—	12	—	pF	f=1.0MHz

Source-Drain Ratings and Characteristics

	Parameter	Min.	Typ.	Max.	Units	Test Conditions
I_S	Continuous Source Current (Body Diode)	—	—	30	A	MOSFET symbol showing the integral reverse p-n junction diode.
I_{SM}	Pulsed Source Current (Body Diode) ①	—	—	120		
V_{SD}	Diode Forward Voltage	—	—	2.5	V	T_J=25°C, I_S=30A, V_{GS}=0V ④
t_{rr}	Reverse Recovery Time	—	90	180	ns	T_J=25°C, I_F=51A
Q_{rr}	Reverse Recovery Charge	—	0.65	1.3	µC	di/dt=100A/µs ④
t_{on}	Forward Turn-On Time	Intrinsic turn-on time is negligible (turn-on is dominated by L_S+L_D)				

Notes:

① Repetitive rating; pulse width limited by max. junction temperature (See Figure 11)

② V_{DD}=25V, starting T_J=25°C, L=518µH R_G=25Ω, I_{AS}=30A (See Figure 12)

③ I_{SD}≤51A, di/dt≤250A/µs, V_{DD}≤$V_{(BR)DSS}$, T_J≤175°C

④ Pulse width ≤ 300 µs; duty cycle ≤2%.

⑤ t=60s, f=60Hz

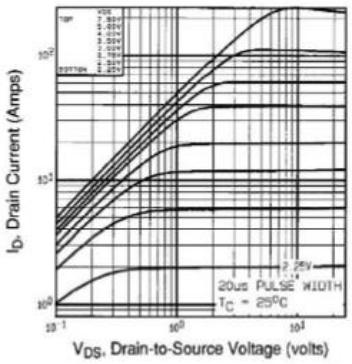


Fig 1. Typical Output Characteristics, TC=25°C

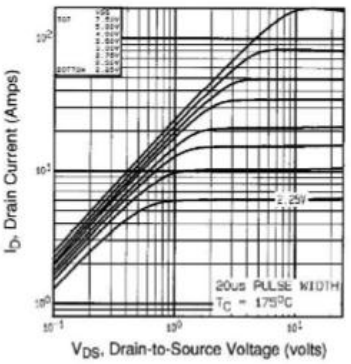


Fig 2. Typical Output Characteristics, TC=175°C

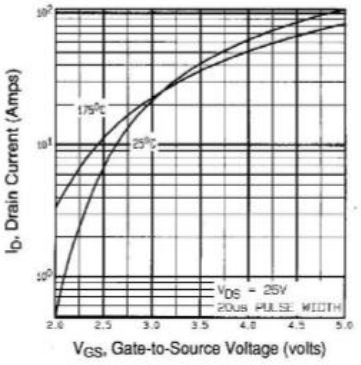


Fig 3. Typical Transfer Characteristics

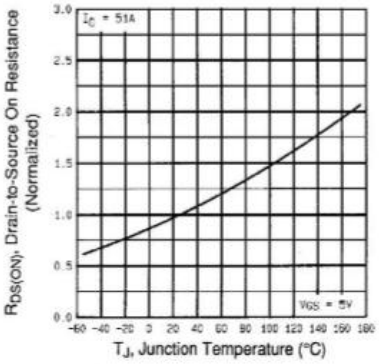


Fig 4. Normalized On-Resistance Vs. Temperature

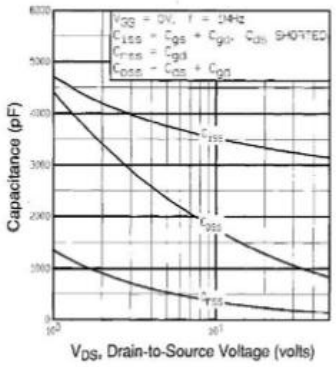


Fig 5. Typical Capacitance Vs. Drain-to-Source Voltage

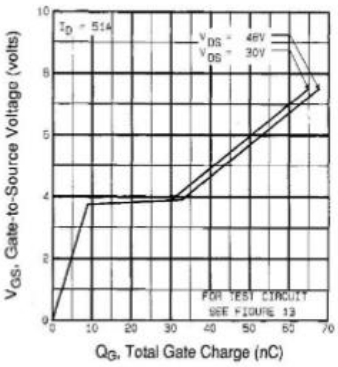


Fig 6. Typical Gate Charge Vs. Gate-to-Source Voltage

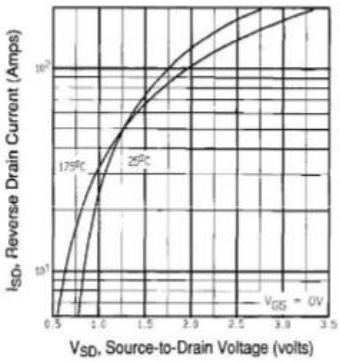


Fig 7. Typical Source-Drain Diode Forward Voltage

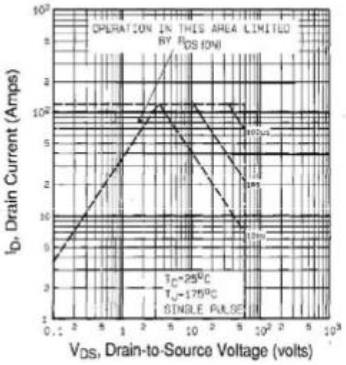


Fig 8. Maximum Safe Operating Area

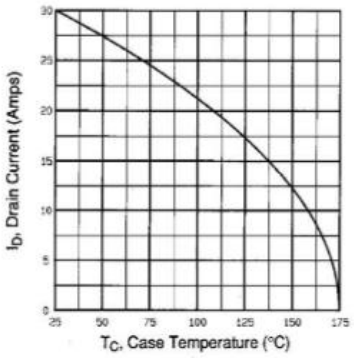


Fig 9. Maximum Drain Current Vs. Case Temperature

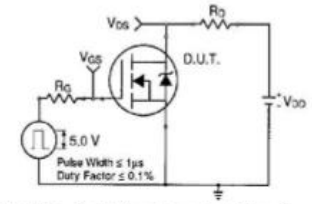


Fig 10a. Switching Time Test Circuit

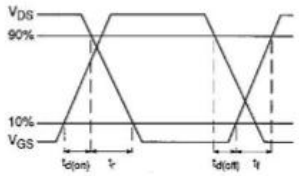


Fig 10b. Switching Time Waveforms

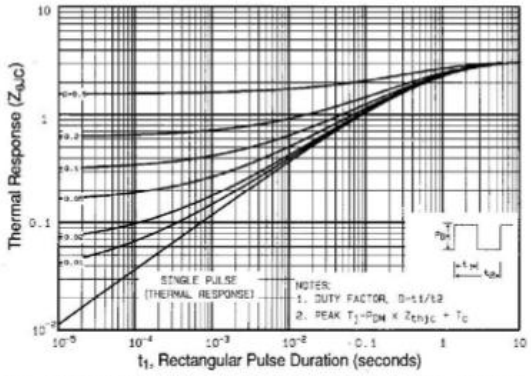


Fig 11. Maximum Effective Transient Thermal Impedance, Junction-to-Case

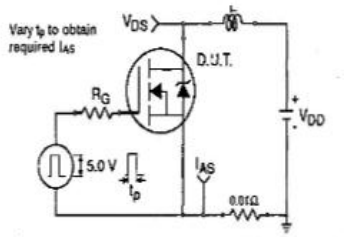


Fig 12a. Unclamped Inductive Test Circuit

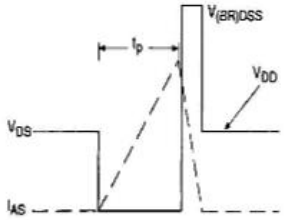


Fig 12b. Unclamped Inductive Waveforms

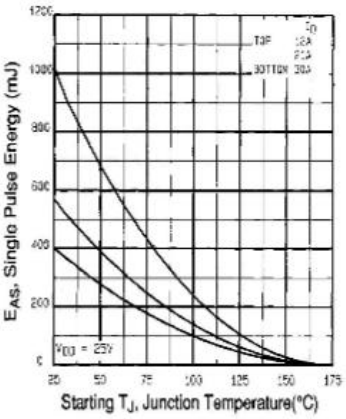


Fig 12c. Maximum Avalanche Energy Vs. Drain Current

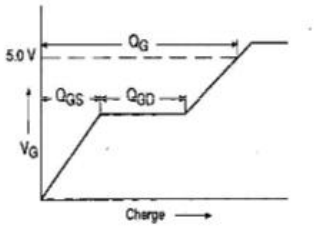


Fig 13a. Basic Gate Charge Waveform

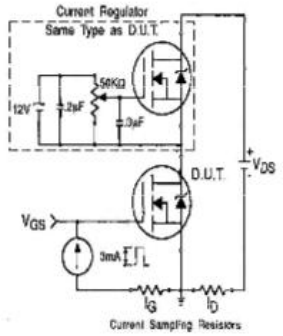


Fig 13b. Gate Charge Test Circuit

Pliego de Condiciones

2.1. Placa

El MOSFET a poner a la placa, deberá ser el IRLIZ44GPbF, o uno con unas características que mejoren a este:

- V_{DSS} = 60V
- R_{DS} = 0.028Ω, con una V_{GS}=5V
- I_D = 30A

Para mejorarlo, la R_{DS} debe ser menor a esa tensión de puerta de 5 V, la V_{DSS} mayor y la I_D mayor.

El diodo puede ser cualquiera que admita un mínimo de 2A.

Para poder soportar correctamente la corriente, el espesor de cobre de la placa deberá ser al menos de 35 um y el material de la placa deber ser baquelita o fibra de vidrio.

2.2. Ordenador

Las características mínimas que debe tener el ordenador es:

- Procesador — Intel® Core™ i5-2410 CPU @ 2.30GHz
- Memoria instalada(RAM) — 4,00 GB
- Tipo de sistema — Sistema operativo de 64 bits, procesador x64
- Sistema Operativo — Windows 7 o superior
- JAVA 8 o superior

2.3. Matlab

La versión Matlab necesaria será la *R2011b* o superior. Además debe contener los archivos donde se encuentra el código del programa, detallados en el apartado 1.9.

2.4. Arduino

El código se ha hecho para la placa Arduino MEGA 2560, el cual posee 54 pines digitales que funcionan como entrada/salida, 16 entradas análogas, un cristal oscilador de 16MHz, una conexión USB, un botón de reset, y una entrada para la alimentación de la placa. La comunicación entre el ordenador y Arduino se produce a través del puerto serie.

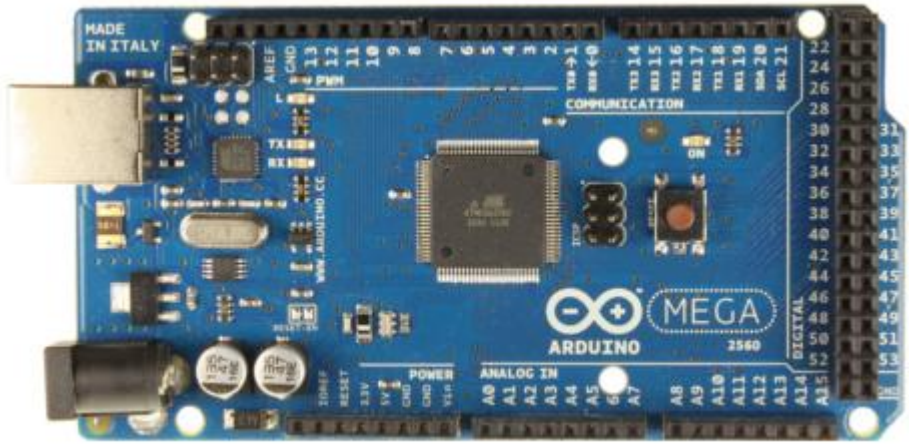


Figura 84. Arduino MEGA 2560

Las especificaciones de este son las siguientes:

Microcontroller	ATmega1280
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	128 KB of which 4 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Figura 85.Especificaciones Arduino MEGA 2560

Además el sensor de ultrasonidos debe ser el HC-SRC04, cuyas características están descritas en el apartado 1.6.

Presupuesto

Sistemas de depósitos

Cada sistema de depósitos cuesta 2500€ por lo tanto al tener 2 el precio será de 5000€. En el precio, está incluido el sensor de nivel capacitivo.

Software

El único software que tiene coste, es el MATLAB, ya que los otros programas son gratuitos. El precio de este es 2500€.

Fabricación PCB

Fabricar una PCB cuesta 20€, si se decidiera realizar más el precio bajaría, siendo el precio de 10€ para la fabricación de 10.000 PCB.

Componentes PCB

Componente	Referencia	Precio (€)	Unidades
MOSFET	IRLIZ44GPbF	2.05	4
Diodo		0.16	4
Resistencia 1K		0.03	4
Resistencia 100K		0.04	4
Conector 8 pines		1.23	1
Conector 5 pines		1.31	1
Conector 2 pines		0.93	1
Condensador 220nF		1.12	1

Tabla 8.Componentes electrónicos precio

Otros

Objeto	Precio (€)	Unidades
Arduino MEGA	19	1
Sensor Ultrasonidos	3	1
Cables	4	1

Tabla 9. Componentes necesarios precio

En el precio de los cabes se encuentra todos los cables utilizados.

Horas personal

Profesional	Precio (€)	Horas
Operario	25	2
Ingeniero Hardware	40	50
Ingeniero Software	40	125

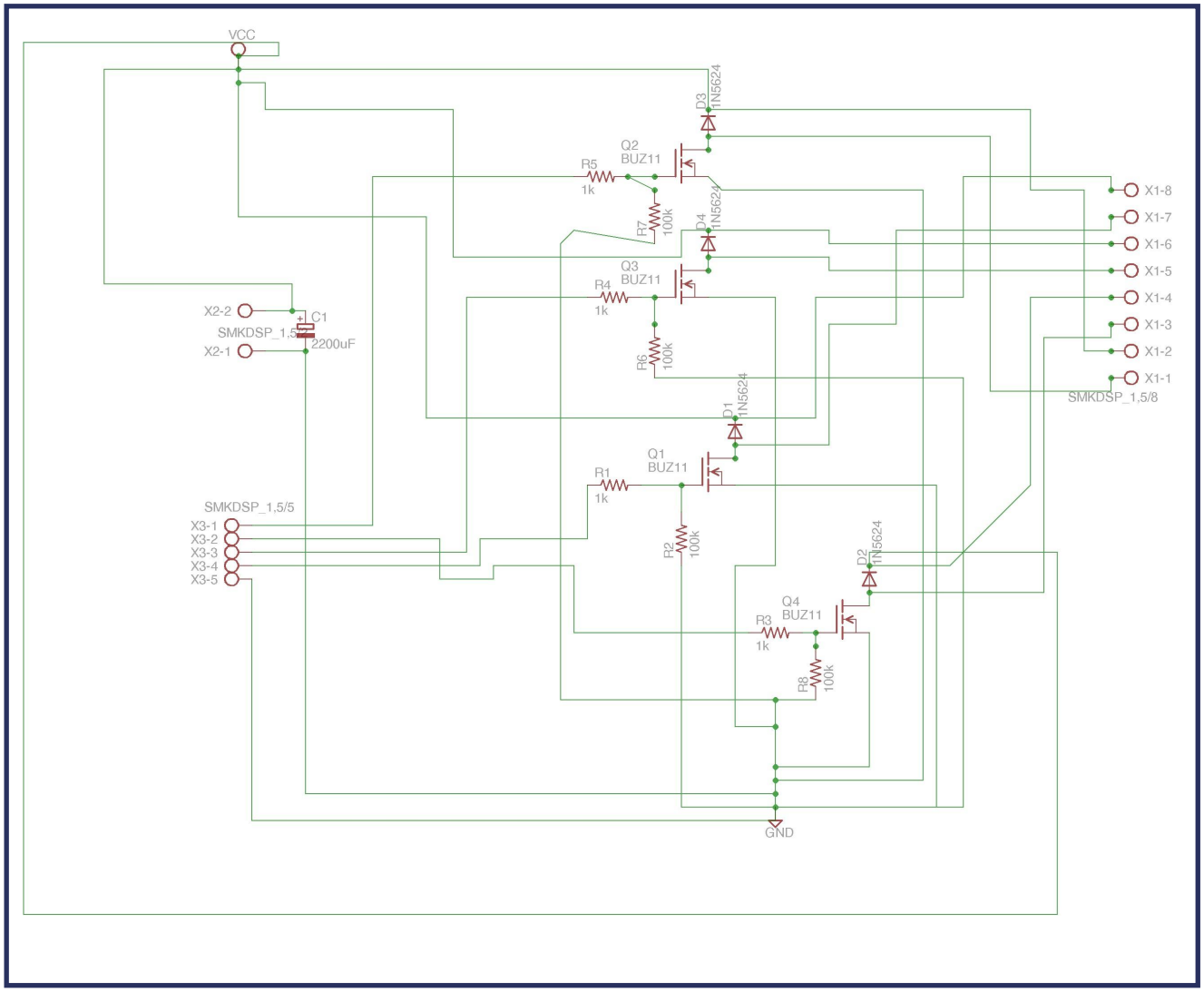
Tabla 10. Horas personal

Presupuesto final

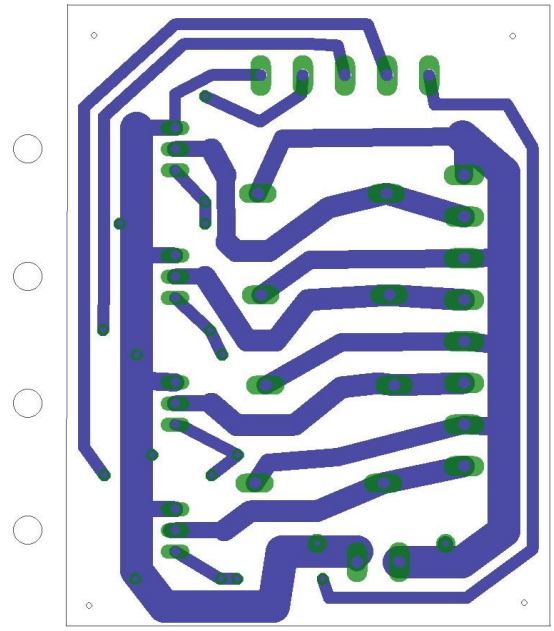
El presupuesto final es la suma de todos los apartados anteriores, el cual es de 14.609,71€.

Planos

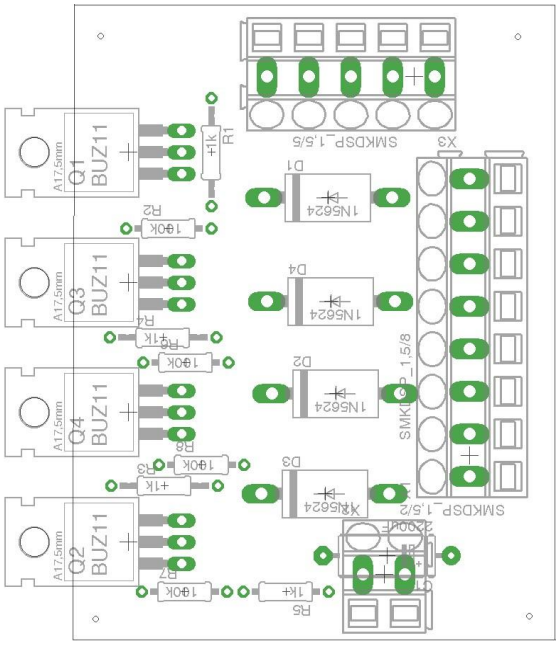
Índice de planos



Aitor Roig Alemany		Universidad Jaume I
2015	18 de septiembre	
Plano nº 1	Amplificador para bombas. Circuito esquemático	UNIVERSITAT JAUME·I



Aitor Roig Alemany		Universidad Jaume I
2015	18 de septiembre	
1:1	Amplificador para bombas. Cara de soldaduras	UNIVERSITAT JAUME·I
Plano nº 2		



Aitor Roig Alemany		Universidad Jaume I
2015	18 de septiembre	
1:1	Amplificador para bombas. Ubicación de componentes	UNIVERSITAT JAUME·I
Plano nº 3		