Escola Universitaria Politécnica

UNIVERSIDADE DA CORUÑA

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO DE FIN DE GRADO

TFG Nº: **770G01A176**

TÍTULO: **IMPLEMENTACIÓN DE UN REGULADOR PID GENÉRICO EMBEBIDO**

AUTOR: **ALEJANDRO FERNÁNDEZ ROMERO**

TUTOR: **ESTEBAN JOVE PÉREZ**
JOSÉ LUIS CASTELEIRO ROCA

FECHA: **SEPTIEMBRE DE 2019**

Fdo.: EL AUTOR

Fdo.: EL TUTOR

TÍTULO: **IMPLEMENTACIÓN DE UN REGULADOR PID GENÉRICO EMBEBIDO**

ÍNDICE

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2019

AUTOR: EL ALUMNO

Fdo.: ALEJANDRO FERNÁNDEZ ROMERO

Listado de figuras

Listado de tablas

Listado de códigos de programación

TÍTULO: **IMPLEMENTACIÓN DE UN REGULADOR PID GENÉRICO EMBEBIDO**

MEMORIA

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2019

AUTOR: EL ALUMNO

Fdo.: ALEJANDRO FERNÁNDEZ ROMERO

Índice del documento MEMORIA

1 OBJETO

Este Trabajo Final de Grado aborda la implementación de un regulador PID dentro de una de las plataformas de bajo coste basadas en microcontroladores integrados (en este caso, Arduino). El trabajo consistirá en el desarrollo de un regulador PID mediante MATLAB Simulink, con la función de controlar un sistema a través de una de las plataformas mencionadas. Una vez desarrollado el regulador, se programará internamente en la plataforma para su uso independiente. Además, se realizarán las guías necesarias para el soporte a futuros alumnos en el uso de este tipo de programación (Simulink-Arduino).

2 ALCANCE

- Estudio de la programación Simulink en plataformas hardware de bajo coste.
- Estudio de las limitaciones del uso de este tipo de plataformas: comprobación de los tiempos de respuesta, de los recursos u otros posibles inconvenientes.
- Implementación de un regulador PID embebido genérico en Simulink.
- Desarrollo de una guía de uso de las plataformas de hardware de bajo coste en Simulink.

3 ANTECEDENTES

Es bien sabido que el uso de reguladores es de gran importancia en el mundo moderno actual, tanto cotidiano, como industrial. Con este Trabajo Fin de Grado (TFG), estudiamos y posteriormente mostramos, que su utilización no tiene que ir siempre vinculada con el uso de material con un coste considerable, como un ordenador personal, o a software especializado de difícil programación.

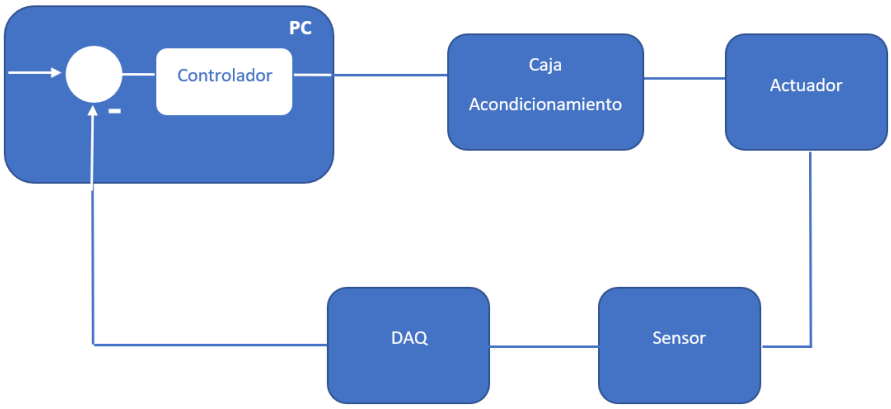


Figura 3.1 – Esquema actual de la planta

Actualmente, en el laboratorio de optimización y control existen dos plantas, las cuales siguen el esquema mostrado en la figura 3.1. Tal y como se puede observar, el ordenador posee una posición de gran importancia dentro del montaje, siendo este el que ejecuta el lazo de control. Además, cuentan con elementos como son las tarjetas de adquisición de datos, las cuales realizan únicamente dicha función.

Es por ello que vemos interesante la implantación de un diseño físico que simplifique el actual, tendiendo a la forma de un controlador embebido. Aprovechando la posibilidad de ejecutar programas de control directamente sobre la placa Arduino, eliminamos la presencia del ordenador. Por lo tanto, el microcontrolador se convertirá en el centro del diseño, adquiriendo datos, ejecutando el programa de control y comunicándose con el actuador.

4 NORMAS Y REFERENCIAS

4.1. Disposiciones legales y normas aplicadas

Este trabajo ha sido desarrollado siguiendo la normativa de TFG.

4.2. Software empleado

The MathWorks. MATLAB-Simulink [software]. Versión R2017b. 21 Septiembre 2017 [consulta: 5 julio 2019]. Disponible en: https://es.mathworks.com/campaigns/products

ARDUINO. Arduino [software]. Versión 1.8.9. 15 Marzo 2019 [consulta: 5 julio 2019]. Disponible en: http://arduino.cc/en/Main/Software

KiCad EDA. KiCad [software]. Versión 5.1.2. 5 Junio 2019 [consulta: 12 agosto 2019]. Disponible en: https://kicad-downloads.s3.cern.ch/index.html?prefix=windows/stable/

4.3. Referencias

[1] *Controlador PID*. Wiki en Internet. Wikimedia Foundation, Inc. 2001. [Consulta 22 abril 2019]. Disponible en: https://es.wikipedia.org/wiki/Controlador_PID

[2] *Modulación por ancho de pulsos*. Wiki en Internet. Wikimedia Foundation, Inc. 2001. [Consulta 22 abril 2019]. Disponible en: https://es.wikipedia.org/wiki/Modulacion_por_ancho_de_pulsos

[3] *Static Accuracy Tests of the Arduino Internal ADC*. [Consulta 18 abril 2019]. Disponible en: https://forum.arduino.cc/index.php?topic=120004.0

[4] *Memoria Flash, SRAM y EEPROM*. [Consulta 2 marzo 2019]. Disponible en: https://aprendiendoarduino.wordpress.com/2017/06/21/memoria-flash-sram-y-eeprom-3/

[5] *Comunicaciones con Arduino*. [Consulta 18 abril 2019]. Disponible en: https://aprendiendoarduino.wordpress.com/2014/11/18/tema-6-comunicaciones-con-arduino-4/

[6] *EL BUS I2C EN ARDUINO*. [Consulta 18 abril 2019]. Disponible en: https://www.luisllamas.es/arduino-i2c/

[7] *Método de Ziegler-Nichols*. [Consulta 2 marzo 2019]. Disponible en: https://www.picuino.com/es/arduprog/control-ziegler-nichols.html

[8] *CURSO CONTROL PID PRACTICO*. [Consulta 10 julio 2019]. Disponible en: https://instrumentacionycontrol.net/resumen-p-i-d-lo-justo-y-necesario-que-debes-saber-y-que-nunca-entendiste/

[9] *Controladores PID. Ajuste empírico*. [Consulta 5 mayo 2019]. Disponible en: http://www.dia.uned.es/~fmorilla/MaterialDidactico/ajuste_empirico.pdf

[10] *Ajuste empírico de reguladores PID*. [Consulta 5 mayo 2019]. Material proporcionado al alumno. Autor: José Luis Calvo Rolle.

[11] *ArduinoPinCurrent*. [Consulta 18 abril 2019]. Disponible en: https://arduinoinfo.mywikis.net/wiki/ArduinoPinCurrent

[12] *Installation of some support packages from inside MATLAB fails with "Install Error" message*. [Consulta 18 abril 2019]. Disponible en: https://es.mathworks.com/support/bugreports/150766

[13] *MathWorks Account*. [Consulta 18 abril 2019]. Disponible en: https://www.mathworks.com/mwaccount/register

[14] *PWM generation using arduino mega 2560 through simulink* . [Consulta 22 abril 2019]. Disponible en: https://es.mathworks.com/MATLABcentral/answers/152374-pwm-gneration-using-arduino-mega-2560-through-simulink

[15] *DISIPADORES TÉRMICOS PARA DISPOSITIVOS ELECTRÓNICOS*. [Consulta 3 julio 2019]. Disponible en: http://ieb-srv1.upc.es/gieb/tecniques/pdf/disipadores_termicos.pdf

[16] *Cálculo del disipador de un LM317 en funcionamiento regulable*. [Consulta 3 julio 2019]. Disponible en: https://www.forosdeelectronica.com/threads/calculo-del-disipador-de-un-lm317-en-funcionamiento-regulable.46298/

[17] *PCB Trace Width Calculator*. [Consulta 10 julio 2019]. Disponible en: http://circuitcalculator.com/wordpress/2006/01/31/pcb-trace-width-calculator/

[18] *Getting Started with Simulink in MATLAB: Designing a Model*. [Consulta 11 julio 2019]. Disponible en: https://circuitdigest.com/tutorial/getting-started-with-simulink-in-MATLAB

[19] *Detect and Fix Task Overruns on Arduino Hardware*. [Consulta 25 abril 2019]. Disponible en: https://es.mathworks.com/help/supportpkg/arduino/ug/detect-and-fix-task-overruns-on-arduino-hardware.html

[20] *El condensador como filtro*. [Consulta 21 julio 2019]. Disponible en: https://apuntesdeelectronica.files.wordpress.com/2011/10/4-el-condensador-como-filtro.pdf

[21] *Sense Room Temperature and Display in LCD Using Simulink and Arduino UNO*. [Consulta 2 marzo 2019]. Disponible en: https://www.instructables.com/id/Sense-Room-Temperature-and-Display-in-LCD-using-Si/

[22] *PID Compensators Implementation Practical Aspects for a Car Control Position*. [Consulta 22 agosto 2019]. Disponible en: https://dialnet.unirioja.es/descarga/articulo/5038423.pdf

[23] *S-Function Builder Dialog Box*. [Consulta 18 abril 2019]. Disponible en: https://es.mathworks.com/help/simulink/sfg/s-function-builder-dialog-box.html

4.4. Hojas de características

[24] *LM317 3-Terminal Adjustable Regulator*. [Consulta 22 mayo 2019]. Disponible en: http://www.ti.com/lit/ds/slvs044x/slvs044x.pdf

[25] *LM340, LM340A and LM7805 Family Wide VIN 1.5-A Fixed Voltage Regulators*. [Consulta 22 mayo 2019]. Disponible en: http://www.ti.com/lit/ds/symlink/lm7800.pdf

[26] *Disipadores térmicos en semiconductores TO-220-3*. [Consulta 3 julio 2019]. Disponible en: https://tinyurl.com/y53hg94f

[27] *Transformador 12V 16 VA*. [Consulta 3 julio 2019]. Disponible en: https://tinyurl.com/y4g2saub

[28] *LMx58-N Low-Power, Dual-Operational Amplifiers*. [Consulta 3 julio 2019]. Disponible en: http://www.ti.com/lit/ds/symlink/lm358-n.pdf

4.5. Enlaces de compra de los componentes

[29] *Arduino Mega 2560*. [Consulta 16 julio 2019]. Disponible en: https://store.arduino.cc/mega-2560-r3

[30] *Teclado Matricial 4x4*. [Consulta 16 julio 2019]. Disponible en: https://tinyurl.com/y55kxv8k

[31] *Pantalla LCD 16x2*. [Consulta 16 julio 2019]. Disponible en: https://tinyurl.com/y34u6vuh

[32] *Kit de iniciación*. [Consulta 16 julio 2019]. Disponible en: https://tinyurl.com/y3ymlwuc

[33] *Potenciómetro 10 K*. [Consulta 16 julio 2019]. Disponible en: https://tinyurl.com/y6t6rxf6

[34] *Transformador 12 V*. [Consulta 16 julio 2019]. Disponible en: https://tinyurl.com/y5sf63fh

[35] *Disipador de los reguladores de tensión*. [Consulta 16 julio 2019]. Disponible en: https://tinyurl.com/y2s5cq97

[36] *Interruptor PCB*. [Consulta 16 julio 2019]. Disponible en: https://tinyurl.com/y6eyjf8q

[37] *Puente de diodos*. [Consulta 16 julio 2019]. Disponible en: https://tinyurl.com/y3zljp2y

[38] *Regulador de tensión LM7812*. [Consulta 16 julio 2019]. Disponible en: https://tinyurl.com/yydttq6h

[39] *Regulador de tensión LM317*. [Consulta 16 julio 2019]. Disponible en: LM317:https://tinyurl.com/yxpuj9cq

5 DEFINICIONES Y ABREVIATURAS

- **Controlador PID:** Es un sistema realimentado con gran uso industrial que consta en la acción conjunta de una componente proporcional, una integral y una derivativa.
  El valor proporcional depende directamente del error en ese preciso instante. El integral de los errores acumulados pasados y el derivativo predecirá los futuros. La suma de estas tres acciones es usada para ajustar al proceso por medio de un actuador [1].
- **Kc:** (Ganancia proporcional crítica), ganancia de un controlador únicamente proporcional, que provoca que el sistema entre en un estado de oscilación permanente [7]. Su conocimiento, permite calcular la constante proporcional del regulador, Kp.
- **Tc:** Periodo de oscilación sostenida de un sistema que se consigue por medio de la ganancia crítica [7].
- **Td:** Constante temporal que hace referencia a la constante del tiempo derivativo. Matemáticamente es el cociente del desplazamiento de la salida y la velocidad de entrada [8]. Para conseguir la constante derivativa Kd, bastará con multiplicar el tiempo Td por la contante Kp del sistema.
- **Ti:** Constante temporal que hace referencia al cociente del error de entrada entre la velocidad de salida [8]. Para conseguir la constante integral Ki, bastará con dividir la constante del sistema Kp entre Ti.
- **PWM:** Forma de modulación de una señal que se efectúa por medio de la graduación del ancho del pulso de una señal periódica de una determinada frecuencia. En este trabajo la usaremos para poder conseguir un valor medio de tensión en la señal inferior al valor de inicio [2].

6 REQUISITOS DE DISEÑO

- Se busca controlar una planta del laboratorio por medio de un microcontrolador, actuando este de forma totalmente autónoma.
    - Adquiriendo datos.
    - Calculando sus parámetros automáticamente sin contribución necesaria por parte del operador.
    - Regulando de forma independiente.
    - Siendo compatible con diferentes sistemas sin ser necesaria su reprogramación.
    - Se busca la compactación del concepto, simplificando el montaje tradicional y proponiendo el diseño íntegro de un regulador independiente.
- Interesa que sea programado con un entorno visual por medio de bloques.
- Gozará de libertad de movimiento dentro del laboratorio.
    - Su funcionamiento no dependerá de ningún ordenador.
    - Poseerá alimentación directamente de la toma de corriente.
- Gozará de un rango de entradas y salidas configurable.
- Si la placa se conecta a un ordenador personal, se podrá monitorizar por medio de este la regulación, graficando, por ejemplo, el parámetro a controlar o modificar variables del proceso.
- Se adjuntarán una serie de programas diseñados, partiendo de soluciones más básicas, que irán ascendiendo en funcionalidades y dificultad.
- Se incluirán ejemplos que, aprovechando otras posibles capacidades del microcontrolador, pueden resultar interesantes hacia un usuario final.
- Se elaborará una guía que servirá como apoyo a alumnos que quieran profundizar en este área de trabajo en concreto.
- Irá orientado a una hipotética formación de alumnado, hecho por el cual se planteará el diseño íntegro del montaje, rechazando soluciones comerciales.

7 ANÁLISIS DE LAS SOLUCIONES

En el presente apartado se analizan las posibles alternativas contempladas a la hora de realizar el trabajo.

7.1. Elección del microcontrolador

Existiendo varias alternativas, para decantarse por una opción, habrá que evaluar, probar y comparar las posibles soluciones que más se adapten a nuestros requerimientos.
Dentro de estas opciones, seleccionamos como candidatos a los mini-ordenadores Raspberry Pi Model 3 y BeagleBone Black y al microcontrolador Arduino.

7.1.1. Raspberry Pi Model 3

El mini-ordenador de la Rapsberry Pi Foundation posee puntos positivos como son contar con una gran comunidad de soporte y un procesador de mayor capacidad que el integrado dentro del microcontrolador Arduino. Este puede ser utilizado para actividades de mayor demanda de potencia, como puede ser el proceso de imagen, vídeo o sonido. Pero son sus desventajas las que descartan su uso, ya que la imposibilidad de utilizarlo para adquisición de datos, al no contar con un convertidor A/D integrado incumple nuestras condiciones de diseño iniciales y la posible solución a este problema no sería ni inmediata, ni simple.

En una primera etapa del trabajo, fue planteada la posibilidad de combinar la Raspberry Pi con el microcontrolador Arduino, para que este se dedicase a las tareas de adquisición de datos y así supliese los déficits de la Model 3. Tras su debido análisis, se llegó a la conclusión de que esta solución aportaba ventajas que para nuestra aplicación en concreto no tendrían un efecto apreciable, mientras que aumentaría la complejidad y coste de nuestro montaje, existiendo dos dispositivos que programar, acondicionar, etc.

7.1.2. BeagleBone Black

La segunda alternativa, la BeagleBone, palía las desventajas de la primera. Incluyendo un convertidor A/D de 12 bits, facilitaría su selección. El problema se halla en la propia filosofía sobre la que se diseñó este dispositivo. Como mini-ordenador que es, arranca por defecto sobre una distribución GNU/Linux, en este caso Debian. En cada arranque, el posible programa de regulación tendría que ser lanzado por el usuario, siendo siempre necesario su input, por lo que incumpliría el objetivo de ser completamente autónomo. Además, el proceso de puesta a punto, instalación de un sistema operativo y solucionar los problemas de compatibilidad con el

programa resultan demasiado arduos para un usuario poco experimentado que quiera introducirse a este mundo, objetivo principal sobre el que va dirigido este trabajo. Su coste elevado, su menor utilización en educación y falta de soporte también influyen en la decisión.

7.1.3. Arduino

Por lo tanto, nos quedaremos con Arduino, porque, además de no poseer las desventajas mencionadas anteriormente de sus competidores, está más extendido a nivel educativo y es un sistema el cual se podrá integrar en todo tipo de actividades y entornos gracias a su gran versatilidad. En el esquema 7.2, se sintetizan los puntos positivos que condicionaron su elección. Además, goza de un precio realmente competitivo al estar licenciado a nivel de hardware y software bajo el código abierto. Esto posibilita el poder comprar modelos no distribuidos por la propia marca sin perder la compatibilidad con todos sus entornos y accesorios.

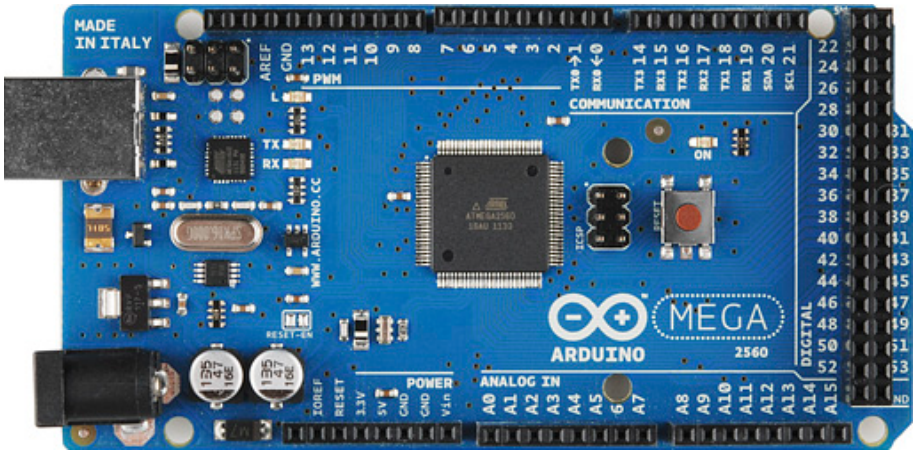


Figura 7.1 – Arduino Mega 2560

Sus accesorios se pueden adquirir de forma conjunta a través de kits de iniciación, facilitando tareas de rápido prototipado, no necesitando en la mayoría de casos ningún tipo de circuito de adaptación, hecho que puede beneficiar a alumnos con falta de material. Gracias a esto, con una simple protoboard, se podrá replicar de forma simple en casa todo el montaje. Además, todas sus librerías, están disponibles libremente para ser revisadas y modificadas en internet.

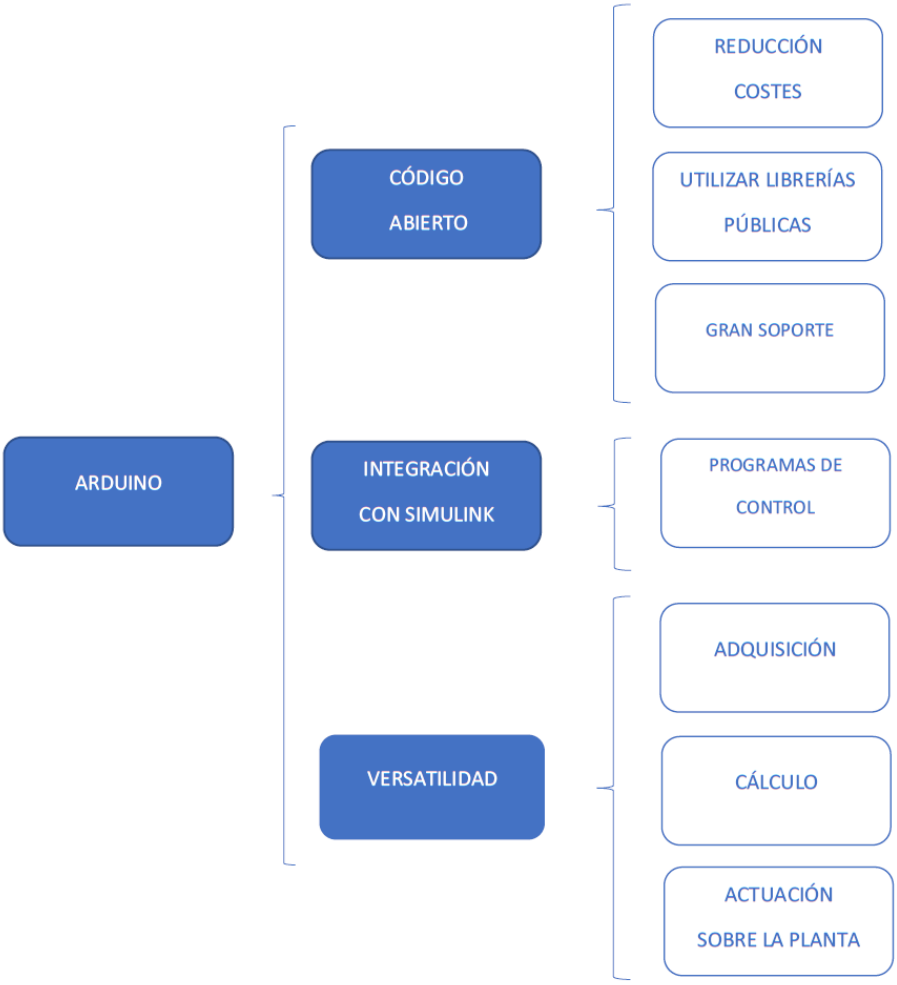


Figura 7.2 – Ventajas de la implementación a través de Arduino

7.1.3.1. Elección entre el Arduino UNO y Mega

En este trabajo, el microcontrolador elegido fue el modelo Arduino MEGA 2560 Revisión 3 7.1. Se decidió entre el modelo UNO y este. Ambos eran los modelos de los que se disponía y son los más conocidos dentro de la marca. Las razones que decantaron su elección, fueron sus memorias de mayor capacidad con respecto al UNO, tanto de programa, (Flash y EEPROM), como de datos [4], lo que será útil a la hora de cargar programas más pesados en la placa. Además se aprovechará su mayor número de puertos de entrada y salida, que serán útiles para poder alimentar numerosos accesorios, conectar pantallas, interruptores, leds, etc. Cabe destacar, que pese a elegir esta gama en cuestión, su coste se sigue manteniendo bastante bajo, sobre 30 euros, en el caso de adquirir el producto oficial. Para la mayoría de casos, sería posible el uso de un UNO y en ambos casos su integración con gran variedad de programas es total.

7.2. Elección del entorno de programación

Como ya se ha comentado anteriormente, el enfoque de este trabajo ha sido distanciarse de los lenguajes de programación tradicionales sintácticos debido a la complejidad que suponen a usuarios poco experimentados el hecho de implementar modelos matemáticos de control. Debido a este motivo, cada vez es más común la aparición de nuevos lenguajes basados en la colocación de bloques. La intención de todo esto es introducir el mundo de la programación a un público como el de la educación secundaria y bachillerato, el cual, en la actualidad, está distanciado de esta.

7.2.1. Programas visuales básicos

El grupo de lenguajes visuales, basados en la agrupación de bloques, nacen como alternativa a los lenguajes tradicionales. Tienen gran recibo dentro de las etapas iniciales de la formación. Esto se debe a que son lenguajes de alto nivel con los cuales el usuario tiene una forma de interacción mucho más simplificada, asemejándose a la estructuración de un simple flujograma, que da como resultado secuencias simples ejecutables en bucle. Existen varios ejemplos, que serían: Scratch for Arduino, ArduBlock, BlockyDuino, MBLOCK y MiniBloq. Estos programas sí reúnen los requisitos de facilidad e intuitividad de uso. Pero debido a que no fueron desarrollados con la idea de implementar lazos de control, poseen una falta de características que resultarían necesarias, por lo que serán rechazados.

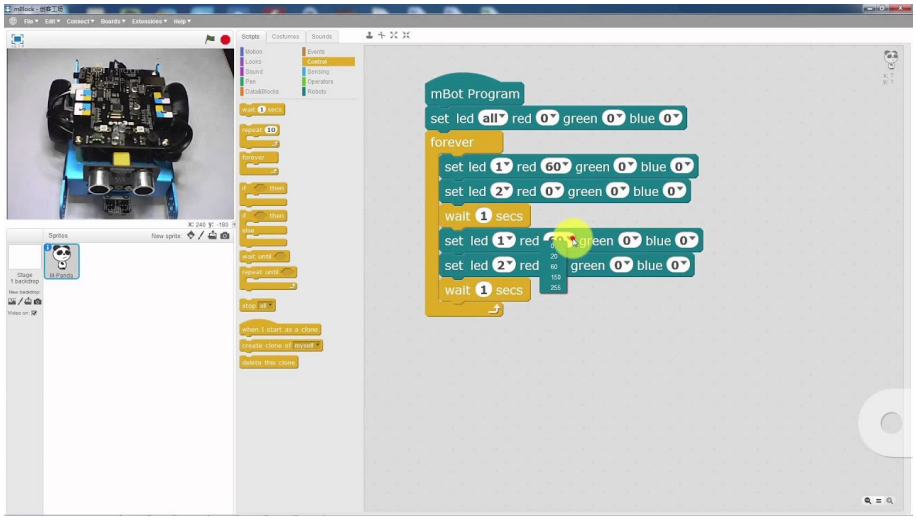


Figura 7.3 – Entorno de programación MBloq

7.2.2. Programas visuales de control

7.2.2.1. Xcos

Programas como MATLAB-Simulink y su alternativa código abierto, Xcos, sí fueron desarrollados con la idea de poder ejecutar lazos de control. Estos programas están basados en bloques, los cuales, en su interior poseen un código a ejecutar. Existe gran cantidad de ellos y con una simple selección, es posible realizar un lazo de control complejo en cuestión de pocos minutos. Los dos poseen una gran similitud de funcionamiento e incluso de apariencia, pues el objetivo de Xcos es el de traer al uso del software libre a actuales usuarios de Simulink, sin que sea necesaria una gran adaptación de por medio. Pero debido a su bajo uso, carece de una gran cantidad de soporte, por lo que se descartaría su uso en este trabajo.

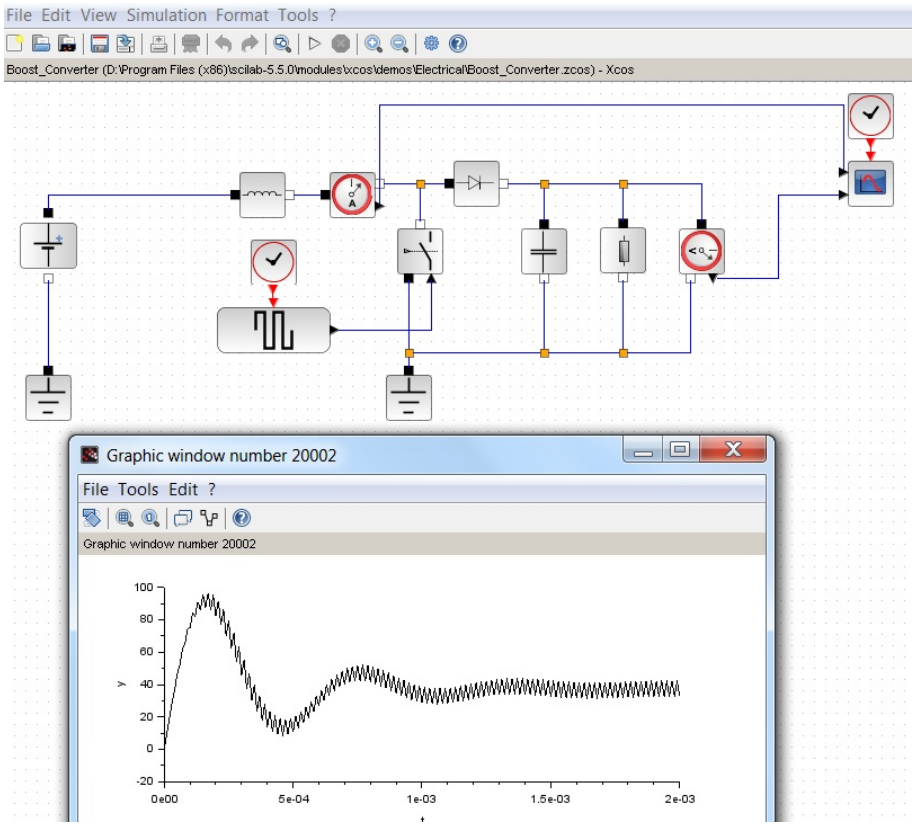


Figura 7.4 – Entorno Xcos

7.2.3. Simulink

El hecho de gozar con un gran soporte con el cual solucionar cualquier tipo de inconveniente es un hecho a tener en cuenta, sobre todo en el uso educativo. La integración MATLAB-Arduino es una opción con gran respaldo por parte de la comunidad, únicamente superada por su IDE por defecto.

Junto a este motivo, cabe destacar que Simulink está ya presente en el centro, por lo que nos decantaremos por este último como solución. Este nos proporcionará una sólida base sobre la que desarrollar nuestras aplicaciones.

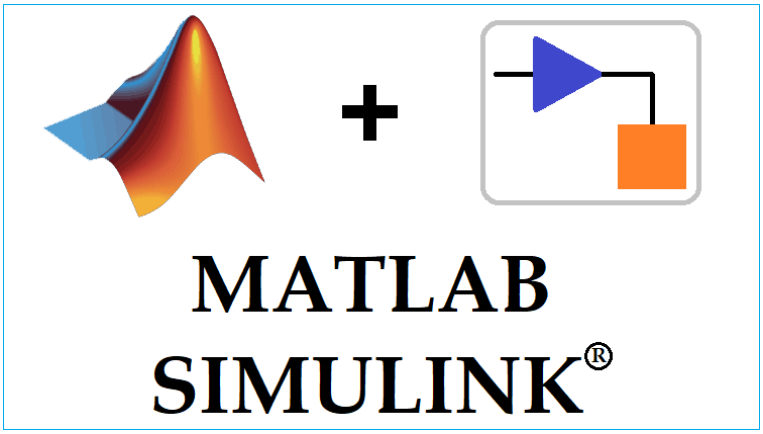


Figura 7.5 – MATLAB y Simulink [18]

En el interior de los bloques, habrá implementado un fragmento de código que desempeñará una determinada función. En Simulink se pueden diseñar funciones programadas por parte del propio usuario. Gracias a esta utilidad, se podrán compilar incluso funciones y librerías en C, importadas directamente desde la comunidad código abierto para poder trasladar a Simulink accesorios del Arduino que solo poseen soporte bajo el IDE por defecto, obteniendo así todos sus beneficios. De esta forma fueron adaptados accesorios, que con otros programas sería imposible. Este apartado se desarrolla en el siguiente anexo 12.7.

Otra gran característica es la posibilidad de ejecutar el programa en la placa y a su vez observar la evolución de este en tiempo real en la pantalla del ordenador, el llamado modo de ejecución externa, como se muestra en la siguiente captura 7.6. Será descrito con más detenimiento en el siguiente anexo 12.5, pero incluso posee la posibilidad de representar gráficas, que serán útiles a la hora de presentar resultados. Esta es otra ventaja sobre su propio IDE.

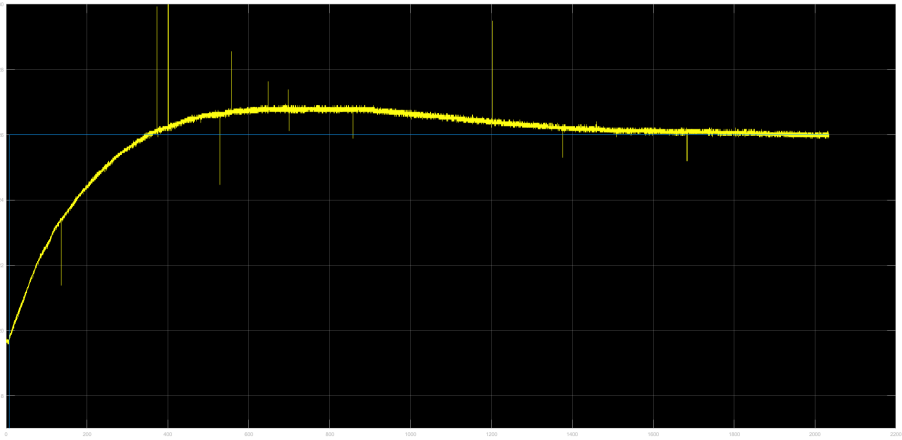


Figura 7.6 – Posibilidad de graficar en tiempo real

7.3. Elección de la planta

En la actualidad, en el laboratorio de control, existen dos tipos de plantas disponibles sobre las que implementar el regulador, la de nivel y el horno. Se debe de elegir cuál es la planta más conveniente para trabajar en la fase de prototipado, sobre la que se efectuarán todas las pruebas incluidas en este trabajo.
Ambas poseen el siguiente esquema:

- Una variable a regular.
  - Nivel de agua.
  - Temperatura.
- Un sensor por el cual se cuantifica en forma de tensión la variable a medir.
  - Sensor de ultrasonidos, (mide la altura del agua).
  - Sensor de temperatura.
- Un actuador que pueda modificar la variable hasta aproximarla a la consigna.
  - Bombilla incandescente.
  - Bomba de agua.
- Tarjeta de adquisición de datos.
- PC ejecutando el algoritmo de control.

7.3.1. Planta de nivel

La planta de nivel, se basa en el control del nivel de agua en un depósito, a través de un sensor de ultrasonidos que mide dicho nivel y una bomba de agua controlada por un actuador accionado por el dispositivo que ejecuta el lazo de control. Existen dos montajes en el laboratorio, uno que usa una tarjeta de adquisición de datos de la marca National Instruments, y otro, que usa un Arduino.

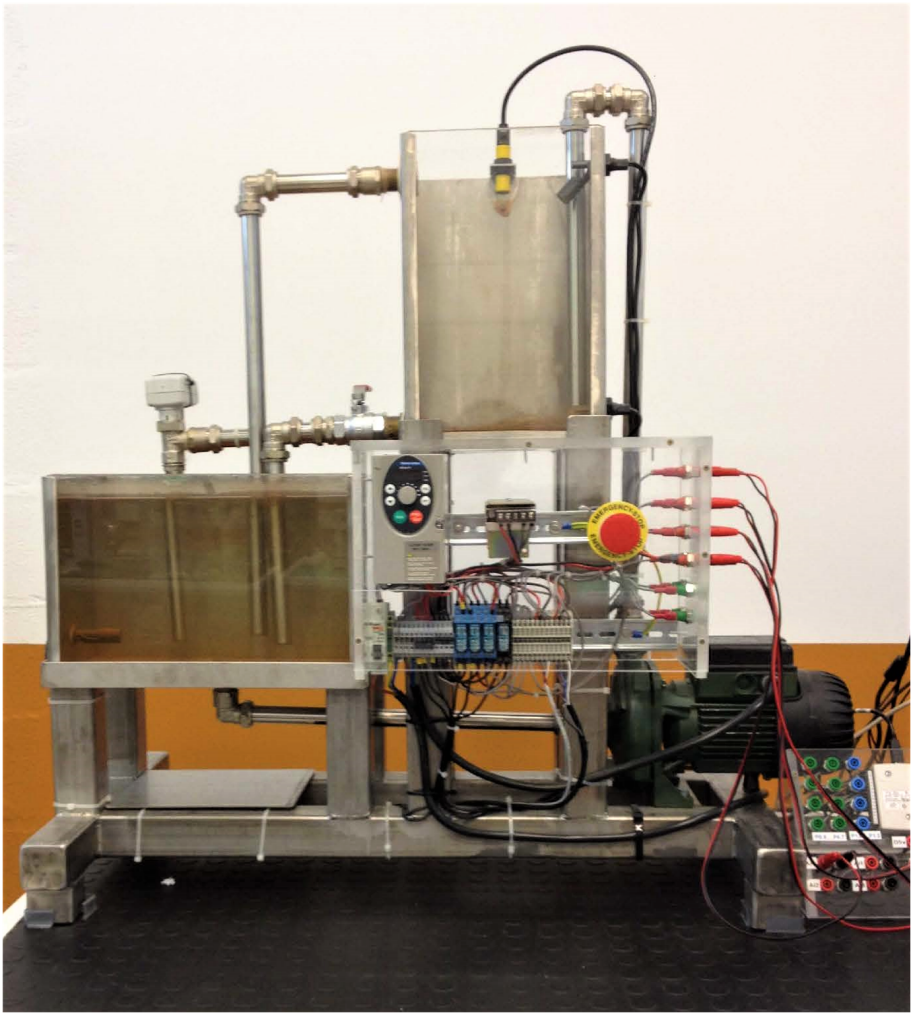


Figura 7.7 – Planta de nivel

7.3.2. Planta del horno

El funcionamiento del horno de metacrilato se basa en la baja eficiencia de iluminación de una bombilla de incandescencia para calentar el entorno cerrado por dicho material. Por medio del conocido sensor de temperatura LM35 de sensibilidad 10 mV/ºC, una caja de acondicionamiento de tensiones y un ajuste dentro del propio programa se podrá obtener el valor en grados centígrados.
Dicha caja también es la encargada de adaptar las tensiones a la salida de la tarjeta de adquisición de datos.

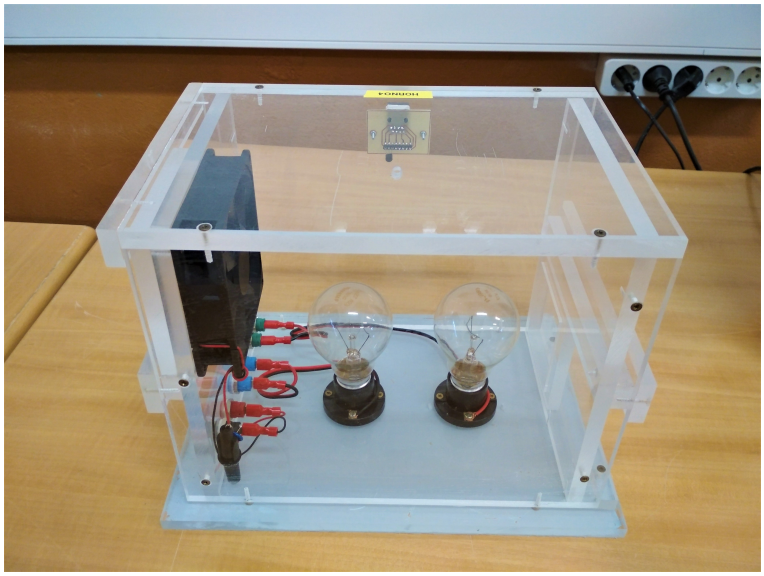


Figura 7.8 – Planta del horno

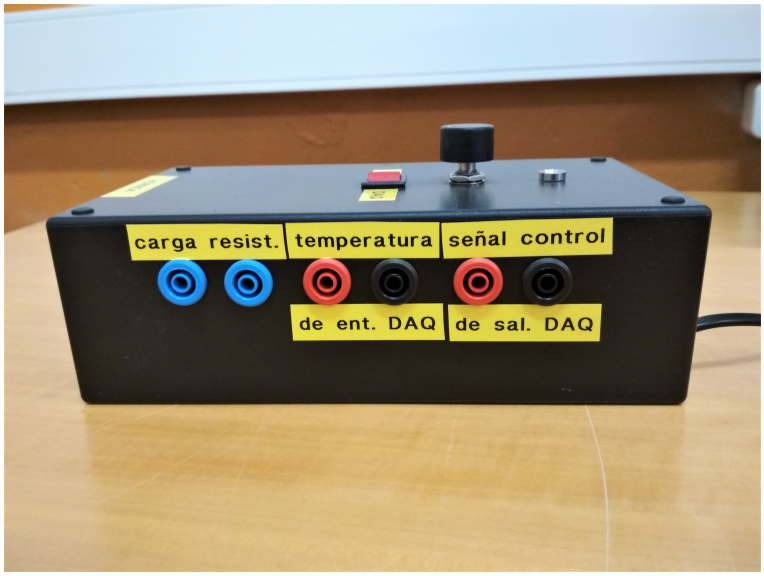


Figura 7.9 – Caja de acondicionamiento

7.3.3. Comparación del rango de tensiones de cada una de las plantas

En la siguiente tabla se adjunta las características eléctricas de las plantas en cuestión 7.1:

Planta	Rango de la señal del sensor (V)	Rango necesario en la señal de control (V)	Rango de E/S Arduino (V)
Planta Horno	0-10	0-5	0-5
Planta Nivel	0-10	0-10	0-5

Tabla 7.1 – Comparación de las plantas

Como se puede ver, la señal de control ha de ser amplificada si se quiere conectar el Arduino a la planta de nivel. En el caso del horno no es necesario, pues la salida de la DAQ National Instruments posee un rango de 0 a 5V, el mismo con el cual la sustituiremos. En la fase de prototipado, como aún no se disponía de una fuente con la tensión necesaria para poder alimentar a un posible circuito de acondicionamiento, se decidió utilizar la planta del horno. Sobre esta se efectuarán todos los siguientes análisis y pruebas. Además, complementa la elección de dicha planta el hecho de que si al regulador se le cargan valores incorrectos, (hecho que puede ocurrir debido al carácter experimental del propio trabajo, con el cual se invita al usuario a probar diversas soluciones), puede desencadenar en un hipotético desborde de agua del depósito, o en daños sobre la bomba de agua.
El montaje final sí estará adaptado para poder actuar sobre ambas.

7.3.4. Conclusión

En resumen, se elegirá la planta del horno y solo se necesitará adaptar su tensión de la lectura del sensor. Y gracias a la uso del Arduino, se sustituye el uso de la tarjeta de adquisición de datos, interactuando este directamente sobre el actuador de la planta. Se ha hecho un diagrama que ilustra el montaje descrito. Como se puede ver, el PC y la tarjeta de adquisición de datos son reemplazados por el Arduino, efectuando así este la ejecución de código y la obtención de datos.

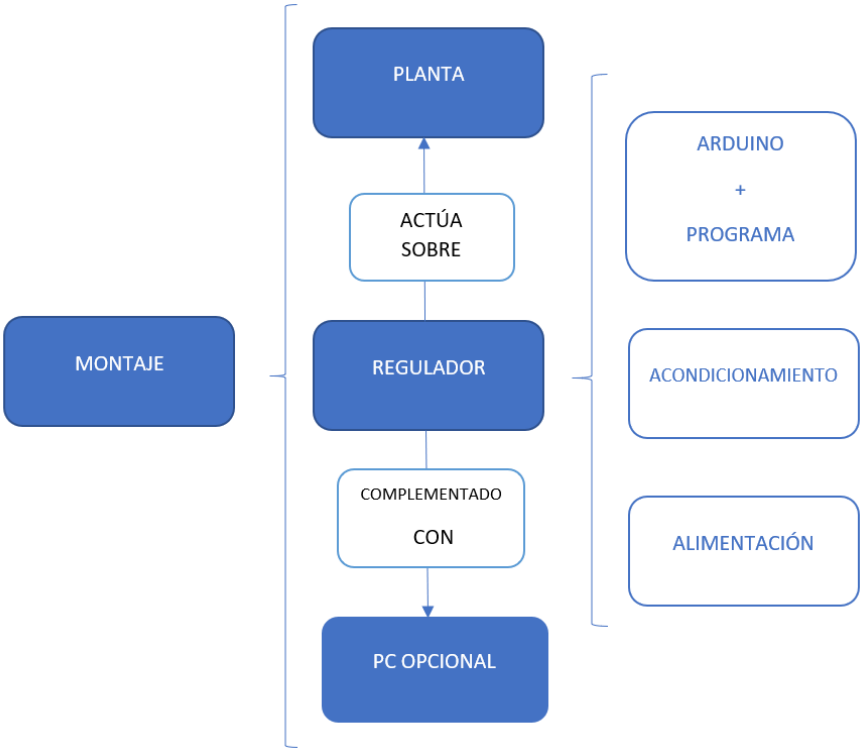


Figura 7.10 – Planteamiento general del trabajo

7.4. Elección del método de ajuste

Dado que se pretende controlar el sistema, se ha de elegir un método de ajuste por medio del cual calcular los parámetros de la regulación. Se podrá realizar un análisis en cadena abierta, o bien, en cadena cerrada, teniendo en cuenta datos en realimentación.

7.4.1. Métodos en cadena abierta

El primer método, consiste en fijar un valor consigna, una entrada escalón por ejemplo, el cual escribirá sobre el actuador, en este caso, la potencia de encendido de la bombilla. Esto provocará una reacción sobre la variable medible, que en este montaje es la temperatura, la cual llegará hasta un valor en régimen permanente.

Apoyándose en el ejemplo proporcionado por la siguiente fuente [9], se puede observar que para obtener los puntos de interés, con sus respectivos tiempos, se debe de realizar un análisis a posteriori, pues es imposible saber cual será el valor correspondiente al 28.3 % o al 63.2 % del estacionario antes de llegar a dicho valor estable, o predecir siquiera cuál será el mencionado valor estacionario.

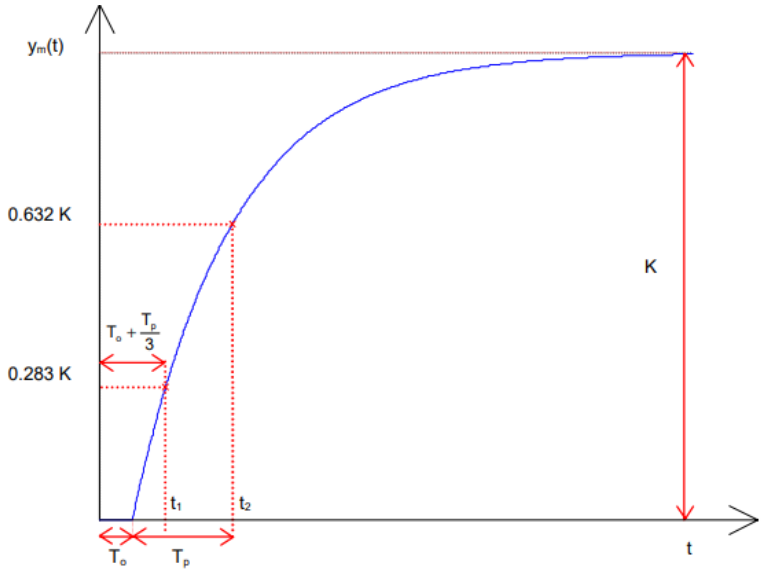


Figura 7.11 – Ejemplo de aplicación de un análisis en cadena abierta

Independientemente del método elegido, para obtener los parámetros que se necesitan habrá que realizar un análisis gráfico de la respuesta del sistema que se ha mostrado por pantalla. Por lo que para este método, se necesita una supervisión por parte del usuario, ya que es él el que procesa los datos y además, se necesita un ordenador para monitorizar en tiempo real la evolución del proceso.

Por lo tanto, concluimos que este método, aunque interesante, será descartado. Buscamos un método con el cual éstas constantes puedan ser calculadas por el propio sistema, de forma más fácil y de forma autónoma.

7.4.2. Métodos en cadena cerrada

Ahora nos centraremos en los análisis en cadena cerrada. Dentro de estos, nos quedaremos con el método de relay-feedback. En este método, dependiendo de una ventana de histéresis fijada con anterioridad en el código, y observando el valor de temperatura actual, el regulador es capaz de mantener al sistema oscilando permanentemente en torno a un valor de consigna fijado por el usuario, en función de como esté programada dicha variación. Esta forma de operar, lleva al sistema a un proceso de oscilación sostenida, lo cual permite obtener dos características propias del sistema, como son su período de oscilación característico, Tc; y su ganancia proporcional crítica Kc.

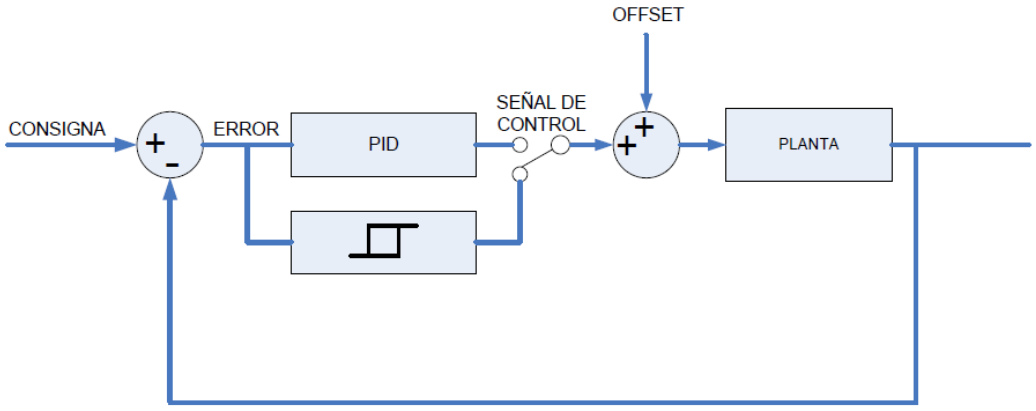


Figura 7.12 – Esquema de la implementación del sistema relé-feedback [10]

Este método, podrá ser calculado de forma matemática, ya que solo se necesitará obtener el valor máximo y mínimo de la oscilación y el período entre dichos picos. Para medir dicho período, se temporizará utilizando la potencia que se envía al actuador durante la histéresis. El cambio en el nivel de potencia (0 % a 100 %, 100 % a 0 %), dará las órdenes de iniciar/pausar la temporización. Para un sistema con una inercia prácticamente inexistente como el horno, el momento de conmutación entre el valor máximo y mínimo de potencia o viceversa, coincide con los valores máximos y mínimos de temperatura en la variable de salida.

A continuación, se obtendrán K, Td y Ti; a partir de las fórmulas de Ziegler-Nichols.

Por último, las constantes de la regulación Kp, Kd y Ki ya pueden ser calculadas.
Cabe destacar que estos parámetros son propios de cada planta, dependiendo de el estado físico único actual de esta y en el caso de los hornos, de la temperatura exterior, la cual tendrá

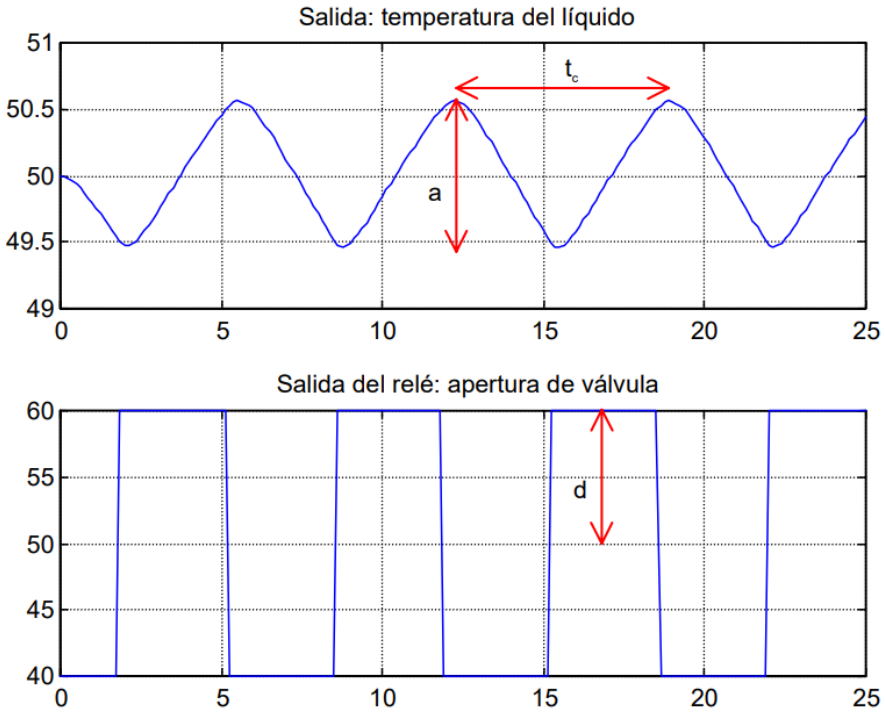


Figura 7.13 – Evolución de un sistema controlado por una histéresis [9]

un importante efecto en la amortiguación térmica de la planta. Por ello, la temperatura actual del laboratorio afecta en los valores calculados.

8 RESULTADOS FINALES

Como ya hemos dicho, el área de conocimiento que abarca la ingeniería de control es desconocida por la gran mayoría de población que no está en contacto con el ambiente industrial. Este desconocimiento se hace aún más presente si nos centramos en los estudiantes de la educación secundaria y del acceso a universidad.
Grandes iniciativas como pueden ser los cursos STEMbach, pueden acercar a este público los fundamentos de la automática, a través de actividades simples que involucren su participación activa en la regulación de pequeños sistemas, comprendiendo sus variables e involucrándose en el aprendizaje de su sencillo funcionamiento.

Con esta idea en mente, fueron diseñados los programas, en los cuales se tratan de plasmar diferentes conceptos de reguladores. En estos códigos se llevan a cabo las tareas de adquisición, cálculo y, por último, regulación.
Con el fin de que estas actividades puedan ser replicadas en los hogares, se ha dejado a disposición de los usuarios las fuentes consultadas, los planos de todas las placas elaboradas, una lista para la compra de materiales y una guía de iniciación a Simulink y Arduino.

8.1. Diseño final del montaje

Se ha recalcado anteriormente que las señales de tensión requerían un acondicionamiento. Como se han sustituido las tarjetas de adquisición de datos, se tendrán que adecuar las tensiones de la caja de adaptación al Arduino y viceversa.
Tratando de solucionar este hecho, se ha de diseñar una placa que acometa esta tarea.

Junto a esta, se ha diseñado una etapa de alimentación. Para cumplir los requisitos de diseño propuestos, vemos necesario diseñar una etapa personalizada para nuestro montaje. Esta tensión se suministraría a la placa por medio del conector de alimentación externa, tal y como se refleja en el anexo de cálculos 11.2.2.
Se trataría de una fuente de alimentación lineal con dos salidas de voltaje. Una para el circuito de acondicionamiento y otra para el propio Arduino.

El tercer diseño, constaría de una placa tipo entrenador donde irá acoplado el microcontrolador, dotando a este de teclado y pantalla, además de las conexiones a leds, interruptores y otros métodos de interacción con el usuario. La gran mayoría de los componentes usados se podrían adquirir fácilmente en los kits de iniciación de Arduino.

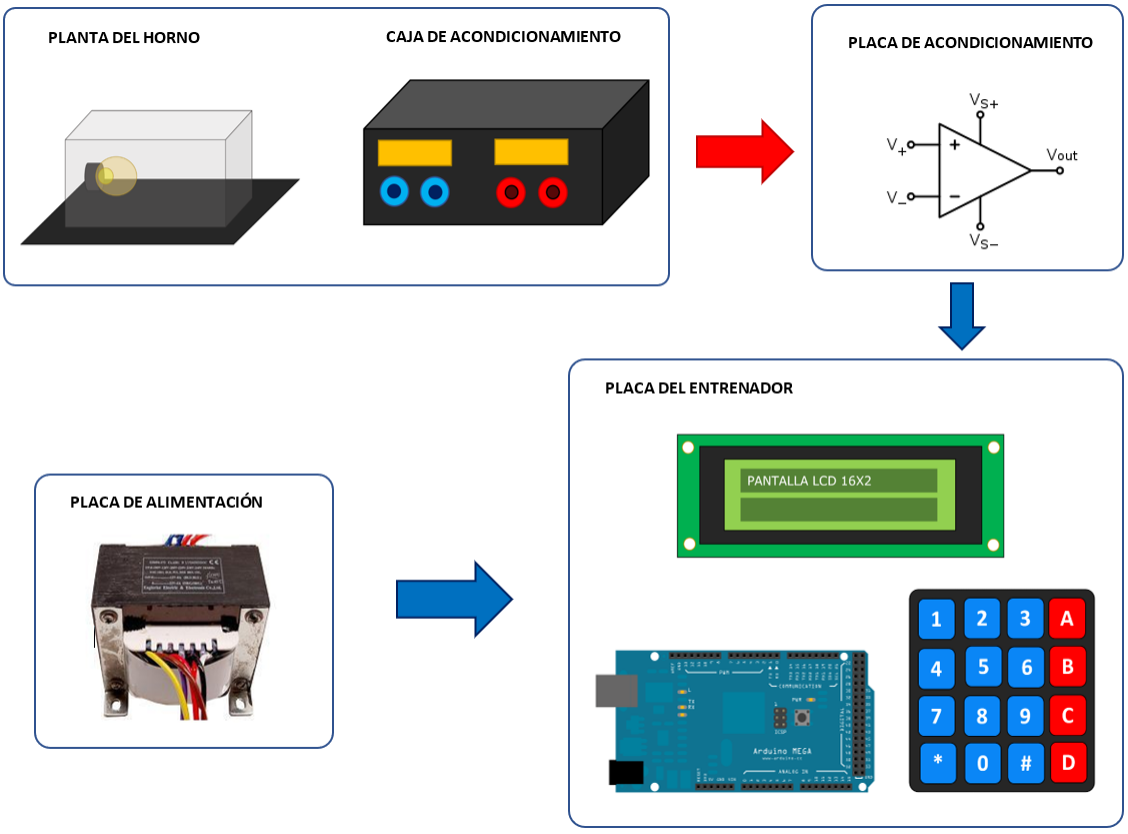


Figura 8.1 – Esquema general del montaje propuesto

8.1.1. Características de la placa del entrenador

- Permite la interacción del usuario con el regulador, mediante una interfaz visual, pantalla LCD y un teclado matricial.
- Poseen otras opciones, como leds de indicación o conmutadores.
- Esta placa fue ideada como un Shield para el Arduino. Este está encajado por medio de pines en la parte inferior.
- Ante un hipotético fallo de este, se puede retirar sin necesidad de resoldar.

8.1.2. Características de la placa de acondicionamiento

- Reducción de la tensión del sensor procedente de la caja de acondicionamiento, del rango por defecto (0-10 V), al compatible por el Arduino (0-5 V).
- Se ha añadido la posibilidad de adaptar en el futuro al montaje una FPGA, u otra placa como la BeagleBone dejando una salida con su tensión característica de entrada (0-3,3 V).
- La señal de control proveniente del Arduino, es amplificada hasta los 0 a 10 V.
- Incluye la posibilidad de ser usada sin amplificación.

8.1.3. Características de la placa de alimentación

- Proporcionará tensión al resto de etapas.
- Partiendo de 230 V de tensión, por medio de un transformador y posterior rectificación, filtrado y acondicionamiento se consiguen valores más adecuados de tensión continua.
- Por medio de un regulador de tensión fija, se consiguen 12 V estables para la alimentación de los operacionales.
- Los 7,5 V de la alimentación del Arduino, se realiza por medio de un regulador de tensión regulable.
- Ambos circuitos integrados cuentan con disipadores de calor.
- Posee protección de un fusible ante posibles cortocircuitos, evitando un fallo en otro posible componente.
- En el siguiente apartado 11.2, se detalla con más detenimiento las características de este montaje.

8.1.4. Circuitos impresos

Una vez fue planteada la funcionalidad de cada montaje, el siguiente paso será diseñar los circuitos impresos. Se adjuntará en el Anexo de planos los archivos necesarios para llevar a cabo su impresión.

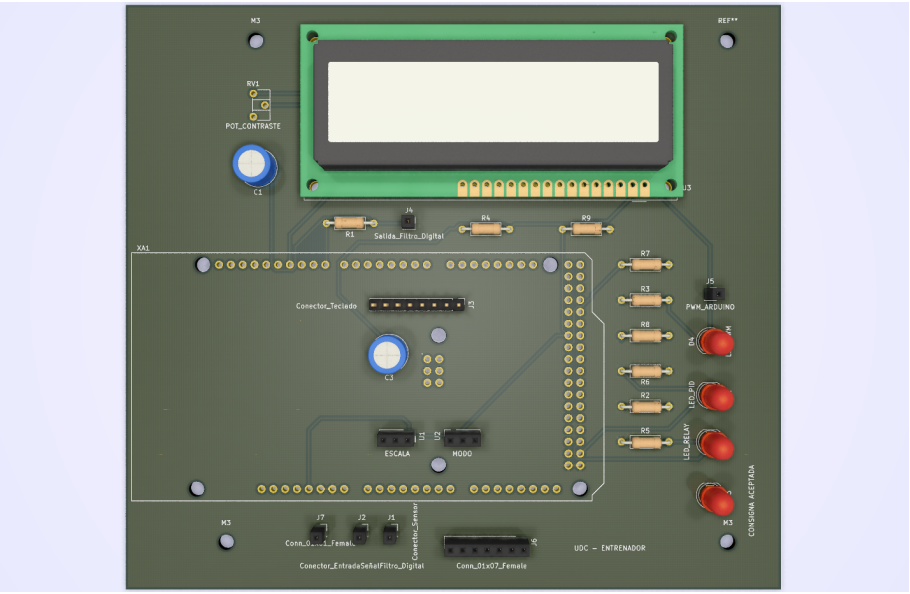


Figura 8.2 – Diseño del entrenador

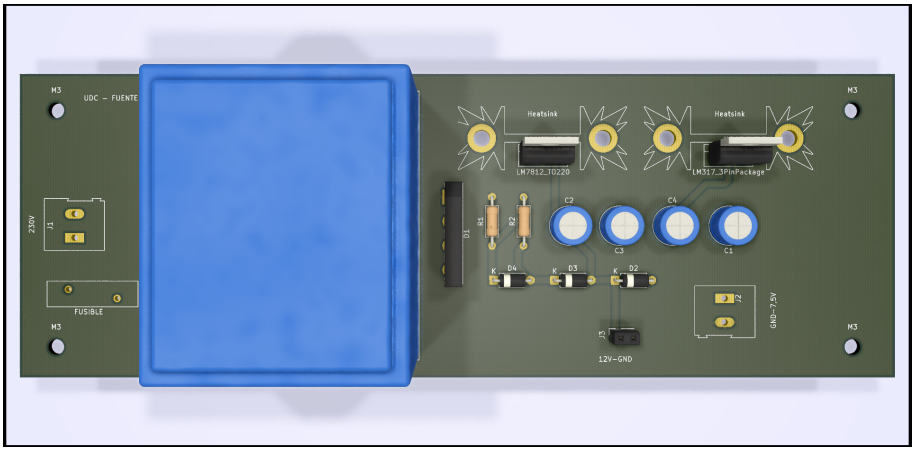


Figura 8.3 – Diseño de la fuente de alimentación

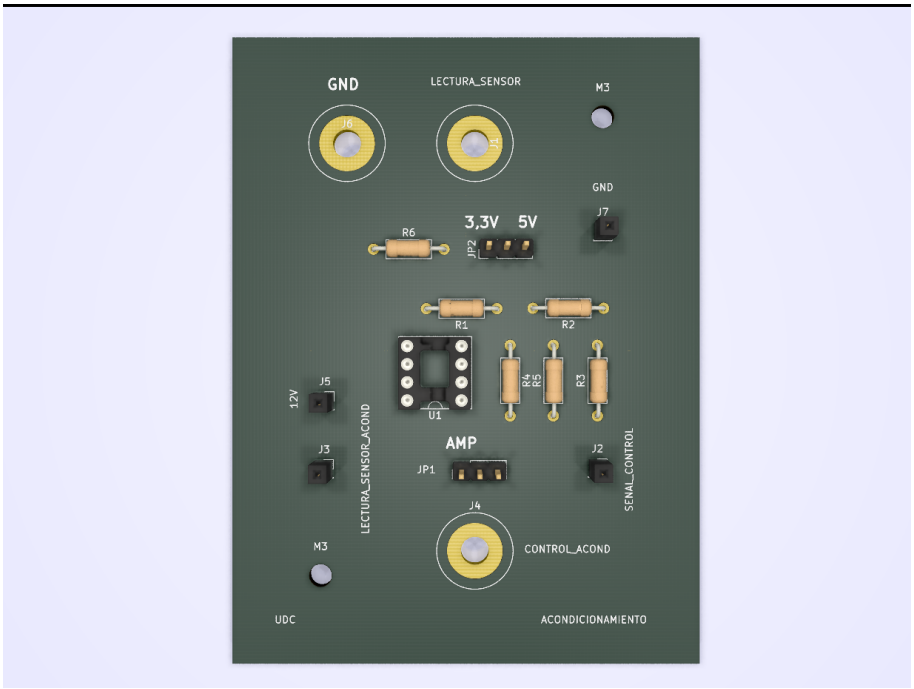


Figura 8.4 – Diseño final de la placa de acondicionamiento

Para el desarrollo de los montajes, se ha tenido que analizar las hojas de características para seleccionar los componentes que cumplían exactamente con las huellas disponibles, desarrollar unas nuevas para aquellos que no contaban con ellas o ajustar el ancho de los taladros. Además, se ha tenido en cuenta otras especificaciones, como la corriente especificada máxima, lo cual determinó aspectos como pueden ser el ancho de pista o el uso de disipadores.

8.2. Análisis técnico del microcontrolador sobre el entorno Simulink

Para poder juzgar correctamente los resultados obtenidos, primero habrá que conocer las características que definen al propio modelo a nivel de hardware.

8.2.1. Frecuencia de muestreo máxima

Dedicamos un apartado a este tema, pues este parámetro es de gran importancia, ya que fijará el tiempo de muestreo que actuará sobre cada uno de los programas mostrados en este trabajo. Independientemente de la frecuencia de muestreo máxima que permitiría el hardware en sí, si el Arduino ejecuta un programa de Simulink, la placa solo se podrá muestrear a una frecuencia máxima de 1 KHz, tal y como se refleja en la figura 8.5, o si se configuran hasta 6 señales, solo hasta un máximo de 200 Hz. (Esta información viene proporcionada por el mismo programa en su ayuda, y será corroborada).

Placa Arduino	Soporte para Shield	Ajuste interactivo y monitorización	Comentarios
Arduino Due*	S	S	El canal CAN no está soportado actualmente.
Arduino Uno*	S	S	Ajuste interactivo soportado a partir de la versión R2016b. Puede registrar una señal a 1 kHz o hasta 6 señales a una velocidad de 5 ms.
Arduino Leonardo*	S	S	
Arduino Mega 2560*	S	S	Registre una señal a 1 kHz o hasta 6 señales a una velocidad de 5 ms en la versión R2016b.

Figura 8.5 – Captura que especifica las prestaciones máximas de la placa

Tras lo dicho anteriormente, y partiendo de esta premisa, haremos una prueba. Suponiendo que el tiempo de muestreo mínimo es de 1 ms, haremos una prueba respetando el teorema de Nyquist, ya que de lo contrario aparecerá aliasing. Este nos indica que la frecuencia de muestreo ha de ser, por lo menos, dos veces superior a la frecuencia de la señal de entrada que quiere ser muestreada.

$$\text{Frecuencia de muestreo} \geq 2 * \text{Frecuencia de la señal de entrada}$$

Por lo tanto, generaremos una señal cuadrada de 500 Hz con ayuda de un segundo Arduino, un UNO por ejemplo, y probaremos si el muestreo se realiza correctamente. Para ello, representaremos en el PWM del MEGA la señal leída por este a la entrada.

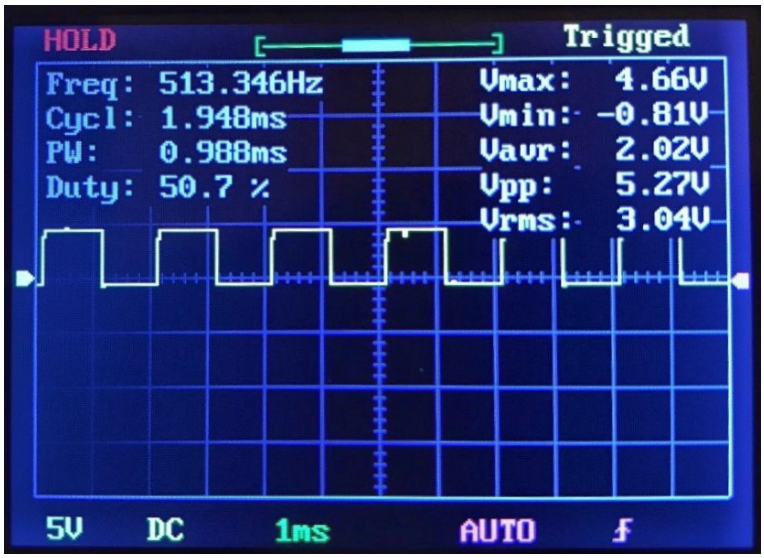


Figura 8.6 – Señal muestreada correctamente

Como se puede ver, se consigue muestrear correctamente la señal de inicio 8.6. Si estas pruebas son repetidas tratando de disminuir la frecuencia de muestreo, o bien aumentando la frecuencia de la señal de entrada, la lectura que se obtiene es inestable. En vez de aparecer sobre el osciloscopio una señal cuadrada, se obtiene una de ciclo de trabajo variable y cambiante, indicando el fallo en el proceso.

La forma de indicar esta frecuencia de muestreo dentro del programa es por medio de un parámetro, presente en todos los bloques de entrada o en aquellos que de alguna forma el tiempo de muestreo tiene que mantenerse constante, como el bloque PID. Este parámetro es el “Sample Time”, ajustable tal y como se muestra en la figura 8.7.

Un requisito que ha de cumplirse, porque por lo contrario el programa devolvería un error, es que todos los bloques necesitan compartir frecuencia de muestreo.

Por lo tanto concluiremos que el Arduino, en un principio, sí puede muestrear a 1 KHz de frecuencia. Y tendremos en cuenta Nyquist y dicho valor máximo de muestreo a la hora de introducir señales que pretendan ser muestreadas utilizando un Arduino.

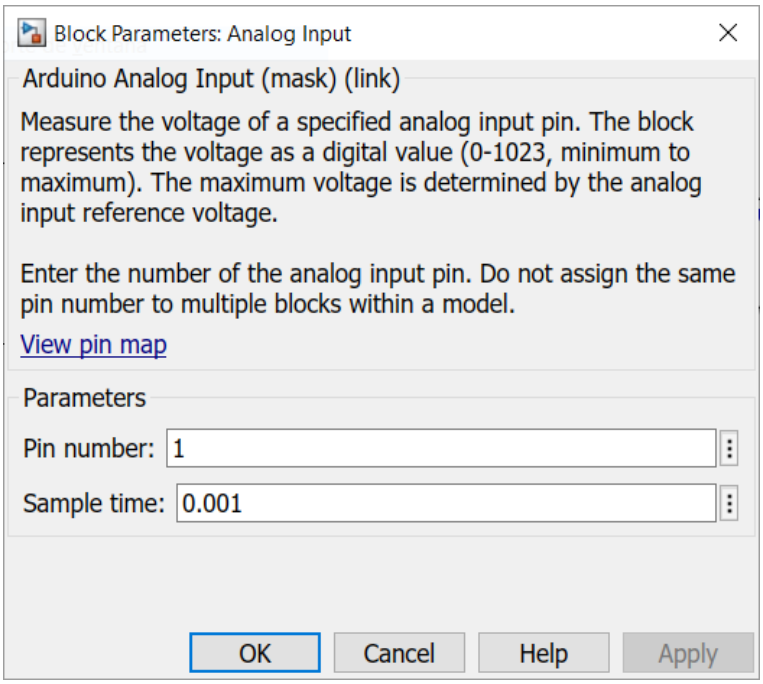


Figura 8.7 – Ajuste del Sample Time

8.2.2. Frecuencia de la generación de la señal de PWM

Los pines de PWM están presentes dentro de la placa con la finalidad de obtener valores intermedios de tensión dentro del rango de 0 y 5 voltios. Esto lo hace por medio de la generación de una señal cuadrada con ciclo de trabajo variable con una frecuencia, en un inicio, constante.

Pero no todos los pines van a generar un PWM de la misma frecuencia. Por ejemplo, en el caso del Arduino UNO, solo tendrán una frecuencia de 1 KHz los pines 5 y 6, y en el caso del Mega, el 4 y 13. En el resto, esta frecuencia sería de aproximadamente 500 Hz.

Esto fue comprobado a mano con ayuda de un oscilospio, testeando la frecuencia máxima por grupo de pines. Además, fue corroborado por un comentario en el foro oficial de MatLAB 8.8.

> Currently the PWM block generates in the default frequency. The frequency of the pins 4 & 13 is 976.5Hz while the other PWM pins generate at 490.1Hz.

Figura 8.8 – Imagen del comentario en cuestión

PWM Frequency and Pin	Arduino Board			
	Arduino MKR1000	Arduino Uno		Arduino Due
PWM Frequency	~772 Hz	~980 Hz	~490 Hz	~1000 Hz
Pin	all PWM supported pins	5 and 6	Other PWM supported pins	all PWM supported pins

Figura 8.9 – Simulink sí lo aclara para el caso del UNO

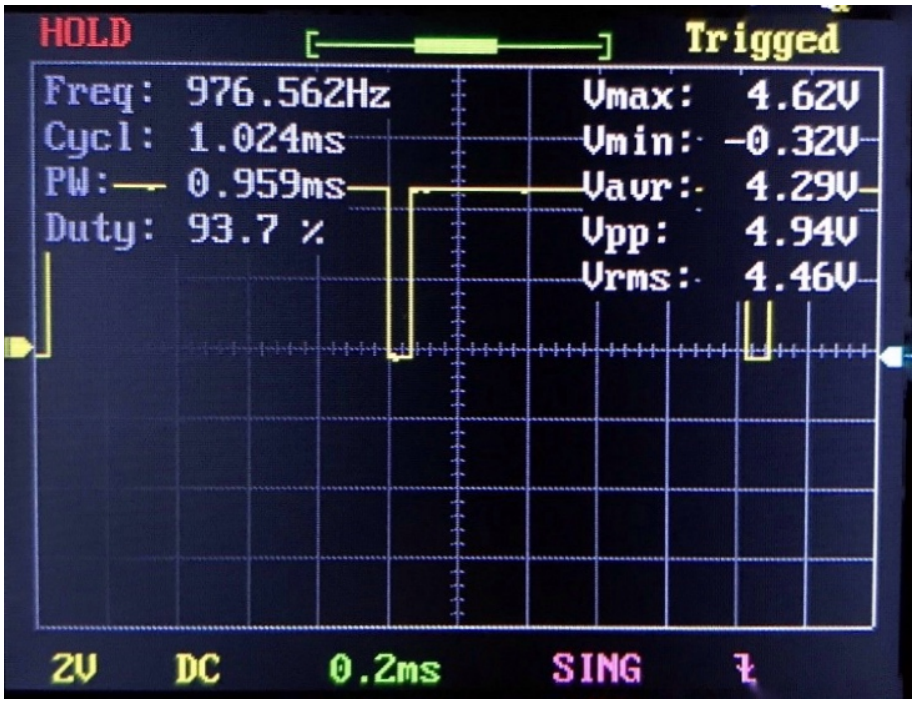


Figura 8.10 – Generación de un PWM de aproximadamente, 1 KHz

En la ayuda del Simulink solo se especifica la frecuencia del Arduino Uno, que como hemos visto es diferente a los del Mega. Aun así, la información que aparece del UNO es correcta.

Poniendo un ejemplo práctico, si se programa un Arduino para producir una señal cuadrada de 100 Hz, esta será efectuada con un refresco de 1 KHz suponiendo que se usa el pin 4. El ciclo de trabajo de la señal se ajustará automáticamente para cumplir con la representación de la señal demandada por el usuario.
Pero cuando el Arduino ejecuta un código de mayor complejidad, podremos observar cómo la frecuencia de actualización del PWM disminuye. Se debe a que debido al consumo elevado de recursos, la ejecución del código no puede mantenerse a una actualización 1 milisegundo, (frecuencia máxima de muestreo) y se ve retardada, disminuyendo la frecuencia máxima efectiva del microcontrolador como conjunto y haciéndose visible sobre la generación de PWM. Por lo tanto, bajo estas condiciones de retardo, todas las especificaciones se verán reducidas a este nuevo máximo efectivo.

Utilizaremos esta característica para evaluar el rendimiento real de cada uno de los programas, para así definir un valor real correcto de tiempo mínimo de muestreo.

8.3. Implementación del regulador embebido

Aprovechando las instalaciones del laboratorio, se ha querido implementar un regulador PID, estándar de la regulación en la industria, implementándolo por medio de las librerías de Simulink. Se comenzará con un programa que reúne los mínimos requisitos para ser funcional para comprobar su correcto desempeño. A continuación, se presentará un programa que permita la introducción de las constantes por parte del usuario, así como un modo de ejecución relay-feedback y por último, un sistema completamente autoajustable. La idea es ir disminuyendo el grado de participación del usuario en cada programa, hasta llegar al diseño completamente autónomo.

Este regulador es complementado por un montaje que se integra con los elementos ya presentes en el laboratorio y que cumple las especificaciones anteriormente mencionadas.

8.3.1. Programas de regulación desarrollados

A continuación se describirá el funcionamiento de cada uno de los programas diseñados. Debido al tamaño de estos, por motivos de espacio y falta de visibilidad, no se adjuntan las capturas de pantalla de los mismos. Serán adjuntados en forma de plano en su respectivo Anexo.

8.3.2. PID embebido con constantes predeterminadas

Este primer programa pondrá en práctica los parámetros calculados previamente gracias a la ejecución en modo relé. Para ello, se diseñó un programa bastante simple con los parámetros ya cargados, en el cual se puede introducir la consigna por teclado y la variable de salida es visualizada por pantalla. Cabe destacar, que estos valores son capaces de regular para un amplio margen de consignas, aunque estarán únicamente optimizados para el valor en torno al cual se ha puesto a oscilar el sistema durante la ejecución del relé (26ºC). En este programa, que consta de la siguiente estructura mostrada en la figura 8.11, vemos que prevalece la simplicidad.

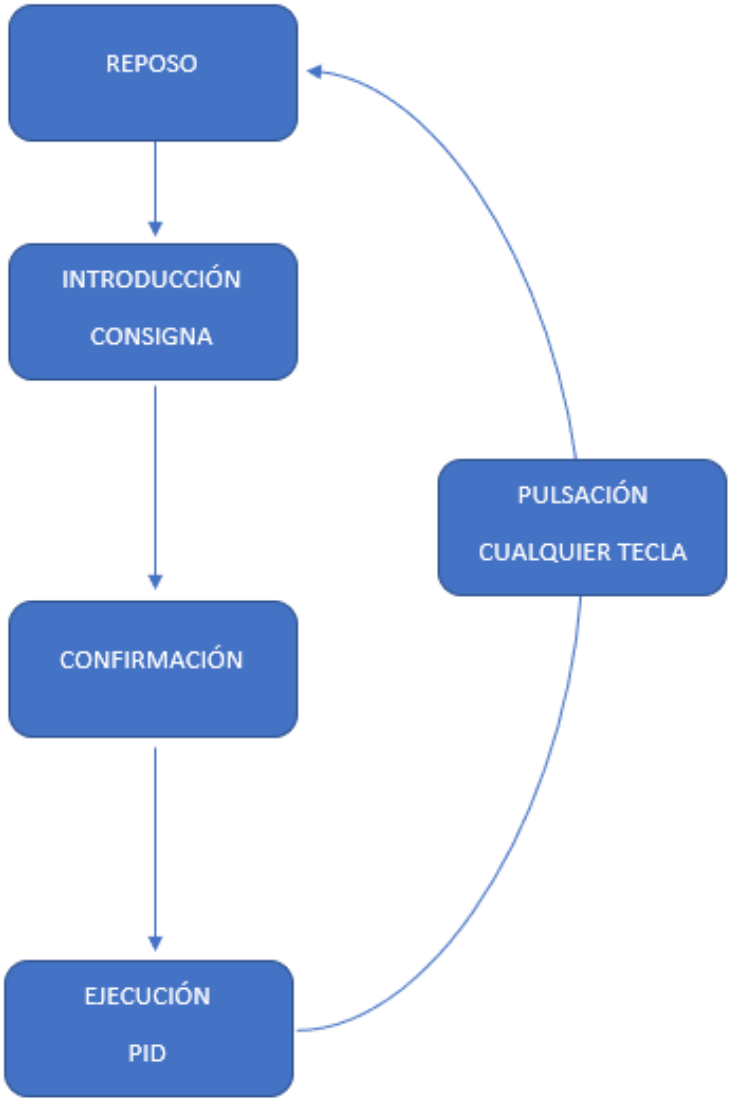


Figura 8.11 – PID con constantes predeterminadas

El sistema está en reposo hasta que la consigna en grados centígrados es confirmada por el usuario por medio del teclado, (para esta función en concreto se ha creado un código desde cero que permite compatibilizar los teclados matriciales con el entorno de Simulink). En este momento, utilizando los parámetros guardados en memoria, ejecutará el algoritmo de control indefinidamente. Su ejecución podrá ser interrumpida tras una pulsación sobre el teclado. Tras una nueva confirmación, el programa retomará la regulación con una nueva consigna. En todo momento, se ve sobre la pantalla LCD, la consigna actual y el estado de la variable del proceso, (temperatura).

Debido a que para cambiar los parámetros de regulación habrá que conectar el Arduino al PC y desde allí cambiarlos manualmente, este diseño dista bastante de una solución autónoma. Por este motivo, se ha desarrollado el programa a continuación.

8.3.3. PID embebido con constantes seleccionables por el usuario

En este programa, se permite tanto la introducción de la consigna, como las constantes del regulador por medio del teclado. En la pantalla LCD se mostrará la consigna, la variable de salida y las constantes introducidas.

El funcionamiento del programa se muestra en la figura 8.12:

Como se puede observar, cuenta con un modo relé, sobre el cual se pueden calcular las constantes como se ha reflejado en el anexo de cálculos correspondiente 11.3. Esto será posible gracias a su conjunción con el modo de ejecución externa. Dicho modo se ejemplifica en el siguiente apartado 12.5.

Debido a que hay que compatibilizar el programa escrito en C con el de Simulink, las constantes de la regulación solo pueden ser de valores enteros. De lo contrario, debido a las diferencias entre ambos estándares no se podía visualizar ni capturar de forma alguna el valor tecleado.
Tratando de suplir esta limitación, nace la implementación de un factor de escala, con el cual las podrán ser divididas entre 10, aumentando la versatilidad del sistema.

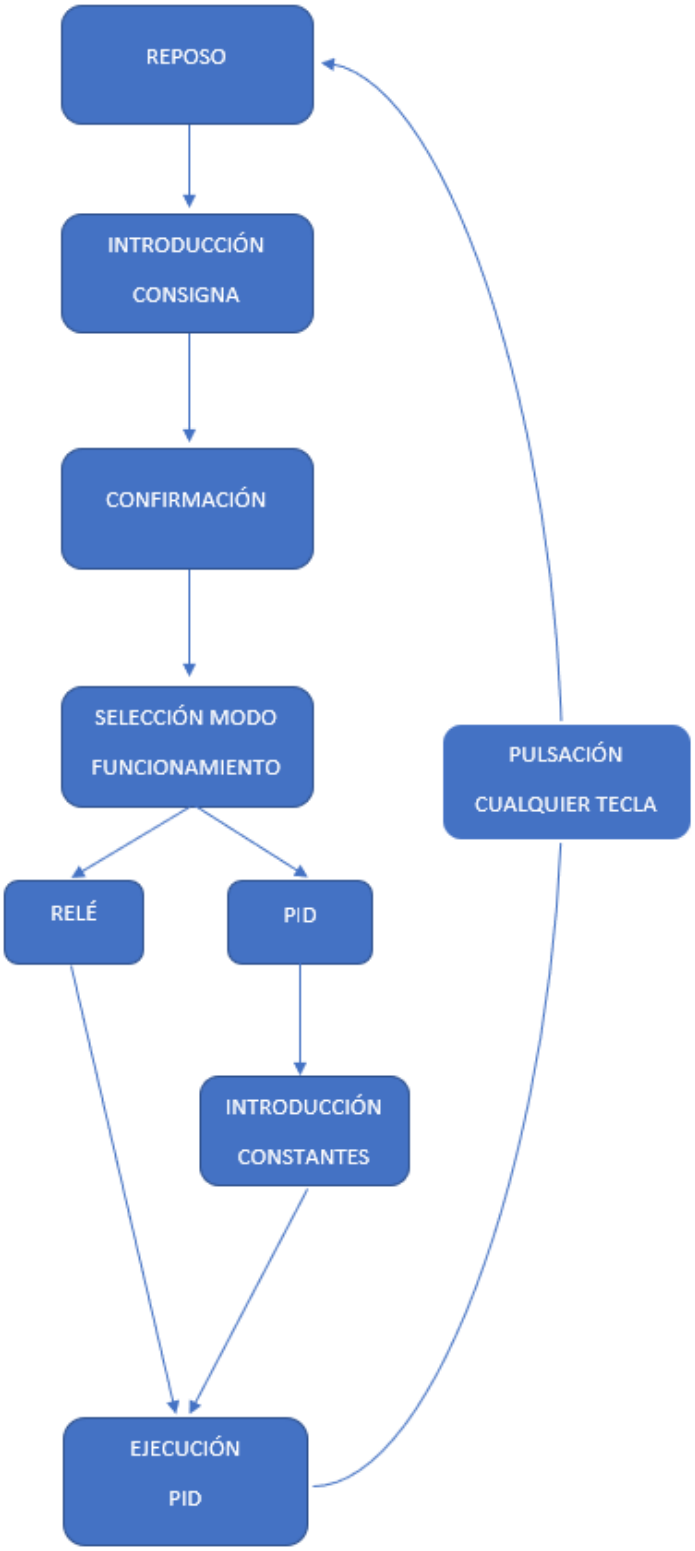


Figura 8.12 – Flujograma del programa

8.3.4. PID embebido autoajustable

El hecho de tener que introducir las constantes por parte del usuario se ha eliminado en este último programa. Utilizando un código que se ejecuta tal y como lo muestra este flujo-

grama 8.13, es capaz de funcionar de forma totalmente autónoma, temporizando, realizando cálculos y llevando a cabo la regulación.

Cuando por teclado se le introduce la consigna a la cual se quiere llegar, primero se ejecutará el modo relé. En este modo, se medirá el tiempo de oscilación transcurrido, a través de una temporización interna efectuada sobre la placa, y se guardará cuál es el máximo y mínimo de la variable del proceso, (temperatura por ejemplo) y sus respectivos tiempos. Esta ventana de histéresis que abarca el valor máximo y mínimo fue predefinida y es de valor fijo. Esto permite disminuir las operaciones que se realizan en el interior de la placa, maximizando el rendimiento.

A continuación, se llevará a cabo el cálculo de las constantes. Estas son procesadas en el interior del microcontrolador e inmediatamente, son puestos a prueba por el sistema, ejecutando el regulador PID. Estos parámetros son característicos del sistema, y quedarán guardados para futuras iteraciones del sistema, siempre y cuando la tensión se mantenga sobre el Arduino. Una desconexión de la tensión reseteará las variables.

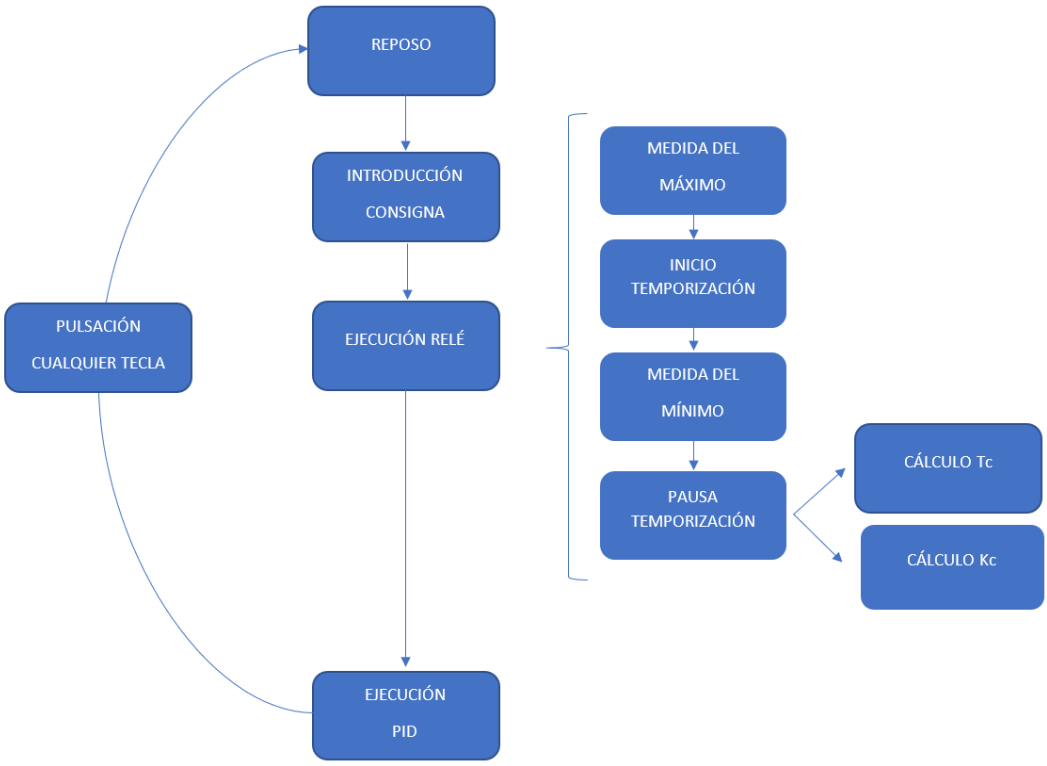


Figura 8.13 – Esquema del funcionamiento del programa autoajustable

8.4. Análisis de tiempos

Para poder analizar el comportamiento de la placa en términos de retardos, se diseñó una pequeña porción de código, la cual, en paralelo con el programa normal, determinará si estos retardos ocurren o no en el cálculo. Estos bloques, realizan la función de invertir el estado de la salida del Arduino lo más rápido posible (utilizando su muestreo más rápido, tal y como se muestra en el siguiente apartado 8.2.1). Si el microcontrolador lee en el pin que la salida está a nivel bajo, la pondrá a nivel alto, y viceversa.

Para ello, se dedicaron dos pines de placa, el 47 y el 49. Al tratarse de salidas de PWM normales, si todo ocurre sin inconveniencias, proporcionarán una frecuencia de 500 Hz, como fue documentado en el siguiente apartado 8.2.2. Si por lo contrario, el sistema no es capaz de ofrecer la velocidad en la respuesta necesaria, la frecuencia de esta señal generada bajará. Esta bajada puede ser, por lo tanto, cuantificada e interpretada.

Este "benchmark" se ha implementado en todos los programas diseñados, pudiendo así evaluar si existen retardos y de si afectan estos a la ejecución normal del programa. Cabe destacar que el tiempo de muestreo se ha colocado al mínimo valor posible, 1 milisegundo, para poder maximizar los delays y así poder hacerlos visibles.

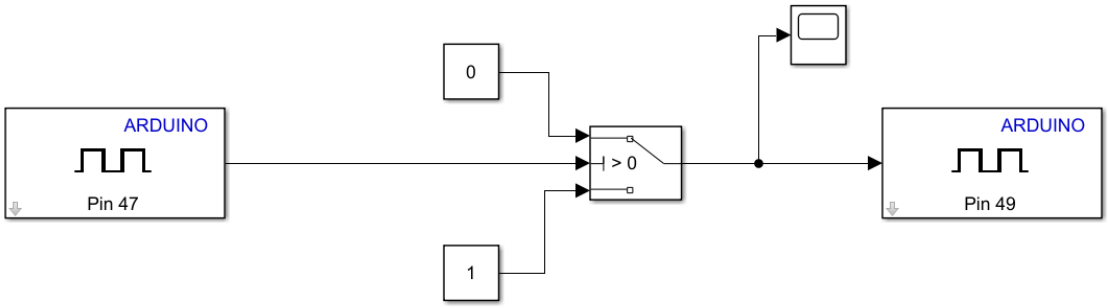


Figura 8.14 – Captura del fragmento de pruebas

Se calculará el tiempo de retardo de la siguiente manera:

$$\text{Tiempo de retardo} = \frac{1 \text{ / Frecuencia máxima de funcionamiento}}{2} \tag{8.1}$$

Se interpretará que la frecuencia máxima de funcionamiento será el doble de la señal observada de PWM, pues esta consta de dos estados, (nivel alto y bajo de tensión), que ocurren uno a continuación del otro.

Junto con estos cálculos, se ha hecho una tabla resumen donde se comparan las prestaciones de cada una de las opciones planteadas en la tabla 8.1.

Analizando los resultados, se puede comprobar que con los programas estándar de regulación funcionarán a una frecuencia máxima de aproximadamente 70 hercios. Debido a este hecho, se ha reducido la frecuencia de muestreo de nuestros programas, ajustándose esta a los valores reales de funcionamiento. Concluiremos que para la aplicación en concreto con la cual trabajamos, con sistemas con unos tiempos de respuesta en el orden de segundos y el hardware del cual disponemos, es un valor suficiente, con el cual se puede obtener resultados positivos.

Programa	Frecuencia máxima funcionamiento (Hz)	Tiempo de retardo (ms)
PID con constantes predet.	70	7
PID con constantes seleccionables	72	7
PID autoajustable	78	6

Tabla 8.1 – Tabla resumen de los tiempos de retardo

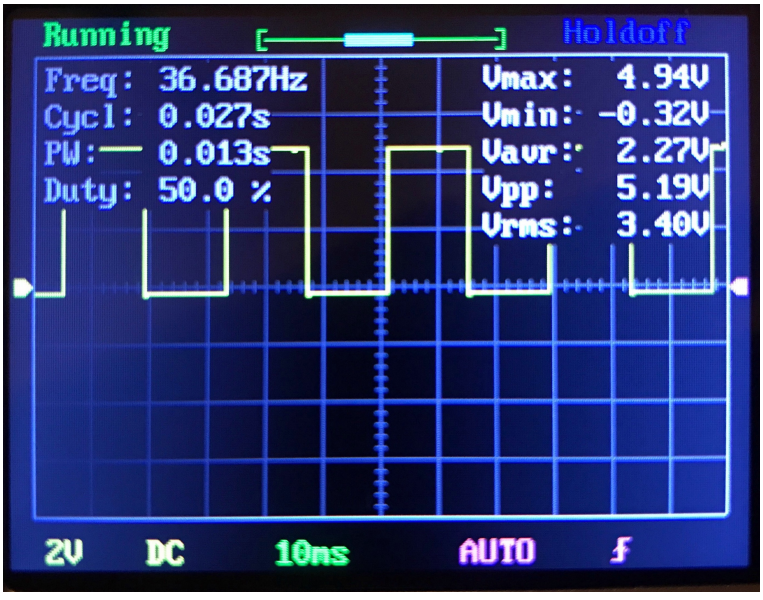


Figura 8.15 – Ejemplo de la señal de PWM generada a aproximadamente 36 Hz

8.4.1. Conclusiones

Aclaremos que estos valores representan el valor máximo de retardo alcanzado. En los tres programas esto sucede mientras el bloque-función de PID está operativo. Viendo que los tiempos de retardo son, aproximadamente dentro de un margen de error, el mismo, deduciremos que este será la sección del código que más recursos consuma. Una ventaja observada en este análisis es que este valor de retardo se muestra constante en el tiempo, además de ser reproducible tras varias ejecuciones consecutivas del código

Conociendo ya los delays presentes, observaremos que el muestreo de 1 ms es desaprovechado por las mencionadas ralentizaciones que aparecen en el microcontrolador al ejecutar el código. Debido a esto, colocaremos el tiempo de muestreo en un valor de **50 ms**, (tiempo que actuará en la totalidad de reguladores propuestos), siendo este un valor más adecuado a nuestra situación en particular.

8.5. Otras funciones

Como ya hemos mencionado anteriormente, la posibilidad de poder adaptar al código cualquier tipo de bloque, solo estando limitados realmente, (aparte del tamaño máximo de la memoria de programa, que el caso del Mega, como ya hemos mencionado, es suficientemente amplia para su uso en estos ejemplos que se proporcionan, y de los recursos de cálculo, que para este tipo de tareas, son suficientes), por nuestra imaginación, nos permite hacer otro tipo de programas, para experimentar con diferentes soluciones, que pueden complementar nuestro trabajo en un determinado aspecto. Es por este motivo, por el que se decidió probar una de estas virtudes, aplicada a la placa Arduino.

8.5.1. Regulador Fuzzy

Aprovechando la naturaleza polivalente de MATLAB-Simulink, se ha diseñado un programa utilizando el bloque controlador Fuzzy presente en el software. De esta forma, se ha portado a la plataforma Arduino un controlador con naturaleza basada en la lógica difusa. Esta solución puede representar una ventaja a la hora de calibrar un sistema, basándose este en la relación de reglas de forma relativa sin conocerse en ningún momento los parámetros tradicionales de regulación.

De esta forma, se ha dejado un programa a disposición del alumno, con un archivo de control "*.fis*" cargado, con el cual puede experimentar diferentes resultados.

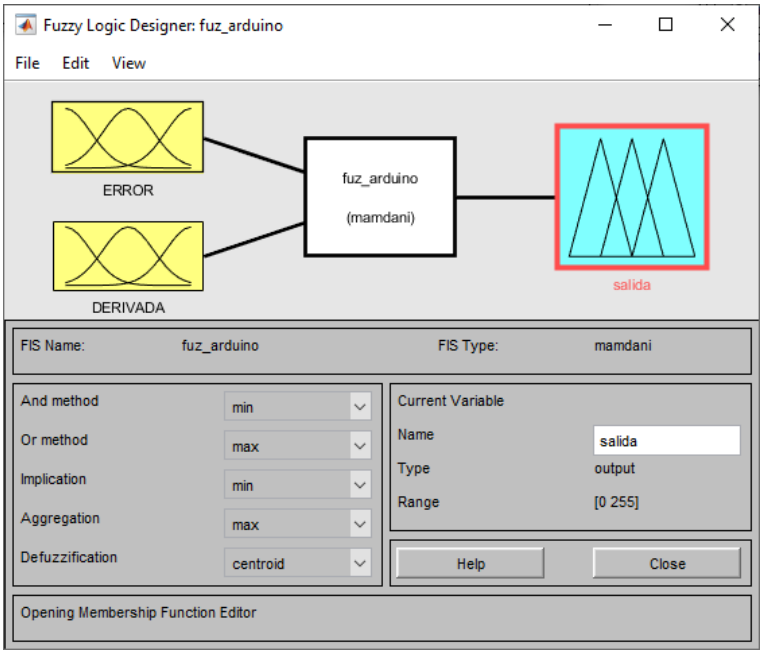


Figura 8.16 – Bloque del controlador Fuzzy

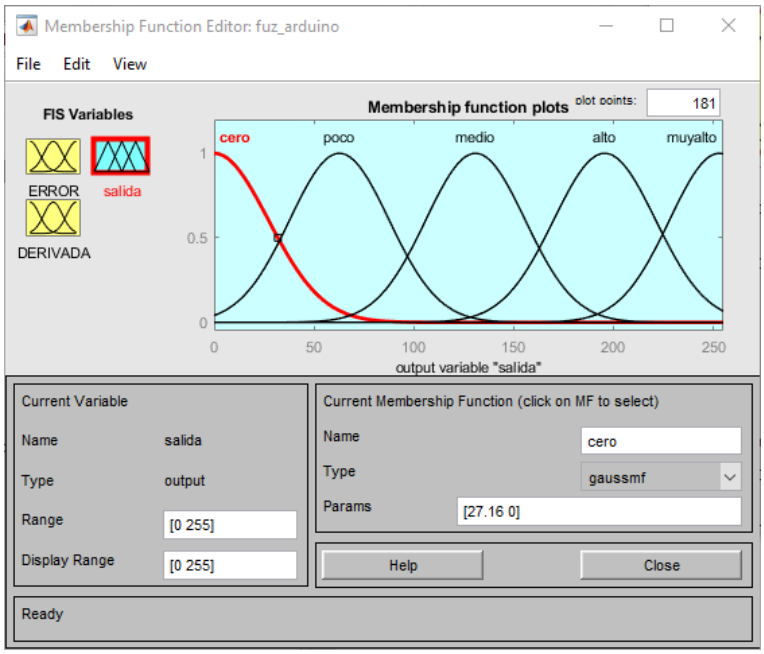


Figura 8.17 – Función de pertenencia del bloque Fuzzy

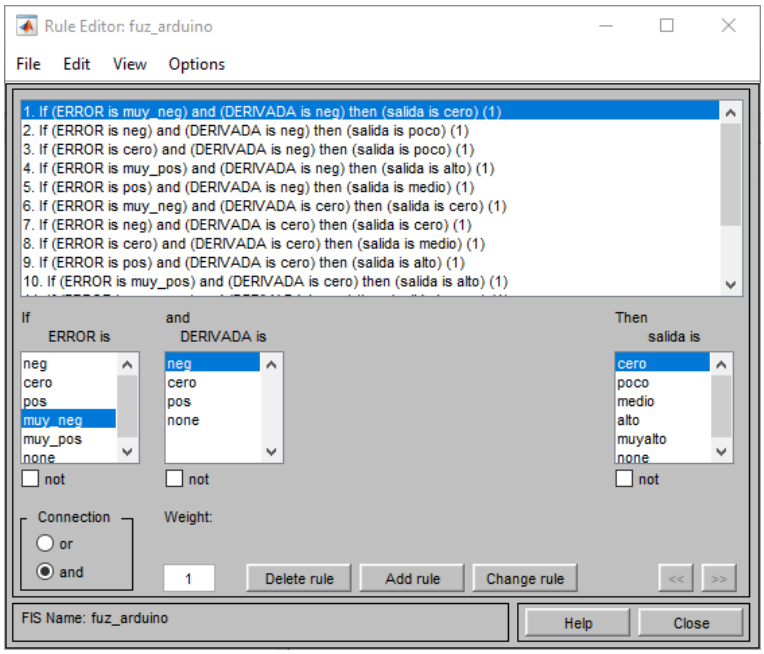


Figura 8.18 – Reglas de la regulación

Se ha analizado el rendimiento de esta solución, alcanzándose una frecuencia máxima de funcionamiento de tan solo 3 Hz. Además, debido al tamaño del código generado, solo el Arduino Mega es capaz de albergar este programa en su interior, quedando el UNO excluido.

8.5.2. Filtrado digital

Un segundo ejemplo adjuntado es el filtro paso bajo digital. Para ello, se introdujo un bloque ya pre-programado, el cual posee cuatro parámetros configurables. Dos entradas, la señal en que se desea filtrar y la frecuencia de corte deseada y con dos variables internas, el tiempo de muestreo y el de orden del filtro.
La función de transferencia del filtro es la propia de uno de únicamente orden 2, por lo que los únicos dos órdenes que se podrán implementar, serán los de 1 y 2, pero para ejemplificar esta aplicación básica, será suficiente. La otra limitación y esta más obvia, es que la señal de entrada no podrá superar la frecuencia a la cual el Arduino pueda muestrearla correctamente. Este límite, irá impuesto por la frecuencia de Nyquist.

Para poner a prueba este programa, se generó, con ayuda de una segunda placa Arduino, una señal cuadrada de baja frecuencia. Reduciendo la frecuencia de corte a valores cada vez más próximos al de la frecuencia de la propia señal provocará la eliminación de los armónicos de la función cuadrada en sí y aproximará, como es lógico, cada vez más su forma a una curva sinusoidal pura. A continuación se puede observar la simplicidad del programa:

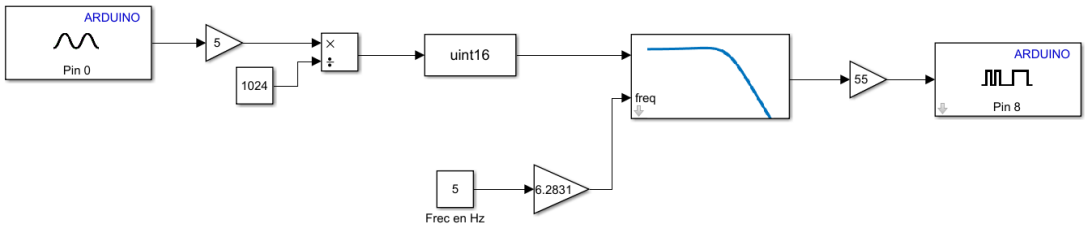
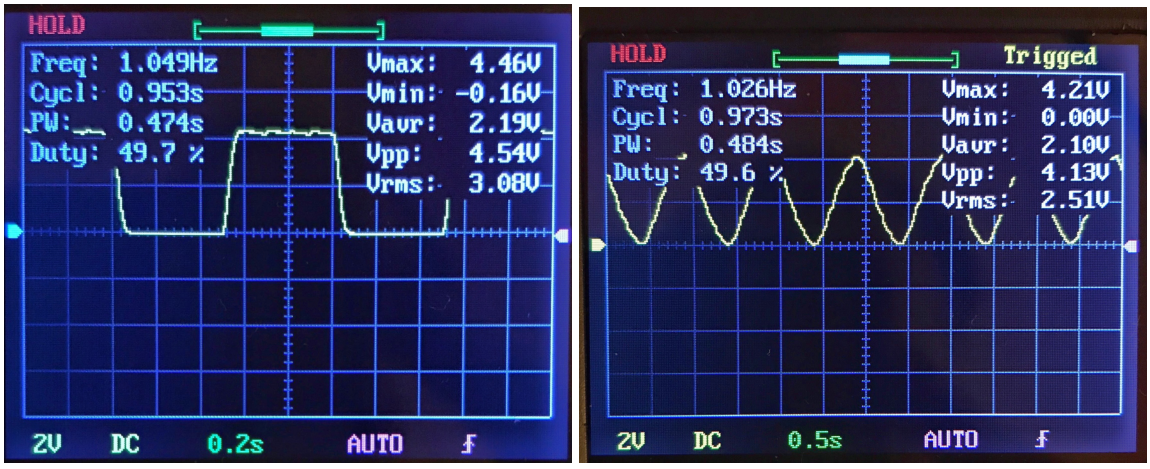


Figura 8.19 – Captura del filtro digital



(a) (b)

Figura 8.20 – Ejemplos de filtrado de una señal cuadrada de 1 Hz, con diferentes frecuencias de corte

8.6. Análisis de la regulación

Una vez concluido todo el análisis anterior, se probará el funcionamiento de los programas de control en la planta del horno. En la teoría, las plantas situadas en el laboratorio de control, son sistemas con una respuesta lenta, por lo que las prestaciones demostradas por el microcontrolador deberían ser más que suficientes.

Como ya se ha comentado, la temperatura a la que se encuentra el laboratorio, juega un factor a tener en cuenta en los tiempos de, sobre todo, de bajada de temperatura. Es por ello, que al hacerse las capturas en días distintos, éstas difieran un poco entre sí. La temperatura aproximada del laboratorio cuando se tomaron las capturas era de aproximadamente 18ºC.

Al tratarse de un sistema muy lento, el mencionado tiempo de oscilación en segundos del sistema, (Tc), da como resultado un valor muy elevado. Esto implica, que la parte derivativa del regulador cobra una gran importancia. Cuando la constante derivativa se eleva en valor, provoca grandes oscilaciones en la señal de control, lo cual puede estropear el actuador con el cual trabajemos. Esta es la razón por la cual la parte derivativa se mantiene desconectada en muchas soluciones industriales. Para poder solucionarlo, Simulink incluye la posibilidad de controlar un parámetro propio del regulador, el parámetro N, constante del filtro, la cual, por lo normal en PID comerciales, puede tomar un valor entre 1 y 33 [22], y cuanto menor, más efecto tiene. Situando este parámetro con un valor de 0,001, (valor elegido experimentalmente), se consiguieron valores satisfactorios con una señal de control suave, poco oscilante y progresiva, lo cual es beneficioso para cualquiera que sea el actuador del sistema.

8.6.1. PID embebido con constantes seleccionables por el usuario

Si las constantes de regulación se introducen de forma azarosa, lo más probable es que la regulación no se efectúe de forma óptima, alargándose el período de estabilización de la variable a controlar, o incluso, no llegando esta nunca a un valor estable. En la siguiente captura 8.21, se puede observar este hecho, la señal, pese a llegar a estabilizarse en el valor de consigna, ha sobreoscilado de forma muy pronunciada, por lo que el tiempo de establecimiento se ha alargado de forma evidente.

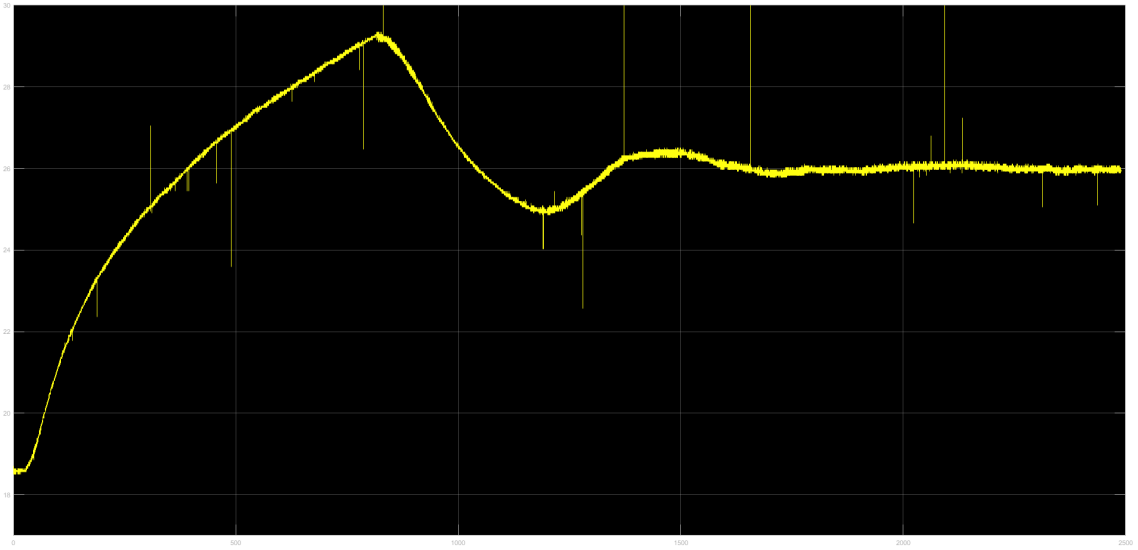


Figura 8.21 – Resultados con las constantes introducidas por teclado

Como este programa también posee el modo de ejecución relay-feedback, se adjunta un resultado del mismo:

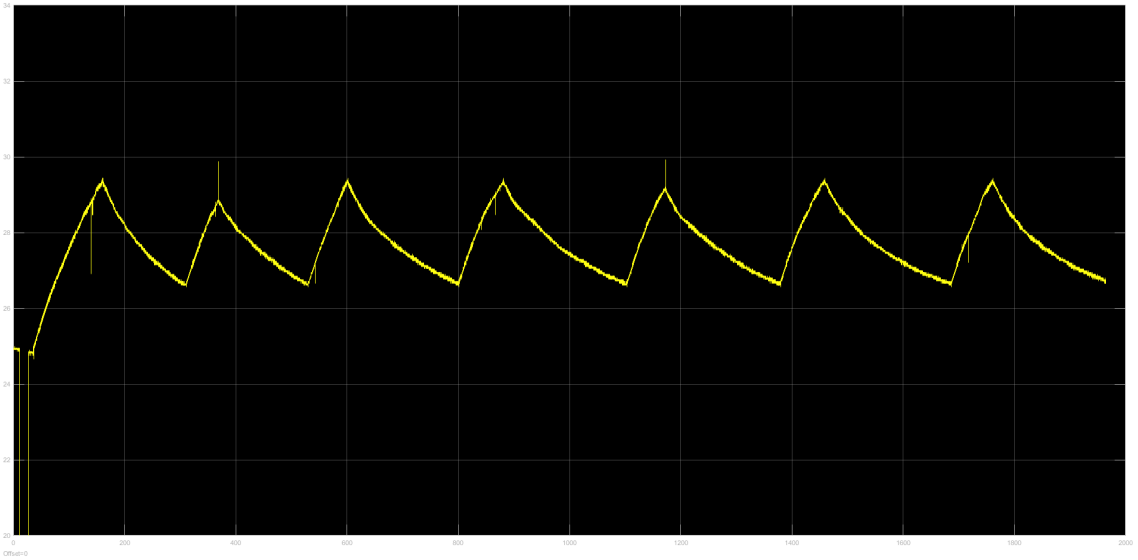


Figura 8.22 – Modo relé funcionando

8.6.2. PID embebido con constantes predeterminadas

Al contrario que con el caso anterior, con un valor optimizado se conseguirá una respuesta mucho más rápida, menos oscilante y como es lógico, más adecuada.
Se obtuvieron los siguientes resultados tras aplicar los ajustes mencionados:

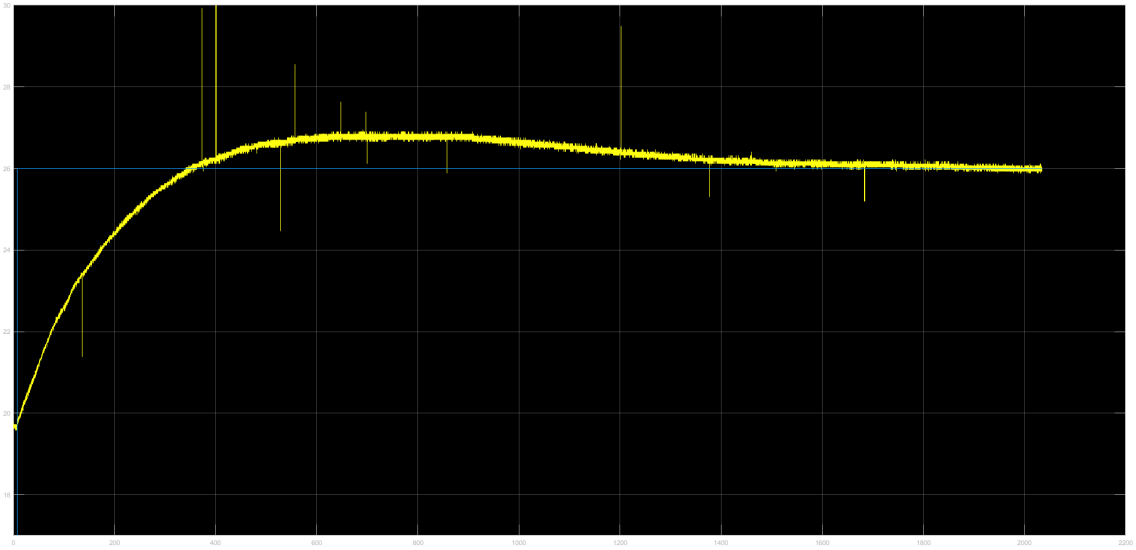


Figura 8.23 – Sistema funcionando con las constates calculadas

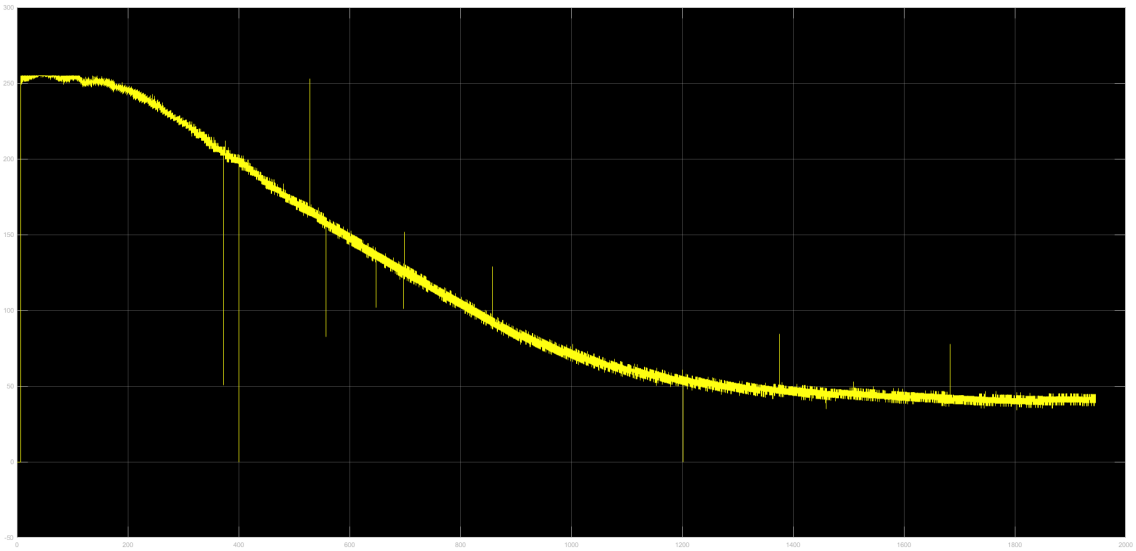


Figura 8.24 – Señal de control de apariencia suave

8.6.3. PID embebido autoajustable

Cuando se dejó al sistema calcular las constantes de regulación por sí mismo usando el mencionado programa, estas se procesaron correctamente. Si analizamos los parámetros correspondientes a la fase de relé y el tiempo temporizado, vemos que se tratan de los valores pertinentes. La regulación, por lo tanto, se tendría que realizar de forma correcta. En la figura 8.25, se ilustra.

En esta captura se puede ver una primera fase, que es reconocible por una gran oscilación, propia de la ejecución del relé, con objetivo de calcular los parámetros de la regulación y a continuación la estabilización del valor introducido por teclado, en este caso, 26ºC. Cuando los

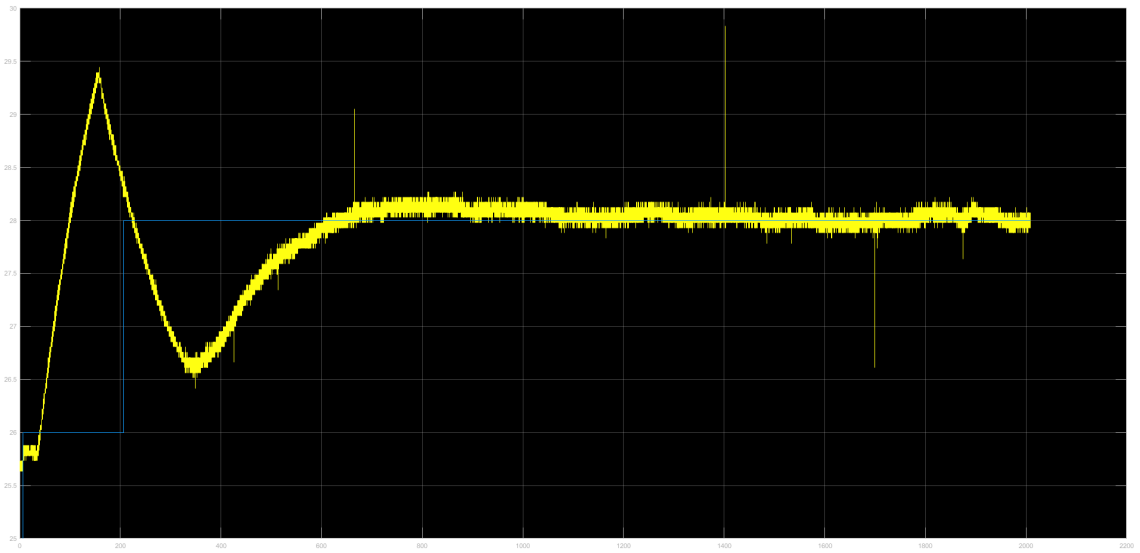


Figura 8.25 – Sistema autoregulable funcionando

valores son calculados, son mostrados por pantalla. En esta se representan cada una de las constantes, y en este caso, debido a la falta de espacio, al tratarse de una pantalla de 16x2; se ha añadido únicamente el valor de la salida medido en grados centígrados, variable llamada PV, ”Processs Value“ 8.26.

Figura 8.26 – Muestra de las constantes y la evolución de la variable del proceso por pantalla

8.7. Conclusiones

El objetivo con el cual nació este trabajo era el de investigar en el mundo del hardware de bajo coste posibles soluciones que permitiesen su integración con los sistemas de control. Esta sinergia permitiría crear un entorno más directo, simple, de menor coste y el cual podría acercar a un público más joven las bases de los microcontroladores, la programación y la automática.

Tanto Simulink como Arduino eran ya usados en la escuela y uniendo sus virtudes se ha conseguido una base sólida de desarrollo sobre la que asentar nuestro estudio, añadiendo nuevas posibilidades a las bases existentes y analizando otras posibles soluciones.
Es por ello por lo que se ha optado por integrar los programas de control en el entorno ya conocido del laboratorio de control, en el cual se ha comprobado la eficacia del sistema para esta tarea en concreto, pero además ha permitido conocer sus límites.

Su gran versatilidad a un precio y consumo reducido son grandes virtudes a encontrar en un montaje, pero estas mismas son las que ponen límites a su funcionalidad final. Estos afectan directamente a su rendimiento, el cual se ve comprometido a medida que la complejidad del programa aumenta. La unidad de cálculo del Arduino cumple satisfactoriamente con la regulación propuesta en este trabajo, pero si las exigencias fuesen mayores, necesidades de cálculo en el orden de kilohercios o superiores, habría que descartar su utilización.
También sería desaconsejable su uso para procesos relativamente complejos de datos, como son las tareas de proceso de imagen, vídeo y sonido. En estos ámbitos, las características de adquisición de datos y cálculo propias de la placa se ven sobrepasadas, pues esta no fue ideada para su uso en este campo en concreto.

Concluyendo, partiendo de una idea simple, se ha propuesto, posteriormente desarrollado y documentado un modelo plenamente competente que puede resultar como una buena base para la formación, ayudando en su periodo de aprendizaje, orientándolo hacia el ámbito de la ingeniería de control. Los alumnos, siguiendo los pasos marcados por la guía adjuntada en los anexos, podrán replicar desde cero la solución ejecutada, pudiendo, en el desarrollo de esta acción, tomar otros caminos que finalicen con el desarrollo de otra idea o aprovechar las posibilidades del diseño actual para reconducir el montaje hacia otro campo de aplicación.

9 ORDEN DE PRIORIDAD ENTRE LOS DOCUMENTOS

1. Memoria
2. Anexos
3. Presupuesto

TÍTULO: **IMPLEMENTACIÓN DE UN REGULADOR PID GENÉRICO EMBEBIDO**

ANEXOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2019

AUTOR: EL ALUMNO

Fdo.: ALEJANDRO FERNÁNDEZ ROMERO

Índice del documento ANEXOS

10 DOCUMENTACIÓN DE PARTIDA

ESCUELA UNIVERSITARIA POLITÉCNICA

ASIGNACIÓN DE TRABAJO FIN DE GRADO

En virtud de la solicitud efectuada por:
En virtud da solicitude efectuada por:

APELLIDOS, NOMBRE: *Fernández Romero,Alejandro*
APELIDOS E NOME:

DNI: [redacted] **Fecha de Solicitud:** Feb2019
DNI: *Fecha de Solicitude:*

Alumno de esta escuela en la titulación de Grado en Ingeniería en Electrónica Industrial y Automática, se le comunica que la Comisión de Proyectos ha decidido asignarle el siguiente Trabajo Fin de Grado:

O alumno de esta escola na titulación de Grado en Enxeñería en Electrónica Industrial e Automática, comunícaselle que a Comisión de Proxectos ha decidido asignarlle o seguinte Traballo Fin de Grado:

Título T.F.G:Implementación de un regulador PID genérico embebido.

Número TFG: *770G01A176*

TUTOR:(Titor) *Jove Pérez,Esteban*

COTUTOR/CODIRECTOR: *José Luis Casteleiro Roca*

La descripción y objetivos del Trabajo son los que figuran en el reverso de este documento*:*

A descrición e obxectivos do proxecto son os que figuran no reverso deste documento.

Ferrol a Martes, 3 de Septiembre del 2019

Retirei o meu Traballo Fin de Grado o dia _____ de _____ do ano _____

Fdo: Fernández Romero,Alejandro

DESCRIPCIÓN Y OBJETIVO:OBJETO:

Este Trabajo Final de Grado abordará la implementación de un regulador PID dentro de una de las plataformas de bajo coste basadas en microcontroladores integrados (Raspberry Pi o Arduino). El trabajo consistirá en el desarrollo de un regulador PID mediante MatLAB Simulink, y su comunicación con un sistema a través de una de las plataformas mencionadas. Una vez desarrollado el regulador, se programará internamente en la plataforma para su uso independiente. Además, se realizarán las guías necesarias para el soporte a futuros alumnos en el uso de este tipo de programación (Simulink-Rasberry Pi, Simulink-Arduino).

ALCANCE:

Estudio de la programación Simulink en plataformas hardware de bajo coste.

Estudio de las limitaciones del uso de este tipo de plataformas: comprobación de los tiempos de respuesta, de las capacidades...

Implementación de un regulador PID genérico en Simulink. El regulador tendrá que ser ajustable para poder usarlo en diversos sistemas sin necesidad de reprogramación.

Desarrollo de una guía de uso de las plataformas de hardware de bajo coste en Simulink. La guía incluirá la configuración tanto del PC como de la plataforma, juntó con la explicación de los diversos tipos de programación existentes con ejemplos básicos.

11 CÁLCULOS

11.1. Corriente máxima consumida por la placa

Tal y como se recoge en hojas de características de la CPU, ATMEGA 2560, existe una corriente máxima que podrá proporcionar el aparato, sumando la intensidad que da a través de sus pines de Vcc y GND, (200 mA) y pines de I/O, (40 mA). Pero además, hay que fijarse en la corriente que se transmite por grupo de pines. Estos se dividen por subconjuntos y cada uno posee su propia limitación, (100 o 200 mA). En caso de conectar elementos de alto consumo, tendrán que ser divididos y repartidos teniendo en cuenta esta regla de consumo. Haciendo una aproximación, se podría decir que el máximo de corriente que es capaz de proporcionar en total la placa es de sobre 800 mA [11]. Con este valor máximo se realizarán los cálculos de las siguientes etapas 11.2.

11.2. Etapa de alimentación

11.2.1. Obtención de los 12 V

Interesa conectar el montaje directamente a los 230 V de la línea eléctrica, para ello, se realiza una reducción de la tensión hasta un valor lo menor posible que permita proporcionar la tensión suficiente a los operacionales por medio de un transformador. Estos operacionales tendrán que proporcionar una tensión de hasta 10 V, por lo que los alimentaremos a una tensión de 12 V, debido a que consultando en la propia hoja de características del aparato se puede ver que posee una caída es de 1,5 V [28].

Los 12 V que alimentan los operacionales serán generados por un regulador de tensión, el LM7812, circuito diseñado para este propósito. Este regulador, permite proporcionar hasta 1,5 A de corriente, y su único requerimiento es que su tensión de alimentación sea de por lo menos 14,6 V. Es este valor, el que nos fijará la tensión del transformador. Uno de 12V eficaces, 16,97 V de pico, que será la tensión que tendremos en los condensadores de entrada del regulador (sin tener en cuenta las posibles caídas en diodos y el propio trafo), es superior a 14,6 V [25], por lo que nos vale. Es importante tener en cuenta el excedente de tensión a disipar en el regulador. Este, nos provocará un calentamiento en el aparato, lo que hará necesario la utilización de un disipador de calor.

Teniendo en cuenta la siguiente fórmula [16], se puede calcular la potencia total que disipará el regulador en funcionamiento:

$$\text{Potencia} = (\text{Vin - Vs}) * \text{I}$$

Siendo I el valor máximo de corriente que puede consumir el Arduino 11.1, (siendo poco probable que llegue en algún momento a este valor, teniendo en cuenta los elementos conectados a este), 800 mA. Dentro de estos 800 mA también habría que cuantificar la corriente consumida por el operacional, que para este cálculo, despreciamos.

$$Vin = 16,97V$$

$$Vs = 12V$$

La potencia disipada será de por lo tanto: 3,98 W.

Existe otra fórmula [15], la cual nos permite calcular el coeficiente de disipación de calor del disipador en sí, en Cº/W. Serán adecuados, por lo tanto, aquellos que posean un coeficiente igual o menor al devuelto por el cálculo.

$$\text{C/W} = (\text{MaxRunningTemp - AmbientTemp})/\text{Potencia}$$

Considerando que queremos que el integrado en funcionamiento nunca superaría la temperatura de 60ºC y que la temperatura ambiental sería de 25ºC; el coeficiente resultante sería de: 8,79 ºC/W.
Teniendo en cuenta que usaremos el LM7812 con encapsulado de tipo TO-220, hemos seleccionado un disipador de este valor calculado, y que se adapta a nuestro montaje.

Los reguladores de tensión se dotarán de diodos de protección [25], para evitar que la corriente en una situación de corte de alimentación circule de forma inversa en el regulador, estropeándolo. Tras este, se colocará una segunda etapa de filtrado. La primera, se situará tras la rectificación de onda completa efectuada por el puente de diodos 7.

El condensador tras el puente rectificador ha de resistir la tensión de pico del transformador. Uno electrolítico de 25 voltios sería más que adecuado. Además, este sería el de mayor capacidad, ya que en este punto es en el que se debe reducir un mayor rizado.

Su capacidad es calculada en función de la siguiente fórmula [20]:

$$C = \frac{i_{max} * T}{Vr * 2}$$

Siendo Vr el rizado en voltios requerido, (16,97-14,60 V), T el período de la tensión, (1/50Hz)

e i_{max} la corriente máxima consumida (0,8 A). Colocando este valor de corriente máximo, nos aumenta la capacidad de dicho condensador, por lo que para evitar picos de corriente elevados en el arranque, reduciremos el valor de este hasta los 2200 uF.

El condensador colocado después del regulador será de menor capacidad. Hará la función de eliminar el rizado remanente tras el integrado [25]. Dicho integrado ya posee un rechazo al rizado de 60 dB, por lo que la hoja de características recomienda un condensador de una capacidad inferior. Con el fin de estabilizar la alimentación del LM358, se ha decidido aumentar la capacidad de este hasta los 100 uF.

11.2.2. Obtención de los 7,5 V

Se ha decidido implementar un segundo regulador de tensión, para conseguir proporcionar el valor de tensión adecuado para alimentar la placa Arduino. Se ha elegido la tensión de 7,5 V; siendo el mejor compromiso para solventar la caída de tensión del propio regulador integrado en el microcontrolador y el posible sobrecalentamiento producido por un voltaje demasiado elevado . Este regulador incluido en la placa estabiliza la tensión de alimentación a un valor de 5 V de continua, siempre y cuando el voltaje sea superior a 7.

Para conseguir esta tensión, se ha contado con un integrado, el LM317, el cual, por medio de un juego de dos resistencias, permite a la salida, obtener el valor que se quiera, siempre que esté dentro de un rango entre 1,25 y 37 V [24], cuando fue alimentado entre 3 y 40 V de continua. Este circuito, mantiene siempre 1,25 V entre la patilla de ajuste y masa.

$$V_{out} = 1,25 * (1 + \frac{R2}{R1})$$

$$R1 = 240\Omega*$$

** Tal y como recomienda el fabricante*

$$R2 = 1200\Omega$$

$$V_{out} = 7,5V$$

Debido a que la reducción es de 12 V a 7,5; volviendo a aplicar la fórmula del apartado anterior vemos que la potencia disipada por el regulador es inferior. Debido a este hecho, mantendremos el diseño del disipador del anterior caso:

$$\text{Potencia} = (\text{Vin - Vs}) * \text{I}$$

$$Vin = 12V$$

$$Vs = 7,5V$$

$$I = 0,8A$$

La potencia disipada será de por lo tanto: 3,6 W.

Estas dos etapas de fuente lineal son representadas en el plano, 7. Se ha comprobado la efectividad del montaje haciendo sus respectivas simulaciones en Pspice, dando lugar los resultados representados en el siguiente plano: 6.

11.3. Cálculos de las constantes del regulador

Como ya se ha mencionado, por medio del método de ejecución externa, se pudieron capturar las imágenes de la ejecución del bucle de relé. Guardando éstas en formato ".fig", se puede analizar a posteriori con precisión los datos obtenidos.
Los dos datos que se obtendrán de la captura será el período de oscilación Tc y el parámetro "a". A partir de una serie de iteraciones, se seleccionan 4 valores válidos y se realiza una media aritmética de los valores seleccionados. El parámetro Tc, irá definido en segundos.

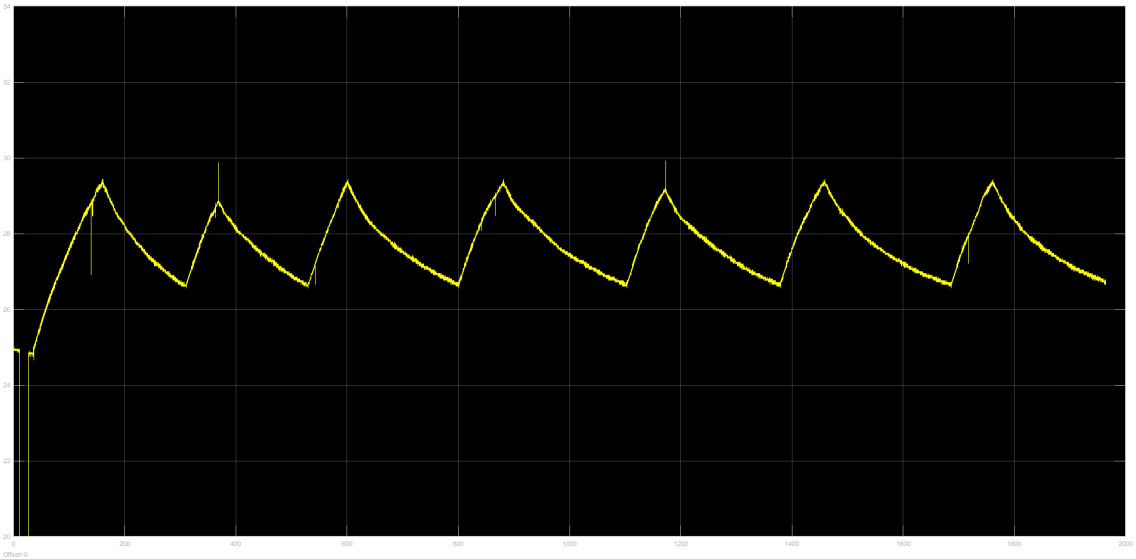


Figura 11.1 – Captura de la salida del relé

Éstos fueron calculados:

Cálculo de Tc y KC					Media
Tc	299,4	272,4	307	280,1	**289,725**
a	2,88	2,88	2,88	2,83	**2,8675**

Tabla 11.1 – Cálculo de Kc y Tc

A continuación se adjuntarán dos imágenes provenientes de los apuntes de clase, los cuales nos servirán útiles para la explicación teórica de términos:

El término feedback hace referencia en que en este tipo de diseño existe realimentación, el sistema consulta el progreso del valor de la salida para ir haciendo, en función de la evaluación de este, la conmutación. Esto se debe a que diseñar un sistema oscilante en cadena abierta resulta muy complejo de implementar.

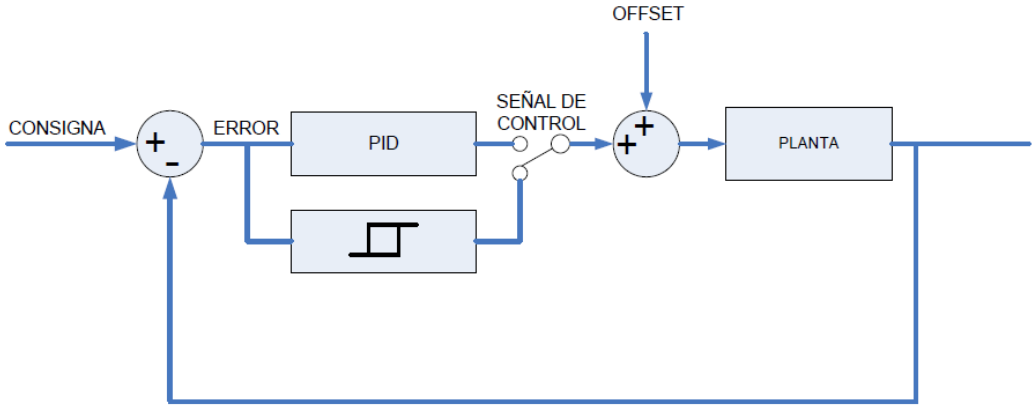


Figura 11.2 – Esquema de la implementación del sistema relé-feedback

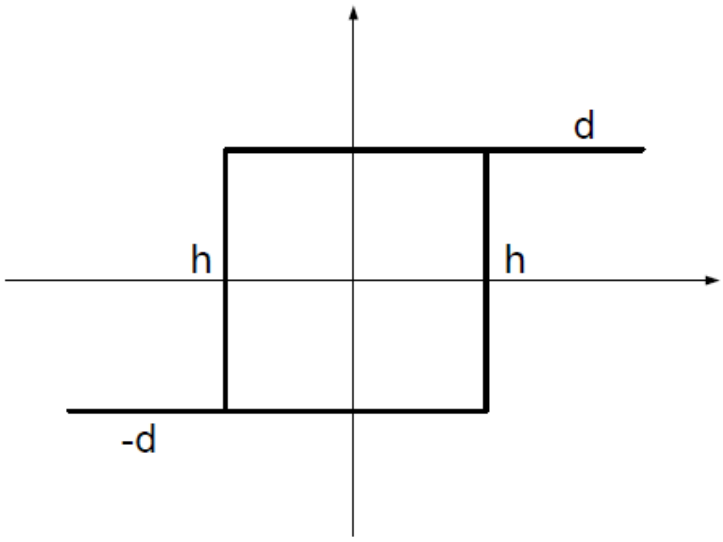


Figura 11.3 – Representación de los parámetros propios de la histéresis

En esta segunda captura, aparecen representados los parámetros d y h, que dependiendo de su elección, variará la histéresis.
El relé funcionará con un nivel potencia, el cual dependiendo del valor de la salida, conmutará correspondientemente. El valor máximo será de 100 y el mínimo, de 0. El valor medio será de 50, lo que definirá el parámetro “d” de “Kc”.
El ancho de la ventana de histéresis será de 2,8; la salida subirá 1,4 por encima del valor definido de 28 y el mismo valor por debajo para el límite inferior; siendo por lo tanto el parámetros “h” igual a 1,4.

Aunque el nivel de PWM real que haya que escribir sobre la placa sea de 255, (rango de 0 a 255) en estas dos etapas, tanto el relé como la saturación del PID estarán fijadas de 0 a 100, para facilitar la compatibilidad de datos entre sistemas, (el montaje antiguo de la planta estaba dimensionado entre 0 y 100). Por lo tanto, se hará un reajuste multiplicando por una ganancia de 2,5 en la última etapa del programa.

$$Tc = 289,73s$$

$$Kc = \frac{4 * d}{\sqrt{h^2 - a^2}} = 25,44$$

$$K = 0,6 * Kc = 15,26$$

$$Ti = 0,5 * Tc = 144,86$$

$$Td = 0,125 * Tc = 36,22$$

$$Kp = K = 15,26$$

$$Ki = Kp/Ti = 0,11$$

$$Kd = Kp * Td = 552,78$$

11.4. Cálculos en los circuitos impresos

El ancho y profundidad de pista son parámetros que se tienen que tener en cuenta a la hora de diseñar un circuito. Vendrán dimensionados por la temperatura máxima disipada y corriente máxima que circule por ellas. Se ha elegido KiCAD como software para el diseño. Por lo tanto, la profundidad de pista irá parametrizada según el valor recomendado por dicho programa, 1,6 mm. Para el ancho de pista, se han efectuado dos decisiones. Para la etapa de alimentación, se ha definido un ancho de 1,2 mm; algo superior al valor dado por fuentes consultadas [17] para el rango de corriente elegido y que deja un margen efectivo más que necesario. Para las pistas de menor corriente, (de aproximadamente 20 mA), se ha dejado el ancho por defecto de 0,25 mm.

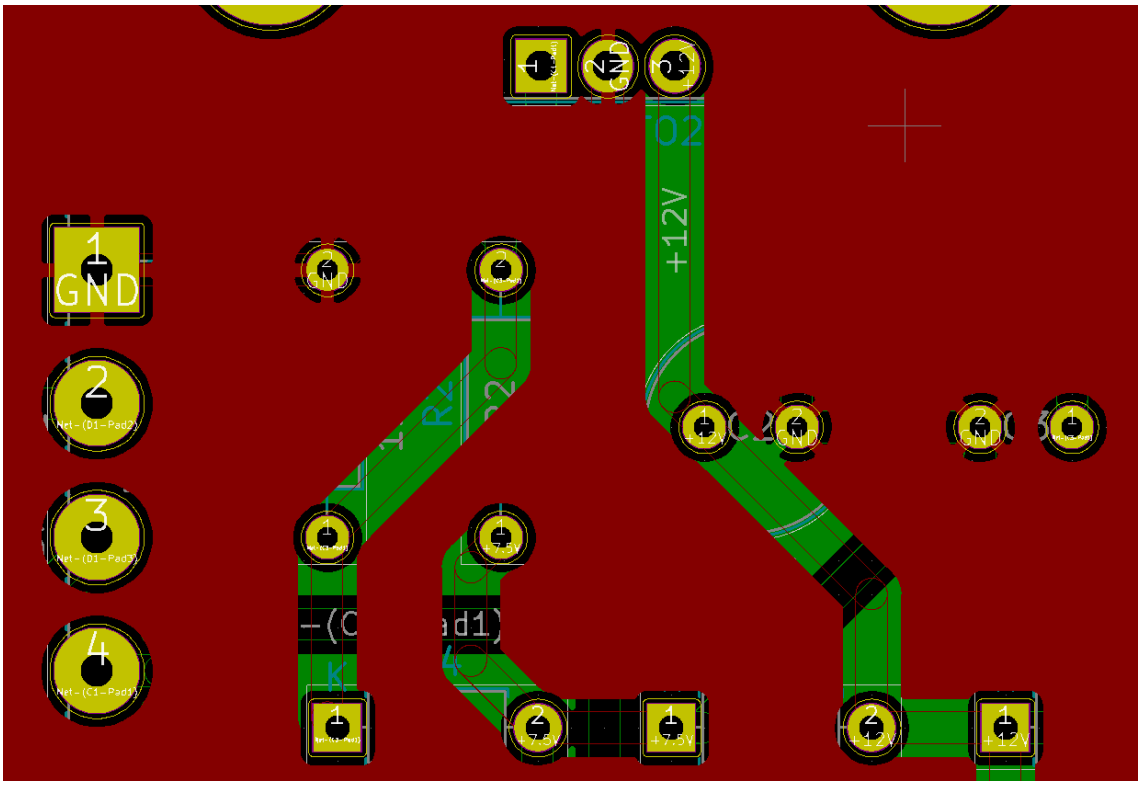


Figura 11.4 – Detalle del circuito impreso

11.5. Cálculo del filtro paso bajo

Como ya hemos visto, la frecuencia de generación de PWM se ve afectada por el tipo de programa que se ejecute sobre el Arduino 8.2.2. Y este PWM, ha de ser convertido a una tensión continua para que pueda ser aprovechada por el sistema a controlar. Para dimensionar los componentes pasivos del filtro, se tomará el caso en el cual la frecuencia de dicha señal mencionada sea del menor valor posible. Aprovechando que los tres principales programas de regulación poseían una frecuencia de ejecución de aproximadamente 36 Hz, se buscará una frecuencia de corte inferior a 1 Hz para que se pueda realizar un filtrado correcto, quedándose solamente con el armónico de continua.

$$F_c = \frac{1}{2 * \pi * R * C} \approx \textit{1 Hz}$$

Con un valor de la resistencia de 22 $k\Omega$ y con un condensador de 10 uF, se consigue, aplicando la fórmula, una frecuencia de corte de aproximadamente 0,8 Hz.

Para la frecuencia de 500 Hz de PWM procedente del filtro digital, se han recalculado los valores de los elementos pasivos. En este caso, con una resistencia de 10 $k\Omega$ y un condensador de un microfaradio, se conseguirá una frecuencia de corte de aproximadamente 16 Hz, suficiente para conseguir la tensión continua deseada.

12 GUÍA DE USUARIO

12.1. Instalación del Simulink Support for Arduino

Lo primero que será necesario para poder seguir correctamente esta guía es tener instalado como mínimo la versión R2017a de MATLAB. En este caso, se hará desde la R2017b. La razón por la cual versiones anteriores no posean el mismo método de instalación es debido a un problema con el instalador del paquete de librerías. Este problema está solucionado por los propios trabajadores de MATLAB los cuales proporcionaron un link a disposición de cualquiera: [12].
Además, todo lo que abarca este trabajo, será compatible tanto con Windows, Linux o MacOS. El único requisito será poseer la mencionada versión de MATLAB y Simulink instalado. Se realizará todo el tutorial desde la versión de Windows 10 Home, que es de la que se dispone.

Otro requisito, será la creación de una cuenta de MathWorks, link: [13]. El proceso es muy simple, y es gratuito, y con ella no solo se tendrá acceso a los paquetes que se necesitarán para esta tarea, sino que se podrá acceder a tutoriales y foros especializados mantenidos por la propia empresa donde se solucionan toda serie de problemas que puedan surgir a la hora de implementar un programa o resolver problemas a la hora de instalar cualquier archivo.

Lo primero que se hará, será instalar los paquetes necesarios para la vinculación del programa con la placa Arduino. Para ello, se navegará a la parte superior del panel principal de MATLAB.

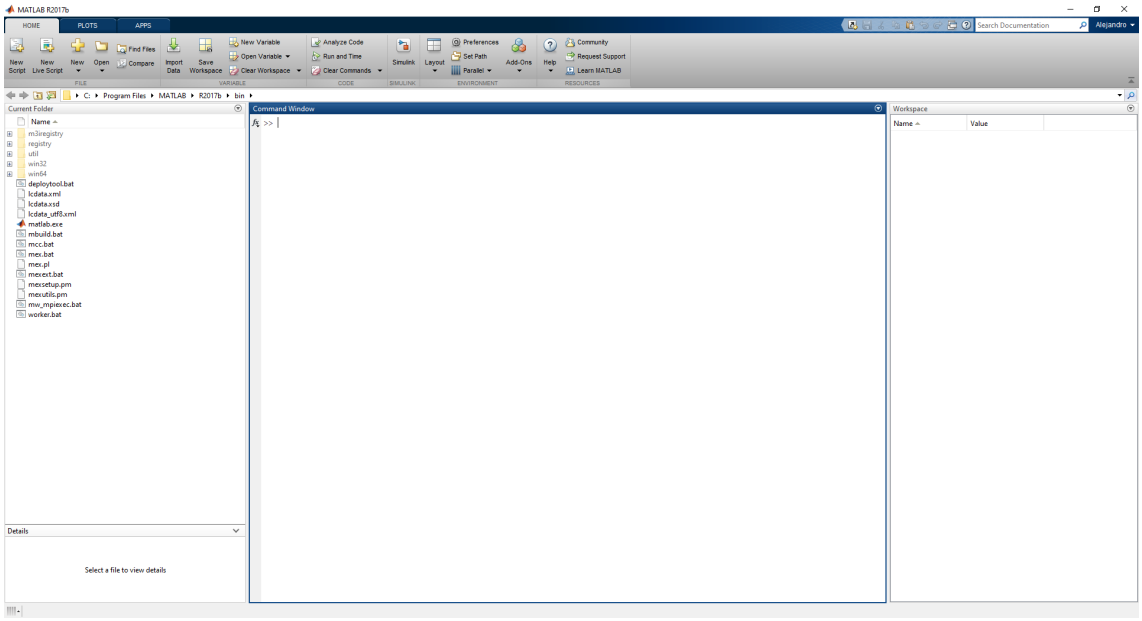


Figura 12.1 – Panel frontal de MATLAB

En la barra superior, hay un botón, llamado “Add-Ons”, el cual pulsaremos.

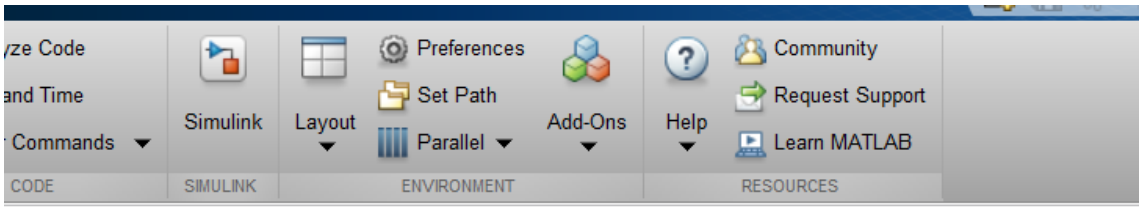


Figura 12.2 – Posición del botón Add-Ons

Se abrirá un menú y dentro de las elecciones disponibles habrá que elegir la opción “Get Hardware Support Packages”. Esto abrirá una ventana en la que aparecen listadas las diferentes librerías disponibles para descarga, con una breve descripción y junto con imágenes ilustrativas. En el menú de búsqueda, habrá que teclear “Simulink Support for Arduino”.

Esto abrirá la siguiente ventana:

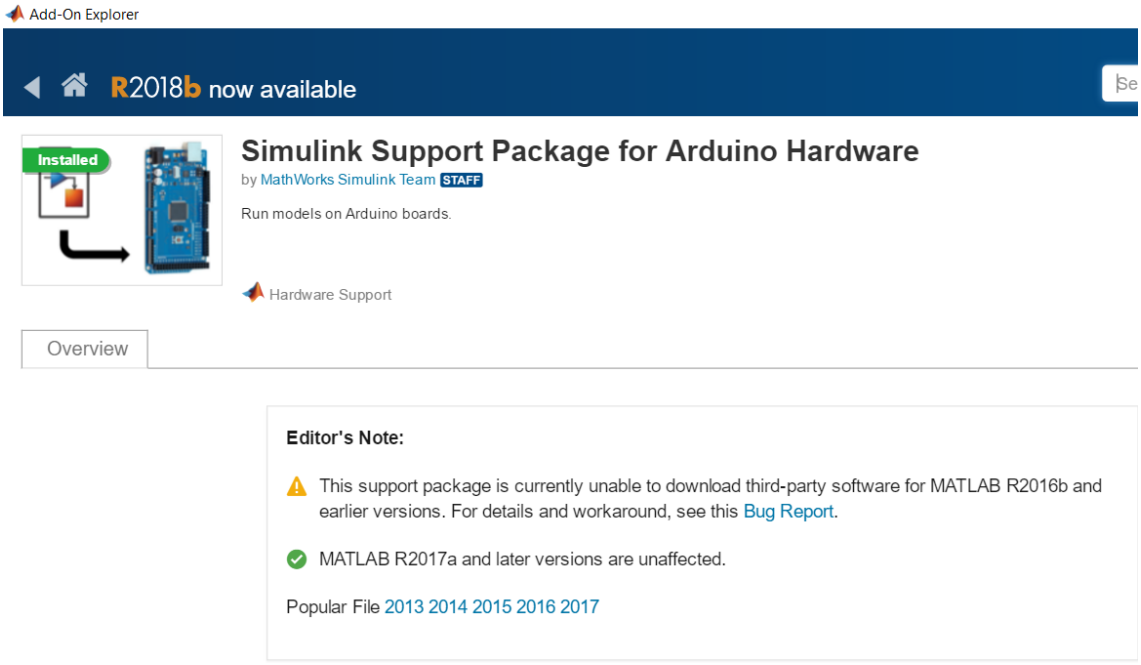


Figura 12.3 – Descripción de la librería dentro del “Add-Ons Explorer”

En esta captura, ya se puede apreciar la advertencia relacionada con las versiones anticuadas de MATLAB y su problema en la instalación si no se efectúan los pasos dictados. En la captura posterior, se puede ver que el programa autodetecta si los complementos necesarios están instalados, (en este caso, Simulink).

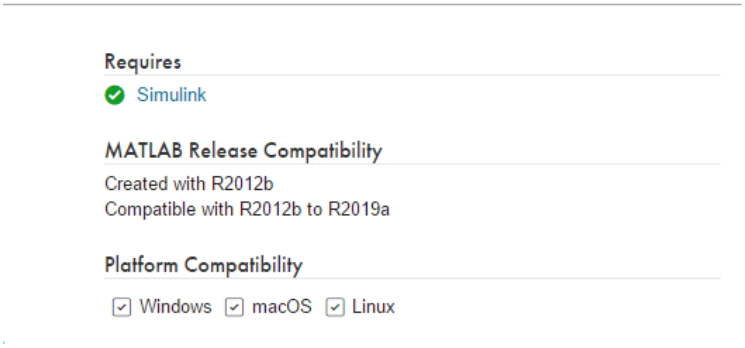


Figura 12.4 – Autocomprobación de Requisitos

El último paso dentro de esta ventana, será pulsar “Install”, esto le abrirá una ventana de instalación, en la cual todo ha de quedar configurado por defecto, esperando a que termine y concediendo permiso a la copia de todos los paquetes. Esta utilidad también instalará los drivers necesarios para que la comunicación entre el PC y Arduino sea correcta.
Una vez efectuada la descarga e instalación de paquetes, se pasará a evaluar si ésta se ha efectuado correctamente. Para ello se abrirá una ventana en la que se ofrece la posibilidad de configurar y probar la conexión de nuestra placa Arduino. (Figura 12.6).

12.2. Conexión del Arduino al PC

Para conectar la placa Arduino con nuestro PC, hará falta un cable USB 2.0, con interfaz de conexión tipo A macho en un extremo y una tipo B macho en el otro. El cable incluido normalmente con el propio Arduino, cumple esta especificación.

Figura 12.5 – Cable USB 2.0 tipo A a tipo B

Una vez conectado el Arduino al cable y este al ordenador, se podrá probar la conexión y configuración propuesta por Simulink. Se elegirá una comunicación por medio de USB. (Figura 12.7). Por seguridad, se desconectarán otros aparatos USB del PC que no sean el Arduino en sí. Si se poseen varios Arduino, solo uno será conectado. Se seleccionará el modelo de placa conectado y se dejará la configuración posterior por defecto.

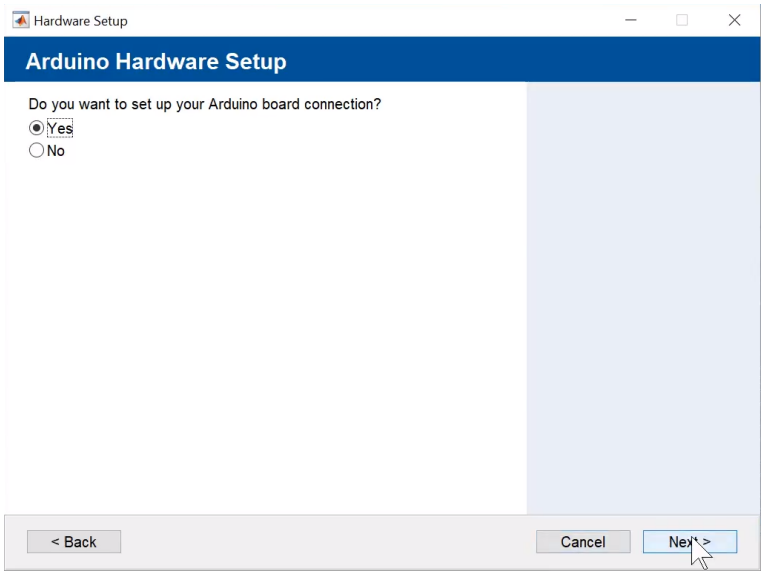


Figura 12.6 – Solicitud de testeo de la conexión

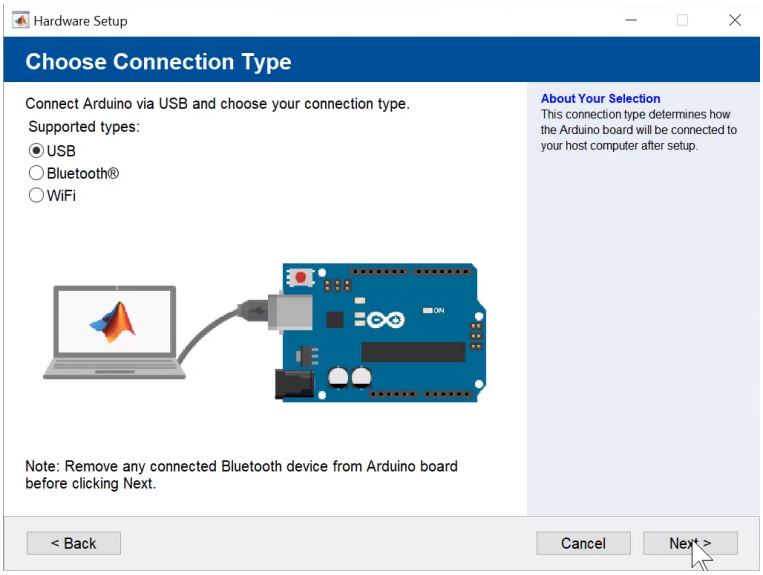


Figura 12.7 – Seleccionar modo de conexión USB

Si el proceso de instalación y detección de la placa se realizó correctamente, al conectar la placa con el PC, en la “Command Window” de MATLAB aparecerá el nombre de la placa conectado en ese momento.
Si se conectan varias placas simultáneamente, ambas aparecerán listadas, una a continuación de otra. (Figura 12.8).

```
Arduino Mega 2560 detected.
This device is ready for use with Simulink Support Package for Arduino Ha
To use this device with MATLAB, install MATLAB Support Package for Arduin

Arduino Uno detected.
This device is ready for use with Simulink Support Package for Arduino Ha
To use this device with MATLAB, install MATLAB Support Package for Arduin
```

Figura 12.8 – Conexión satisfactoria

12.3. Contenido de la librería Simulink Support for Arduino

Para crear nuestro primer modelo con Simulink, lo único que tendremos que hacer es escribir en la Command Window de MATLAB la palabra “simulink”. Esto abrirá la siguiente pantalla, 12.9, y en ella, habrá que clickar, crear un proyecto vacío, “Blank Project”, esto nos abrirá el modelo sobre el cual podremos colocar los bloques-función propios del programa 12.10. Abriendo las librerías instaladas, y navegando por orden alfabético a la parte inferior, podremos ver la del Arduino.
Haciendo click sobre ella y abriendo la subsección “Common”, encontraremos las funciones que se utilizarán principalmente en este trabajo 12.11.

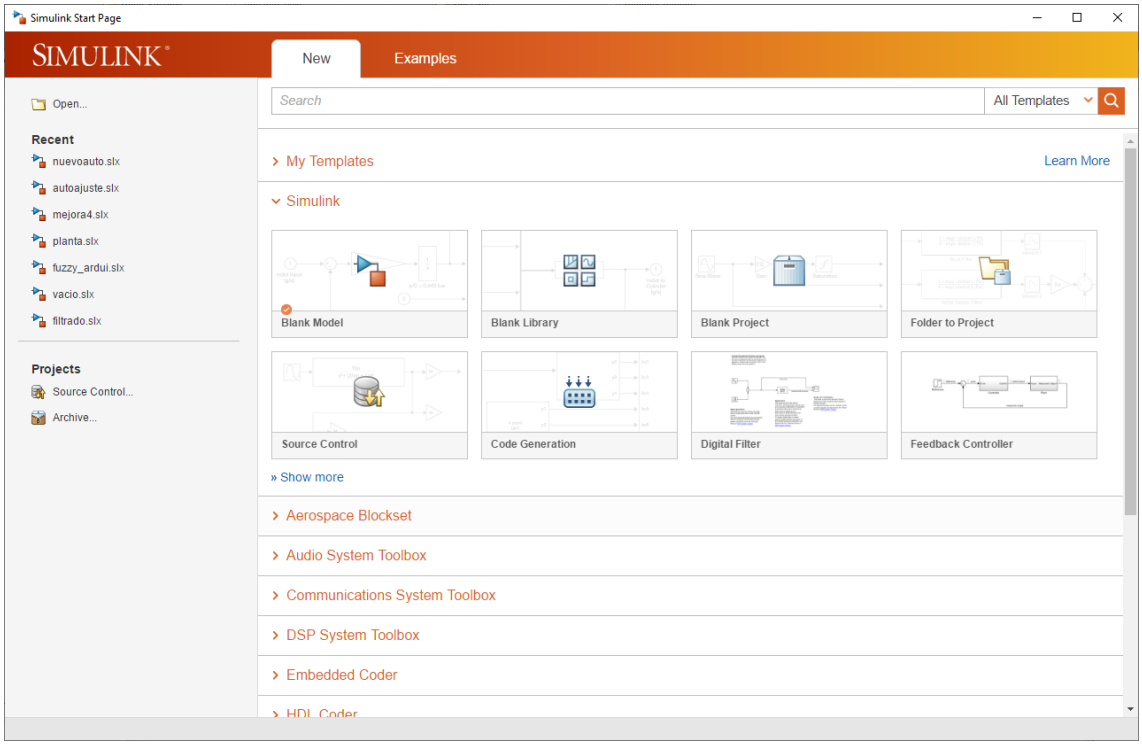


Figura 12.9 – Página principal de la extensión Simulink

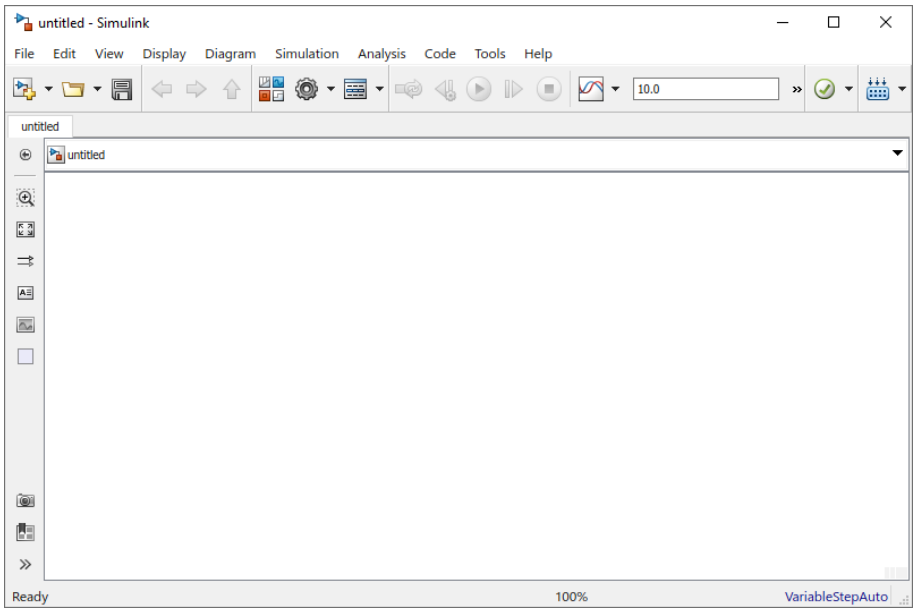


Figura 12.10 – Modelo vacío, sobre el cual se añadirán los bloques

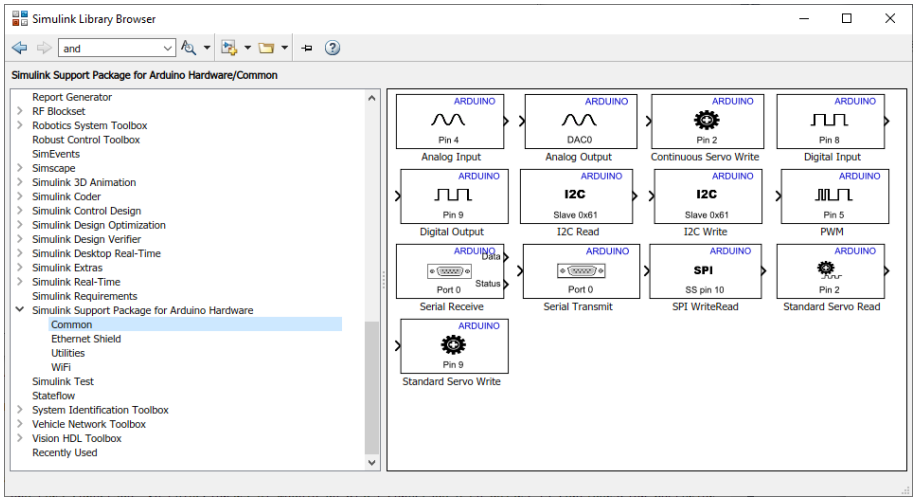


Figura 12.11 – Librería del Arduino instalada

Haciendo doble click sobre esta librería, se puede apreciar un gran número de bloques. Se empezará definiendo el Analog Input. Estos, obviamente vinculados a una entrada analógica concreta del Arduino, traduce un voltaje de rango 0-5V a un valor digital escalado entre 0-1023. Este rango digital será tenido en cuenta a la hora de utilizar la lectura del sensor dentro del montaje.

El segundo bloque será Digital Input, que al contrario del anterior, lo que detectará será un nivel bajo o alto de tensión, (0 o 5V) y consecuentemente, lo traducirá en un valor booleano, (0 o 1 respectivamente). Se usará, para leer, por ejemplo, valores de conmutadores, o pulsadores.

El bloque Digital Output, hará la operación contraria al anterior descrito. Traducirá un valor, 0 o 1, fijado por el programa sobre un pin determinado del Arduino. Se usará para toda serie de utilidades, una de ellas, encender un Led.

El bloque Analog Output, que sería la opción lógica de trasladar sobre un pin un valor analógico concreto de tensión, por ejemplo, 2.2V, no es compatible con el Arduino MEGA ni UNO. La opción que se utilizará será el bloque PWM. Este bloque generará a partir de un rango de entrada de una señal escalada entre 0-255, una señal de PWM de 500 Hz o bien 1KHz, dependiendo del pin elegido. Debido a la forma de esta señal generada para que pueda ser utilizada correctamente, tendrá que ser filtrada con un paso bajo, condensador en serie, resistencia en paralelo.
Los bloques de entrada, vienen con una característica configurable, el tiempo de muestreo.

12.4. Volcado del programa sobre la placa Arduino

Una vez se ha definido toda la documentación anterior, faltará el último paso, trasladar el código a la placa Arduino. Esto se realizará a través de la interfaz USB del microcontrolador, estando este conectado al ordenador. Una vez que la configuración inicial efectuada en la instalación del controlador ya se ha hecho, se da la autodetección de la placa por parte del programa por supuesto. El programa incluye tal utilidad integrada, aunque aun así, se dedicará un apartado para ello 12.4.2.

12.4.1. Selección del modo de compilación

Simulink, al igual que otros softwares existentes en el mercado, incluyen la posibilidad de elegir cuál debe ser la filosofía de ejecución del programa dentro de la placa. Esto se hace aplicando un método diferente a la hora de realizar la compilación del código. Incluye dos alternativas, o bien que se optimice el tiempo de compilación del programa a costa de pérdidas de rendimiento posteriores; cosa que no interesará, ya que las compilaciones se realizan de forma bastante rápida. Debido a este motivo, ni se ha valorado este método.

El procedimiento elegido, será el de producir ejecuciones lo más rápido posibles en el micro, lo que producirá un código lo más afín y optimizado posible al Arduino en cuestión.

12.4.2. Configuración de la placa

En este submenú, se podrá elegir el tipo de dispositivo al que se puede instalar el programa. Aparecerán listados todos los aparatos compatibles. Es decir, figurarán todos los dispositivos compatibles con Simulink que poseen un controlador instalado en el ordenador 12.12 y sobre el cual se puede destinar el código.
Entre ellos, si no se tienen más librerías instaladas, aparecerán todos los modelos de Arduino. De esta lista, se seleccionará el dispositivo que se quiera utilizar en cada caso.

En teoría, por defecto, existe un método de autodetección de dispositivos conectados al USB, así que por defecto no habría por qué especificar a qué puerto COM de la placa se ha linkado la placa. Si tras las compilación, esta da un error relacionado con la mencionada detección del Arduino, se puede especificar manualmente.

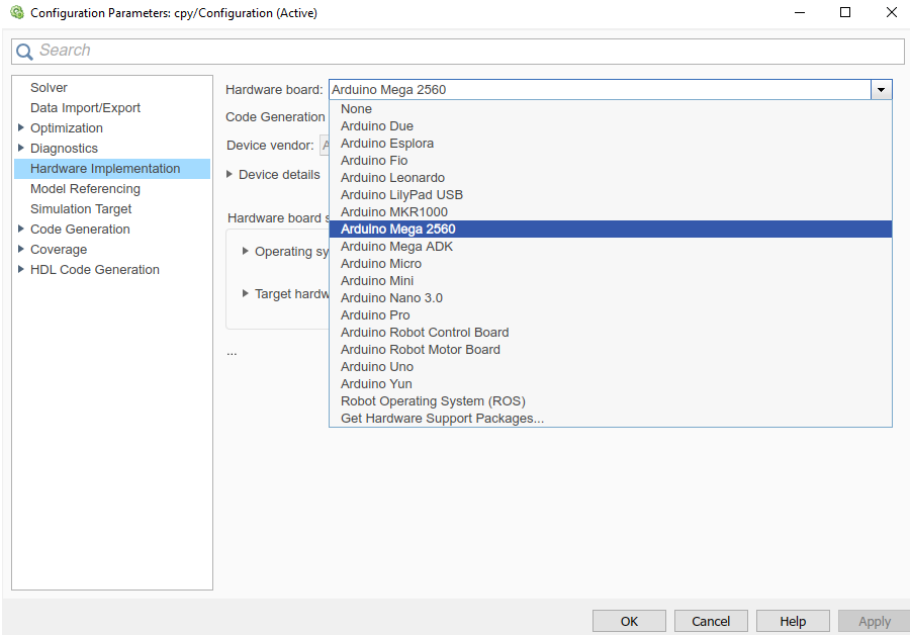


Figura 12.12 – Selección entre los diferentes dispositivos compatibles

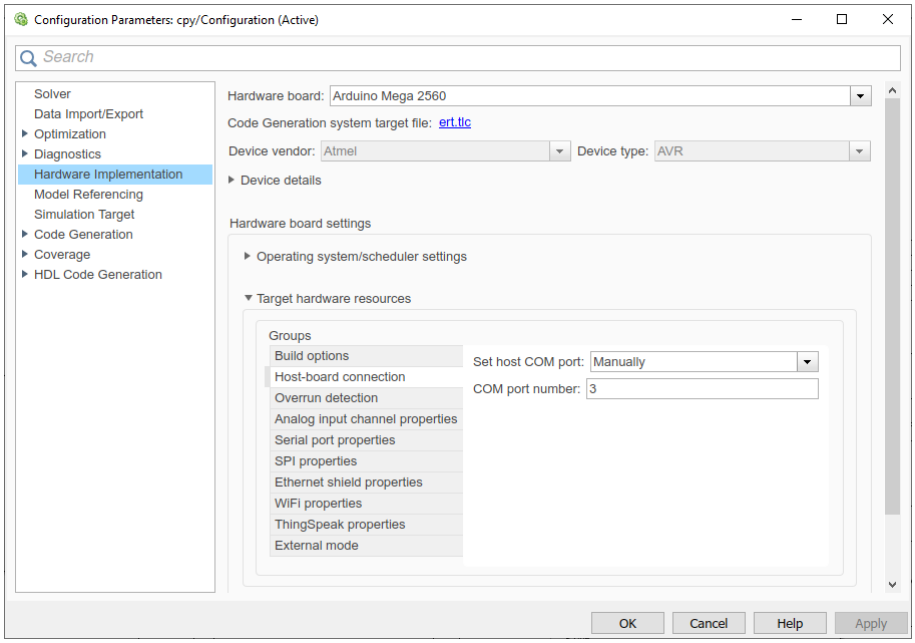


Figura 12.13 – Selección manual del puerto COM

12.5. Modo de ejecución externa

La razón por la que se eligió la opción de utilizar una placa Arduino Mega es porque esta además de poseer mayor número de puertos de comunicación, incluye, por defecto, una opción muy interesante, el modo de ejecución externa, compatible con todas las versiones de MATLAB. A continuación, se adjuntará una captura del propio manual, donde se especifican las diferentes compatibilidades de placas Arduino con el software en cuestión. Como se puede ver, el Arduino UNO es sólo soportado de la versión R2016b en adelante.

Placa Arduino	Soporte para Shield	Ajuste interactivo y monitorización	Comentarios
Arduino Due*	S	S	El canal CAN no está soportado actualmente.
Arduino Uno*	S	S	Ajuste interactivo soportado a partir de la versión R2016b. Puede registrar una señal a 1 kHz o hasta 6 señales a una velocidad de 5 ms.
Arduino Leonardo*	S	S	
Arduino Mega 2560*	S	S	Registre una señal a 1 kHz o hasta 6 señales a una velocidad de 5 ms en la versión R2016b.

Figura 12.14 – Compatibilidad del hardware

Por medio de esta función, y poniendo tiempo de ejecución en infinito, puedo, con el sistema funcionando, ver tanto en el programa, (en los Scopes y en los Displays), como en el osciloscopio los cambios producidos en la salida, tras cambios de diseño en el propio regulador. Es decir, el programa se ejecutará de forma normal y a su vez estos resultados serán mostrados por pantalla en la ejecución de Simulink. La escala de tiempos es correcta y permitirá que las medidas sean certeras y se puedan utilizar para cómputo.

Además, este modo soporta otra gran característica, que es que parámetros determinados como ganancias pueden ser modificados en tiempo real sin necesidad de recompilar ni volcar el programa de nuevo, sino que esta variación o cambio ya es representado al momento. Esto resulta de gran utilidad y es perfecto para el testeo, calibración o mejora del propio regulador PID.

Figura 12.15 – Control de las tensiones en tiempo real

Para acceder a este modo, habrá que fijarse en la parte superior del entorno de Simulink. En esta zona, se podrá modificar tanto la mencionada duración de la simulación, que por comodidad, coloca en ese valor, infinito, “inf”, pero que puede especificarse cualquier otro. Irá definido en segundos.

También cabe destacar, que gracias a este modo, se pueden localizar y solucionar posibles

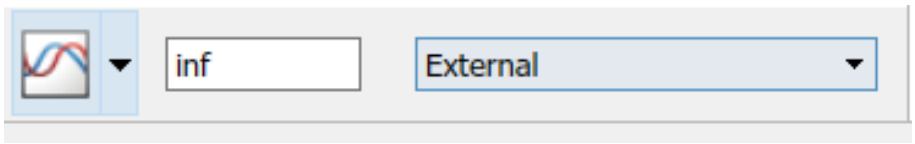


Figura 12.16 – Controles mencionados

problemas, como fallos de lectura de un sensor, que de otra forma serían imperceptibles y pasados por alto.

12.6. Programa de ejemplo: Regulador Proporcional

Una vez hecha la primera parte de configuración, como aparece ilustrado en las figuras 12.1, 12.2 y tras familiarizarse un poco más con el entorno y su hardware, se procederá a realizar el primer programa. Antes de poder llegar a diseñar un regulador PID, primero se realizará una serie de análisis para conocer con detalle características importantes del funcionamiento tanto del micro, como del programa en sí en un entorno real.

Lo primero que se diseñará será un regulador proporcional. Como interesa conocer la respuesta del microcontrolador en condiciones ideales, este irá sin realimentación y con ganancia unitaria. Introduciendo una entrada ya conocida, se quiere obtener esta misma a la salida y así observar sus especificaciones, además de practicar la lectura y escritura de tensiones.

Para las entradas del sistema, se utilizarán las entradas Analógicas o Analog Inputs de la placa Arduino, en concreto las 0 y 1. Debido a que estas están conectadas a un A/D de 10 bits de resolución y de un rango de 0 a 5V, traducirán la tensión conectada a un valor digital de 0 a 1023.

Por tanto, es necesario implementar un bloque funcional que me vuelva a traducir el valor digital a analógico. En el control, se trabajará con valores de tensión para así, llevar una mejor supervisión del proceso.

La tensión de salida se generará a través del pin de PWM.
Lo primero que se hará, es alimentar la entrada analógica 1 con un valor de tensión que podrá ser regulado dentro del margen 0-5V. Esta será la entrada del sistema, su consigna.

Para poder construir esto, únicamente se necesitará un potenciómetro, de por ejemplo, 10 $K\Omega$. Se conectaron sus dos extremos a GND y 5V respectivamente y el pin del selector a la entrada analógica 1 del Arduino.
La entrada analógica 0, que servirá en otros montajes como entrada de lectura de la tensión a la salida, a la que se puede conectar efectivamente un sensor, que necesitará estar acondicionado para que funcione en el rango anteriormente mencionado.
En este primer caso estará conectada a GND, por lo que la realimentación en verdad será nula. Por lo tanto, a la salida se tendrá el error directamente.

El propósito será comprobar si a la salida aparece un valor de tensión exactamente igual que la tensión de entrada, ya que la ganancia es unitaria.

Debido a que la señal de error tendrá valores analógicos de tensión entre 0 a 5 V, y la de

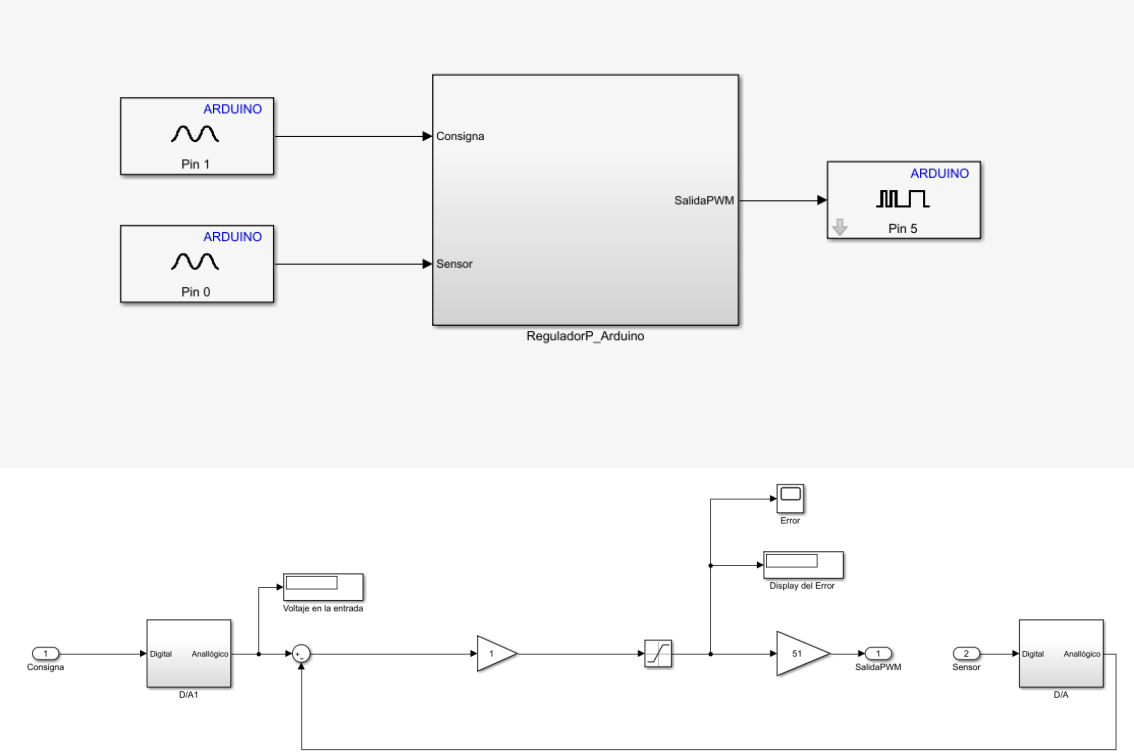


Figura 12.17 – Esquema del Regulador Proporcional

PWM varía entre 0-255, se transformará la señal de error (0-5V) en la de PWM por medio de una ganancia de 51, (255/5). Para evitar señales de error de más de 5V, innecesario en este caso, se ha colocado un bloque de saturación con los valores de 0 a 5.

Una vez el programa fue volcado, 12.4, se comprobará si se consigue una ejecución del código creado por Simulink en el microcontrolador de forma totalmente autónoma y sin depender de un PC o del programa MATLAB supervisando o haciendo los cálculos pertinentes.
Tras hacerse pruebas alimentando el Arduino sin utilizar el ordenador en el que se tiene instalado el programa mencionado, se puede comprobar que este puede actuar de regulador de forma totalmente autónoma e independiente.

12.7. Creación de bloques personalizados a partir de librerías de Arduino

12.7.1. Funciones definidas MATLAB

Consiste en la creación de un bloque el cual contiene una función de MATLAB prácticamente estándar. Esta posee un número de entradas definidas por el programador, y una única salida, junto con un algoritmo de ejecución. El propio programa es capaz de generar un código compatible con el hardware de Arduino. Se utilizará este recurso cuando se quiera simplificar una implementación por medio de bloques.

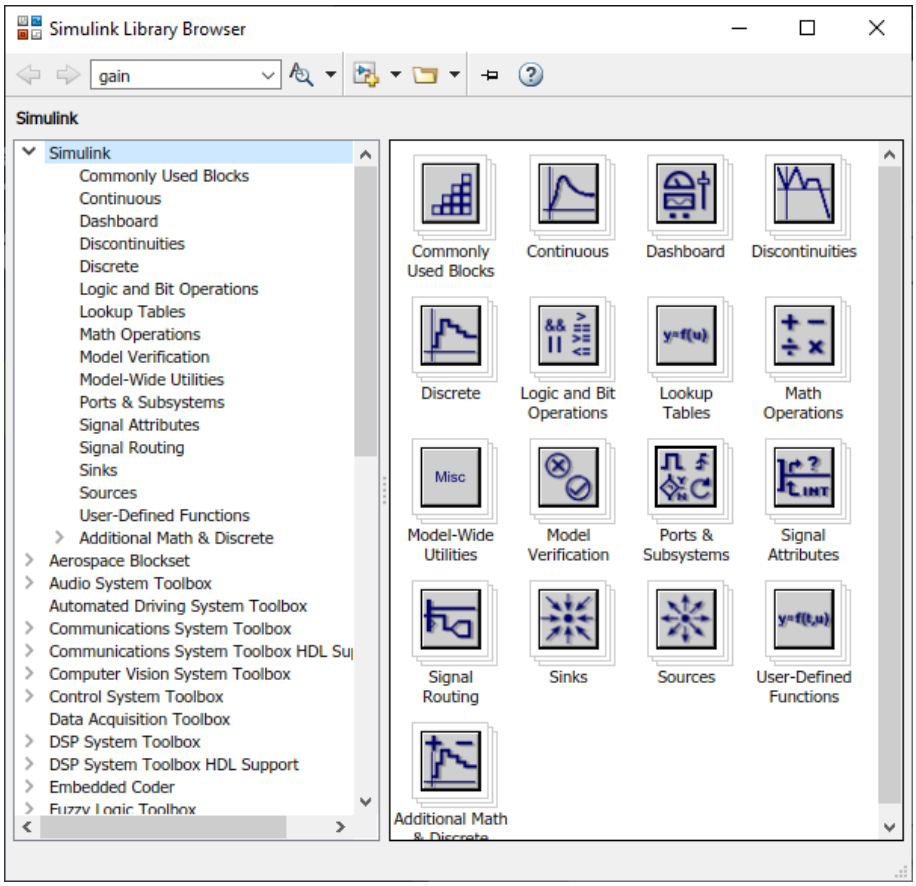


Figura 12.18 – Librerías predefinidas y preinstaladas

12.7.2. Funciones-S

Otra gran herramienta que posee Simulink es la posibilidad de compilar y utilizar funciones y librerías adaptadas al lenguaje de programación C y sus variedades. Gracias a esto, podremos diseñar bloques función a partir de las librerías propias del entorno de programación de Arduino, ya que no todos los bloques vienen incorporados en la librería de Arduino para Simulink. Además, estas líneas de código comparten sintaxis, palabras clave y estructura igual a la que se utilizaría al programar ese mismo bloque, con el compilador estándar del microcontrolador. Gracias a esta gran integración, podremos conectar bloques diseñados por por el usuario, con los bloques estándar de Simulink. Usando este método, adaptaremos y crearemos funciones para hacer funcionar y adaptar el teclado y la pantalla.

Para ello, con la ayuda de GitHub obtendremos las librerías de Arduino, libres para el uso de la comunidad, que hagan funcionar los componentes elegidos. Estas tendrán que ir incluidas en el “Path” del proyecto. Sacaremos ejemplos del código en C del teclado y la pantalla, de ejemplos ya hechos por usuarios en internet. Nosotros solo los adaptaremos a nuestras necesidades.
Si se hace doble click sobre el bloque función, este incorpora varias pestañas y funciones. Un botón de “Buid”, con el que compila. Para poder compilar las funciones en C, hace falta instalar un compilador dentro de MATLAB, para ello hay que viajar a “Add-ons” y añadir el siguiente pulsando sobre instalar, “MinGW-w64 Compiler” 12.19.
En la pestaña “Initialization”, hay que seleccionar el número de estados discretos, en este

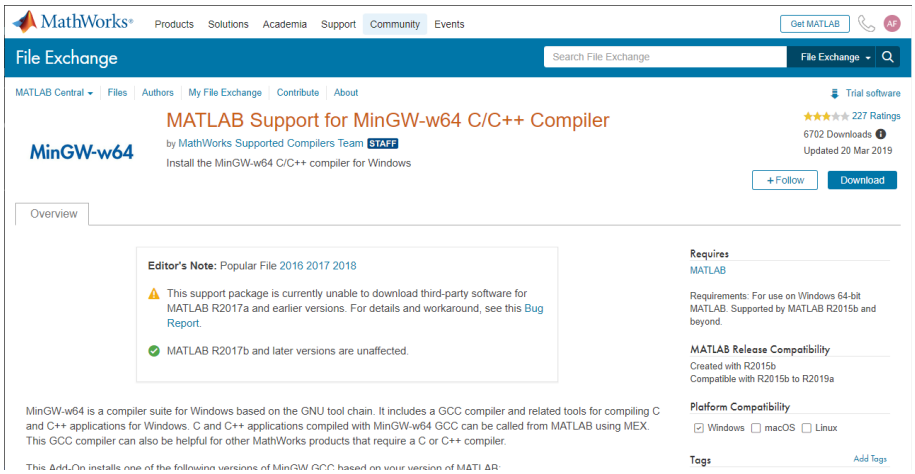


Figura 12.19 – Menú de descarga del compilador

caso, al solo existir uno, se escribirá un 1.

En “Data Properties”, se crean y se definen las señales, tanto entradas como salidas. En este caso solo se definirá una salida y 5 entradas.

Dentro de “Data Properties”, clickando en “Data Type atributes”, se puede seleccionar el número de bits de la señal, así como el tipo de estos otros parámetros que serán configurados según aplicación.

En la pestaña “Libreries”, donde se incluyen, por medio del #include las funciones que son necesarias para que el código se genere correctamente. Estas librerías externas y descargadas tendrán que ir dentro del fragmento definido para la función determinada a tal uso, tal y como aparece en el código adjuntado, así como las variables creadas por el propio usuario dentro de un #define. Cuando estas definiciones se terminen, el “if” se cerrará.

En la pestaña “Update”, dentro de su único, en este caso, estado discreto, irán definidas las ejecuciones que sólo se ejecutarán una única vez. Se utilizará esta función, para, en el caso de la pantalla LCD, poder mostrar un mensaje en el arranque, (como efecto visual). Para ello, se empleará una variable, que tras una comparación se vuelva a ejecutar. Esta variable, que es propia del estado discreto y que por defecto se ha de usar la reservada por el programa para esta acción, “xD”, utilizando así un número inferior de recursos, tal y como se especifica en la siguiente fuente [23].

Y por último, la pestaña “Outputs”, que definirá la acción que se ejecutará periódicamente, actualizando el valor de las variables definidas dentro de este bloque, con el valor calculado dentro del modelo de Simulink, representándolas en el modelo físico.

12.7.3. Creación del bloque de teclado

El primer paso que se requerirá es, obtener de GitHub las librerías necesarias para hacer funcionar este componente en cuestión. Estas son, “Key.h”, “Key.cpp”, “Keypad.cpp” y “Keypad.h”. Entre estas, harán consultas entre ellas, para acceder a determinadas características de funcionamiento. Una vez obtenidas, estas han de ser colocadas en en la misma carpeta en la que se encuentre el proyecto de Simulink. Si esto no se cumple, aparecerá un error en la compilación indicando la falta de dicho archivo. “Keypad.h”, exige que “Key.h” se encuentre dentro de una carpeta llamada “utility”. Por lo que esta se creará y se introducirá una copia extra de esta librería y su versión compilada, “.cpp” en su interior.

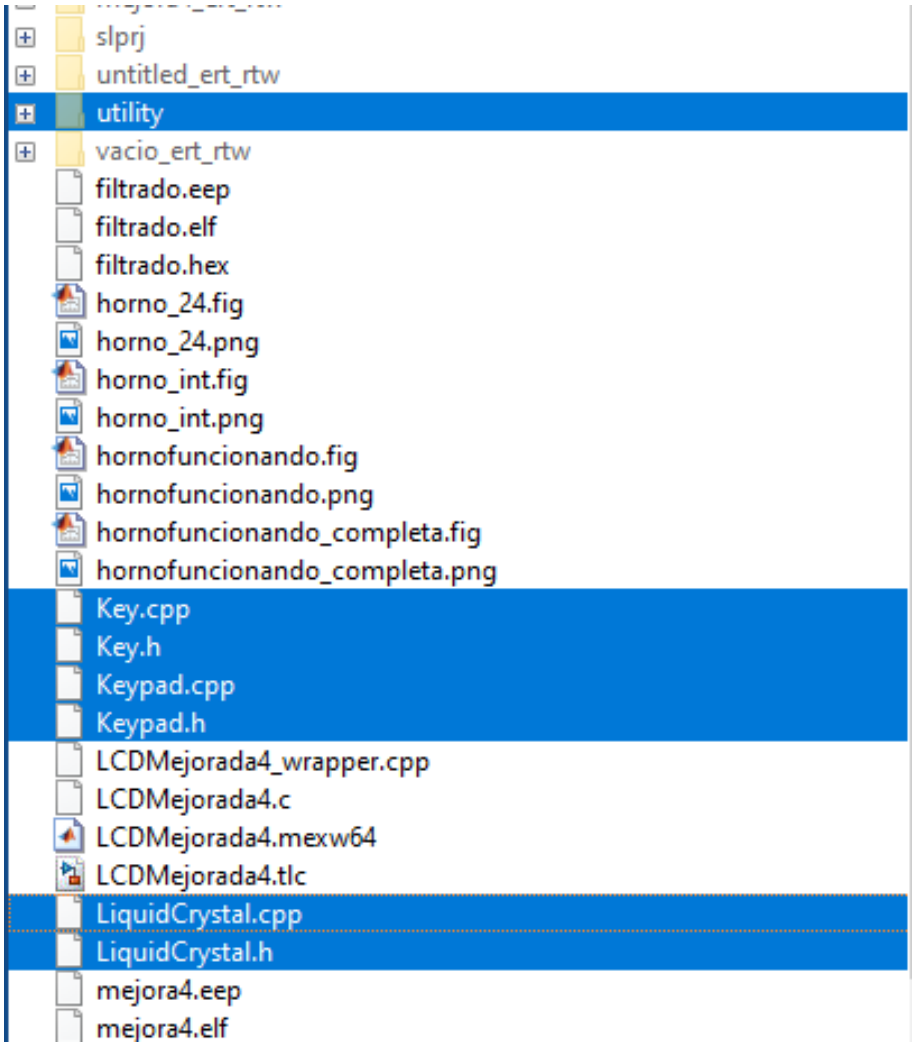


Figura 12.20 – Librerías necesarias que han de ser agregadas, las de la pantalla incluidas

En la primera página, se configurarán el número de estados discretos, para este caso y la mayoría de casos se puede dejar como uno, indistintamente.

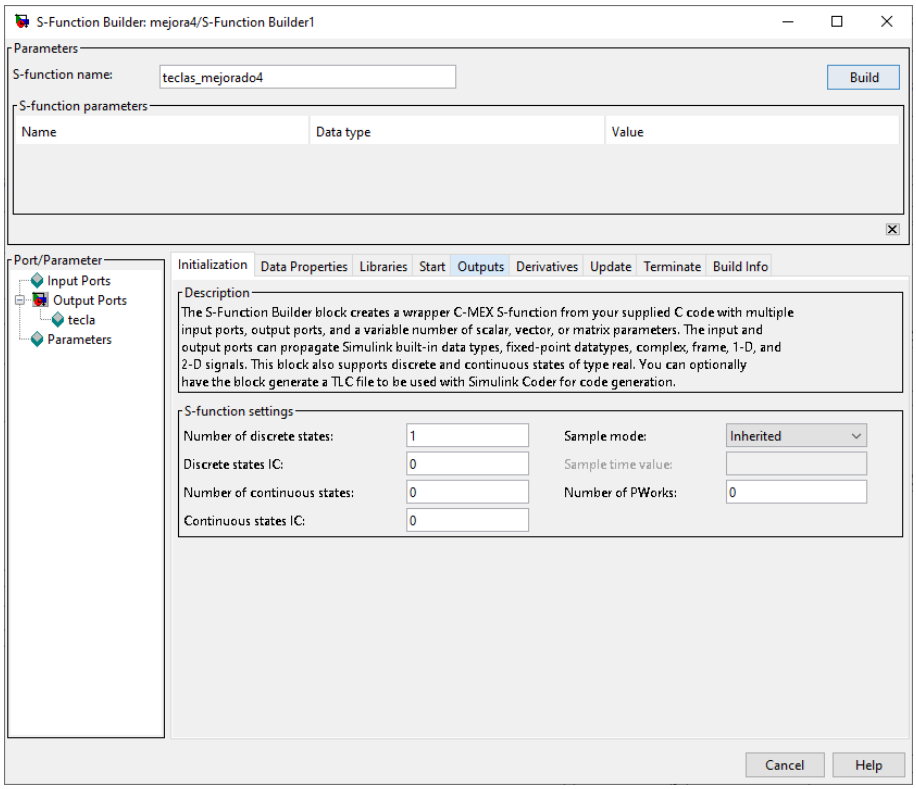


Figura 12.21 – Estados discretos

En la segunda pestaña, se creará la variable de salida del bloque, “tecla”. Debido a limitaciones en la comunicación entre valores creados en el interior del bloque y fuera de este, se decidió que esta variable contenga como valor únicamente un número entero. Será dentro del programa de Simulink, por medio de Sample and Hold y flancos de detección al pulsar las teclas de confirmación de datos, mediante las cuales, el número completo será guardado.

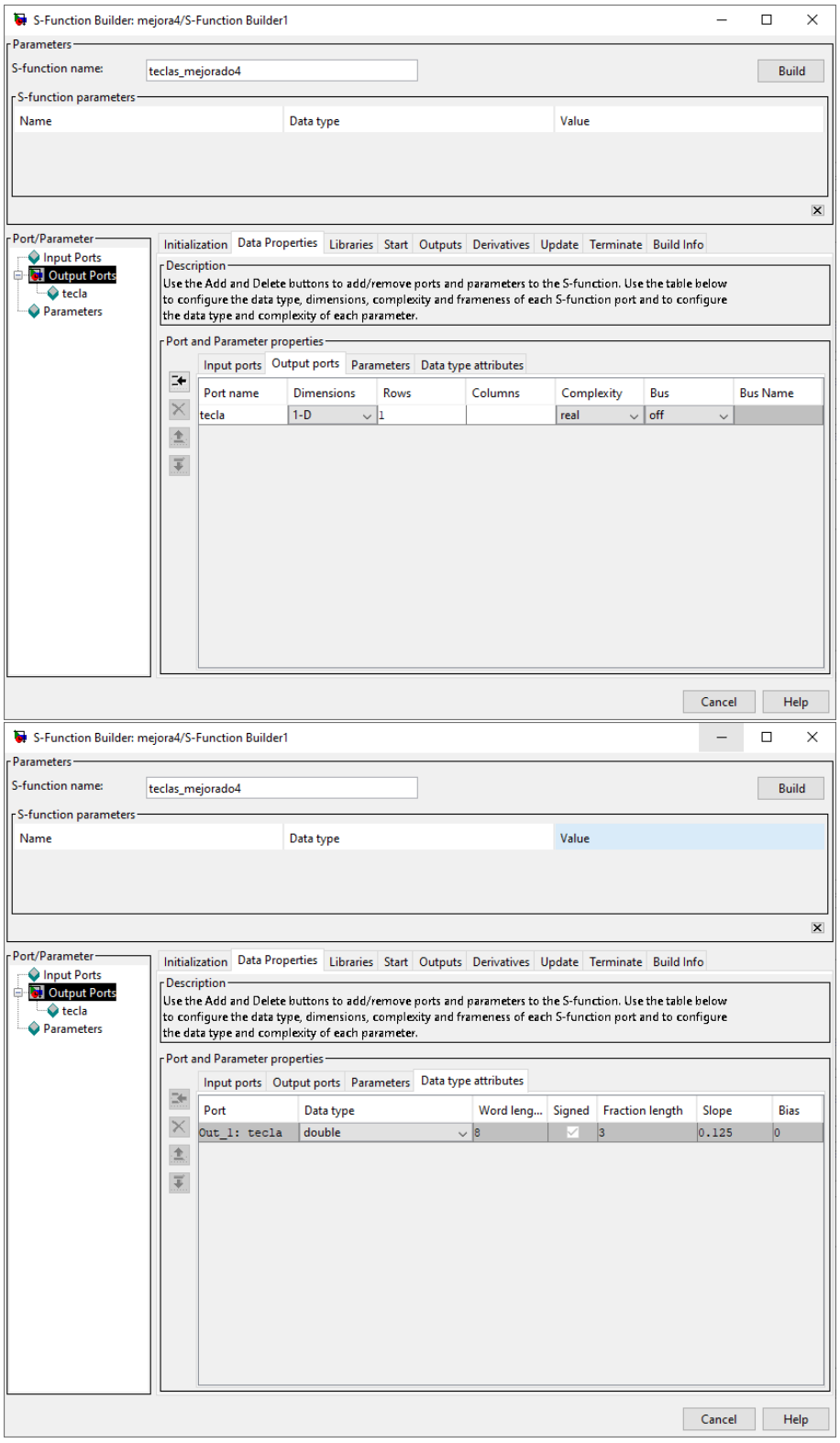


Figura 12.22 – Captura de la creación de la variable de salida

En el tercer apartado, es el dedicado a librerías. Ya que nos encontramos en un entorno de programación C, después de definir correctamente las librerías a utilizar, la creación de constantes, definición de pines del Arduino y el resto de cuestiones, como la definición del tipo de teclado, posición de teclas, etc. el código se redactaría de igual forma que dentro de una función programada con el IDE estándar. El código completo de estas secciones se puede observar dentro del anexo dedicado a este asunto 13.

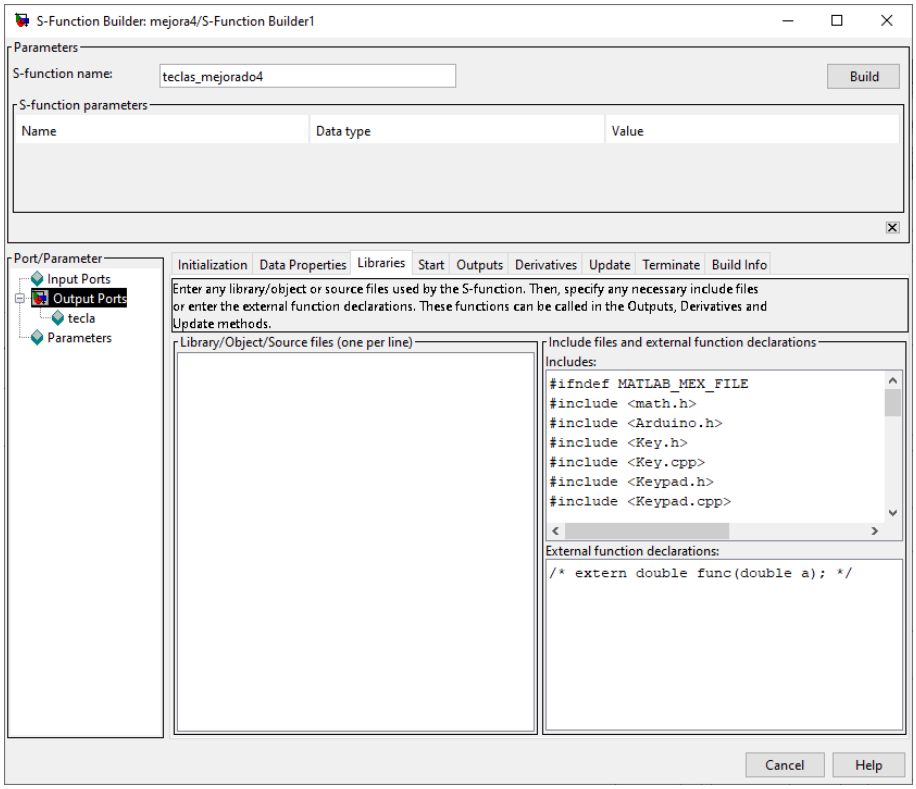


Figura 12.23 – Captura de la sección de librerías

Es en el cuarto apartado, en el de “Outputs”, donde el código de verdad es creado. Aquí se ha definido el funcionamiento de las teclas. Las numéricas irán destinadas a la introducción de números, como es lógico, y las teclas A, B, C, D y E; irán definidas, respectivamente, para confirmar consigna, confirmar ganancia proporcional, confirmar ganancia integral y confirmar la derivativa.

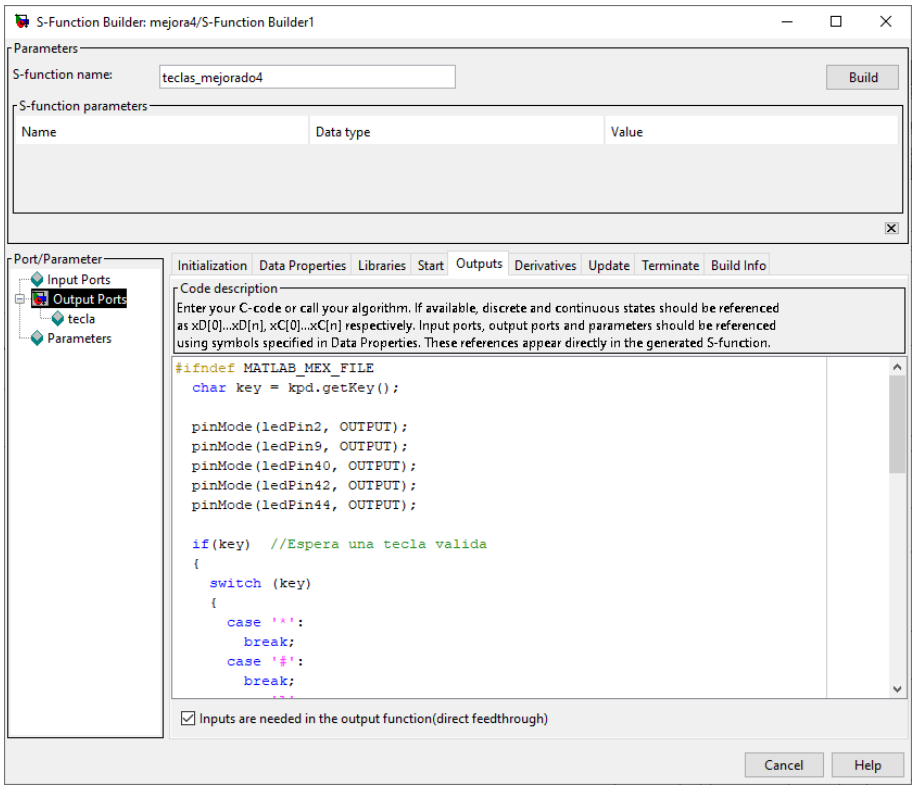


Figura 12.24 – Captura de la sección del código destinado al funcionamiento del teclado

12.7.3.1. Detalles del funcionamiento y programación

Una vez el programa ya esté finalizado, y tras su compilación, se generará un archivo del mismo nombre que el bloque creado, excepto que su nombre terminará en"_wrapper.c". A este archivo tendrá que renombrarse a ".cpp", para que MATLAB lo pueda interpretar en la compilación. Una vez su terminación esté cambiada, habrá que hacer un doble click sobre el archivo y modificar dos líneas de este. En este archivo habrá dos funciones definidas como void. Delante de este, habrá que escribir "extern "C"".

La consigna, que puede variarse de 0 a 99, siempre habrá que introducirla a través de dos pulsaciones. Es decir, si se quiere introducir 7, se tendrá que teclear "07". Si el número ya es de dos cifras, no habrá problema alguno. Así, tras la pulsación de dos cifras numéricas y de su respectiva tecla de confirmación, la consigna es modificada. Si por el contrario, tras el tecleo de la primera cifra, se acaba pulsando la B, (pues se quiere fijar el valor de Kp por ejemplo), se activará el Sample and Hold dedicado a almacenar la Kp y la cifra de las decenas guardada será descartada. Si una segunda tecla numérica es pulsada tras la primera, ambas serán guardadas con la pulsación de A, ocupando la primera la posición de las decenas y la segunda, de las unidades. Con esto, se realiza el cálculo: $Decenas * 10 + Unidades$, y da lugar al número entero de dos cifras que define la consigna.

Las constantes, por el contrario, solo podrán ser de una única cifra significativa.

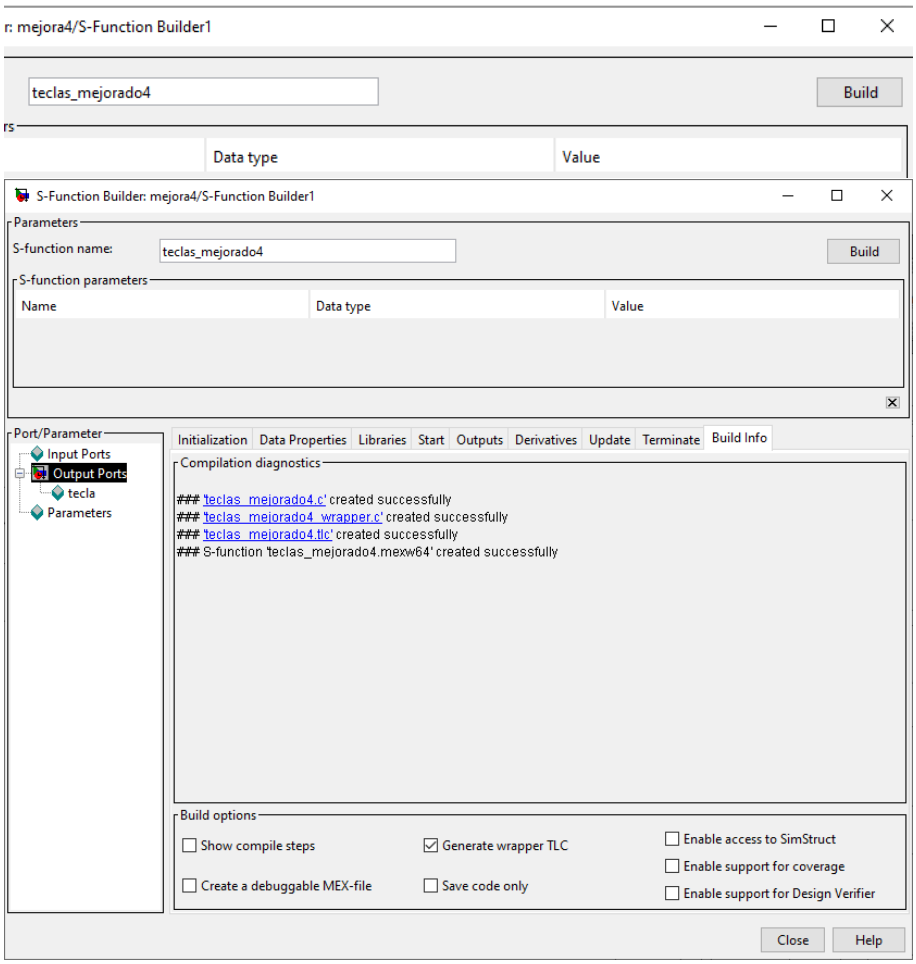


Figura 12.25 – La compilación se efectuará al pulsar el botón “Build”

Tal y como ya se ha reflejado, se ha optado por un método de introducción de variables por teclado de la forma “pulsación y captura”. Cada pulsación crea un flanco que almacena la cifra pulsada en una especie de “buffer”, esperando una segunda pulsación que confirme la acción a llevarse a cabo. Este método de detección de flancos es ejecutado de forma totalmente analógica, que interactúa con el programa en cuestión. La aplicación de este método, en un principio más engorroso que una solución totalmente programada nace por un motivo justificado.

Dentro de la aplicación compiladora e intérprete de C incluida en Simulink, nos ha sido imposible hacer funcionar cualquier tipo de programa que incluyese recursividad. Esto descarta la implementación estándar de introducción de números por teclado en Arduino, donde se hace por medio de una cadena de caracteres de la longitud que uno quiera e incluso con separación decimal. Nuestra solución, intentó simular una ejecución jerarquizada en etapas, simulando una estructura “if...else” simple, de forma totalmente analógica; para así resolver los inconvenientes surgidos y dar utilidad al programa.

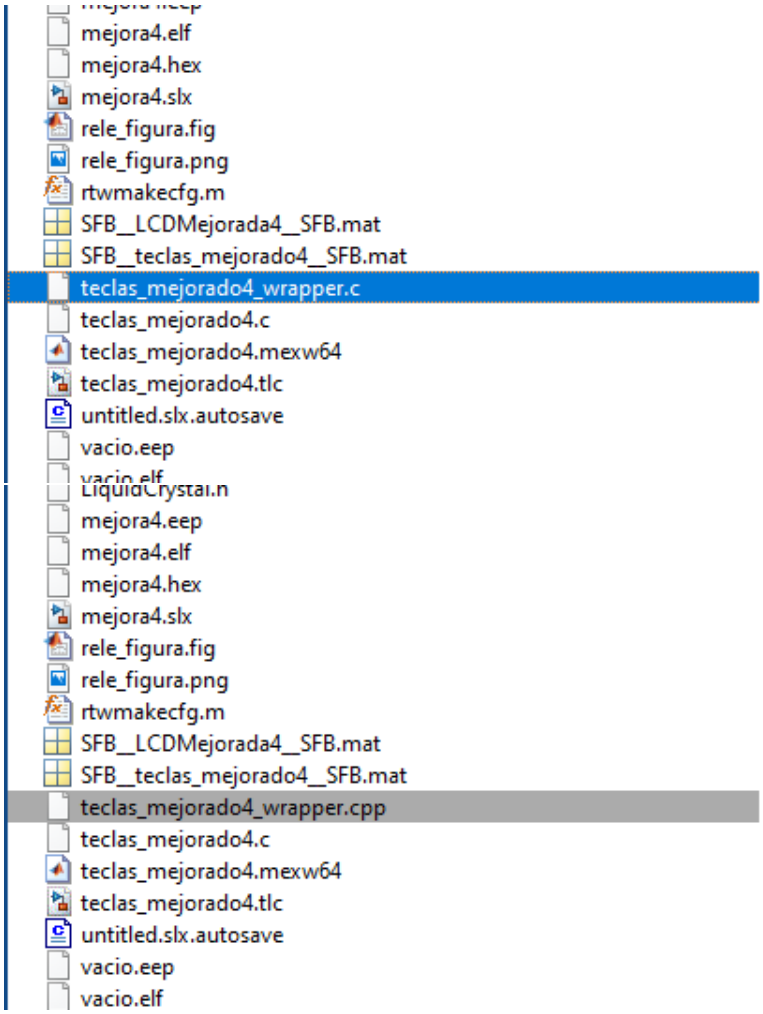


Figura 12.26 – Resultado de la compilación

Debido a que el teclado utilizado fue diseñado para la introducción de caracteres, que a su vez sean mostrados directamente en una pantalla LCD, (cuando ambos funcionan dentro del entorno estándar de Arduino), los caracteres leídos tras una pulsación, están en código de caracteres ASCII. Por lo tanto, para ser utilizados correctamente, será necesario restar 48, (posición del 0 en dicho código), para así obtener su equivalente decimal. Para mostrar caracteres en la pantalla, esta conversión no será necesaria. Los caracteres generados dentro del programa, son mostrados correctamente en la LCD, (las propias librerías son quien efectúan la conversión).

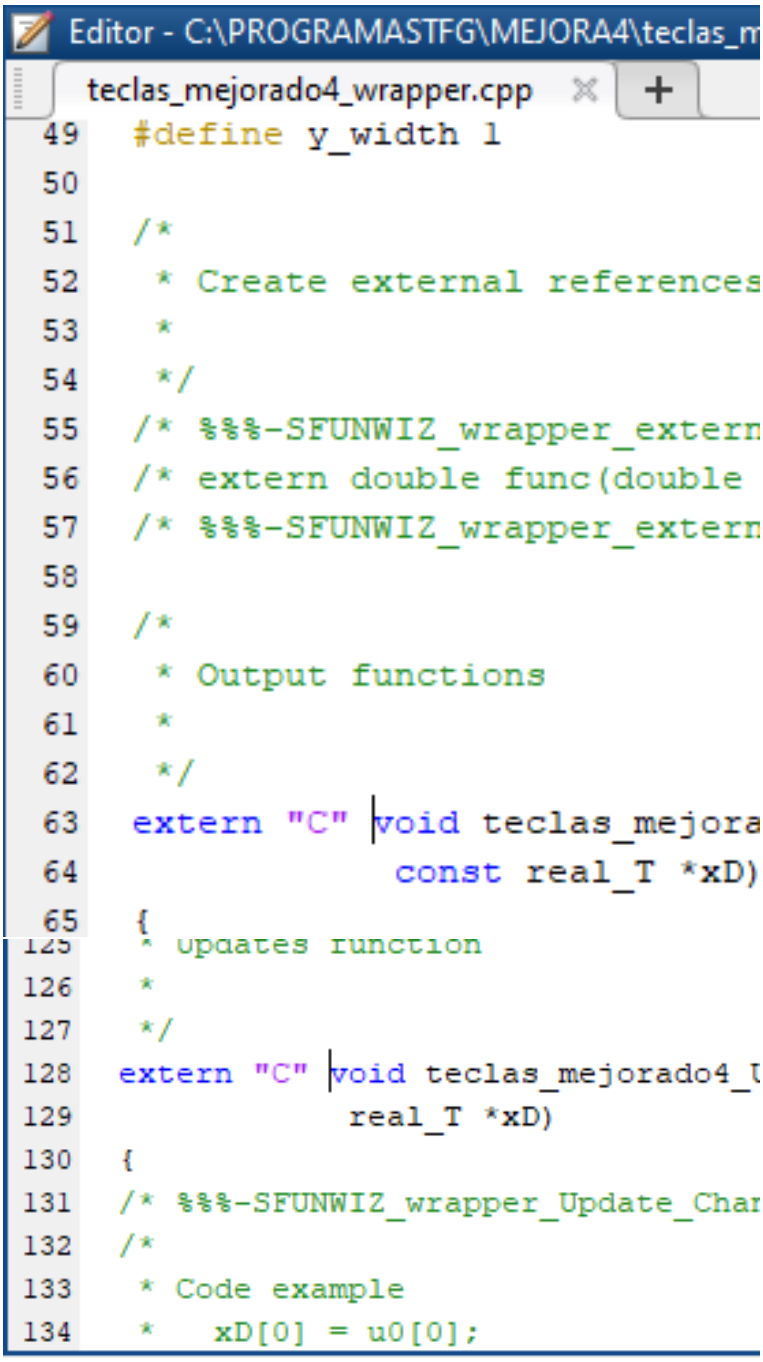


Figura 12.27 – Modificación del resultado de la compilación

12.7.4. Creación del bloque de la pantalla

En el caso de la pantalla; las librerías descargables, son “LiquidCrystal.h” y “LiquidCrystal.cpp”. Este caso es mucho más fácil de diseñar, pues las modificaciones que hay que hacer con respecto a la programación en C de Arduino son mínimas, tal y como se refleja en su código 13. Lo único que añadiremos, serán las variables de entrada, las cuales, serán los valores numéricos del modelo que queremos que sean plasmados sobre la pantalla. Habrá que tener cuidado del tipo de dato que sea definido y el tipo que tenga dentro del programa. Si el dato

tiene formato de entero de bits, y dentro de la pantalla, posee formato de double, se generará un error. Se podrá solucionar por medio de un convertidor de tipo de dato, existente como bloque dentro de Simulink y utilizado en varios lugares dentro de los programas. Al contrario que con el teclado, para la pantalla sí existía en internet trabajo previo de adaptación de esta a Simulink. Una fuente de la que se ha sacado información para este trabajo es la siguiente [21], ya que comparte la utilización del bloque S-Function.

12.8. Recursos utilizados, errores e informe de compilación

Cada vez que un programa es generado, se genera un informe de warnings, errores de compilación, si es que existen, y por último, recursos consumidos de la placa. Los primeros, solo son informativos, deben ser entendidos y asumidos en caso de ser ignorados, pero no suponen un impedimento para ejecutar el programa, mientras que los fallos vinculados a los dos últimos, detienen la compilación y el código no se traslada a la placa. El segundo, hace referencia, o bien a un problema en el programa en sí, a un problema con la configuración de la placa, como si que está desconectada. El informe de recursos, pasará normalmente desapercibido a no ser que genere un error debido a que en el Arduino elegido no cumple con los requisitos exigidos por el programa en cuanto a extensión. En estos casos propuestos, esto no supondrá un problema.

13 Secciones del código utilizado

Código 13.1: Código de la sección “Libreries” de la pantalla LCD

```
//Codigo dentro de "Libreries"

#include <math.h>
#ifndef MATLAB_MEX_FILE
#include "LiquidCrystal.h"
#include "LiquidCrystal.cpp"


LiquidCrystal lcd(12, 11, 22, 24, 26, 28);
#endif
```

Código 13.2: Código de la sección “Libreríes” del teclado

```
//Codigo dentro de "Libreries"

#ifndef MATLAB_MEX_FILE
#include <math.h>
#include <Arduino.h>
#include <Key.h>
#include <Key.cpp>
#include <Keypad.h>
#include <Keypad.cpp>

int ledPin9 = 9;
int ledPin2 = 2;
int ledPin40 = 40;
int ledPin42 = 42;
int ledPin44 = 44;

const byte ROWS = 4; // Cuatro filas
const byte COLS = 4; // Cuatro columnas
// Define el mapa de teclas
char keys[ROWS][COLS] = {
   { '1','2','3', 'A' },
   { '4','5','6', 'B' },
   { '7','8','9', 'C' },
   { '*','0','#', 'D' }
};
// Conecta ROW0, ROW1, ROW2 and ROW3 a esos pines del Arduino
byte rowPins[ROWS] = { 14, 15, 16, 17 };
// Connect keypad COL0, COL1, COL2 y COL3 a esos pines del Arduino
byte colPins[COLS] = { 7, 6, 5, 3 };

// Crea el Keypad
Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

#endif
```

Código 13.3: Código de la sección de “Outputs” de la pantalla LCD

```
//Codigo dentro de "Outputs"

if(xD[0]==1)
{
    #ifndef MATLAB_MEX_FILE

    lcd.setCursor(0,0);
    lcd.print("P");
    lcd.print(char_p[0]);
    lcd.print(" ");
    lcd.print("I");
    lcd.print(char_i[0]);
    lcd.print(" ");
    lcd.print("D");
    lcd.print(char_d[0]);
    lcd.print(" ");


    lcd.setCursor(0,1);
    lcd.print("O:");
    lcd.print(char_val[0]);
    lcd.print((char)223);
    lcd.print("C");
    lcd.print(" ");
    lcd.print("I:");
    lcd.print(char_val2[0]);
    lcd.print((char)223);
    lcd.print("C");
    lcd.print(" ");

    #endif
}
```

Código 13.4: Código de la sección “Outputs” del teclado

```
//Codigo dentro de "Outputs"

#ifndef MATLAB_MEX_FILE
  char key = kpd.getKey();

  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin9, OUTPUT);
  pinMode(ledPin40, OUTPUT);
  pinMode(ledPin42, OUTPUT);
  pinMode(ledPin44, OUTPUT);

  if(key)  //Espera una tecla valida
  {
    switch (key)
    {
      case '*':
        break;
      case '#':
        break;
      case 'A':
        digitalWrite(ledPin2,LOW);
        digitalWrite(ledPin9,HIGH);
        digitalWrite(ledPin40,LOW);
        digitalWrite(ledPin44,LOW);
        digitalWrite(ledPin42,LOW);
        break;
      case 'B':
        digitalWrite(ledPin40,HIGH);
        digitalWrite(ledPin44,LOW);
        digitalWrite(ledPin42,LOW);
        digitalWrite(ledPin9,LOW);
        digitalWrite(ledPin2,LOW);
        break;
      case 'C':
        digitalWrite(ledPin44,LOW);
        digitalWrite(ledPin2,LOW);
        digitalWrite(ledPin42,HIGH);
        digitalWrite(ledPin40,LOW);
        digitalWrite(ledPin9,LOW);
        break;
      case 'D':
        digitalWrite(ledPin42,LOW);
        digitalWrite(ledPin44,HIGH);
        digitalWrite(ledPin40,LOW);
        digitalWrite(ledPin2,LOW);
        digitalWrite(ledPin9,LOW);
        break;
```

```
      default:
        tecla[0]=key-48;
        digitalWrite(ledPin2,HIGH);
        digitalWrite(ledPin9,LOW);
        break;
    }
  }

#endif
```

Código 13.5: Código de la sección “Update” de la pantalla LCD

```
// Codigo dentro de "Update"

// La variable xD (estado discreto) almacena el estado de una variable que puede cambiar de
// valor segun los intereses del programador.
// En este caso la uso para que me haga la animacion solo en el arranque

if(xD[0]!=1)
{
    #ifndef MATLAB_MEX_FILE

    lcd.begin(16,2);
    lcd.setCursor(0,0);
    lcd.print("Arduino&Simulink");
    delay(2000);

    lcd.setCursor(0,0);
    lcd.print("Regulador PID   ");
    delay(1000);
    lcd.setCursor(0,0);
    lcd.print("Regulador PID.  ");
    delay(1000);
    lcd.setCursor(0,0);
    lcd.print("Regulador PID.. ");
    delay(1000);
    lcd.setCursor(0,0);
    lcd.print("Regulador PID...");
    delay(1000);

    lcd.begin(16,2);

    delay(1000);

    #endif
    xD[0]=1;
}
```

14 Uso del regulador

- Una vez la tarea de programación ha terminado, se podrá subir el código a la placa Arduino, conectando el dispositivo a un PC por medio de la conexión USB.
- Tras la compilación y subida, el programa se ejecutará en el microcontrolador tras cada inicio.
- El sistema se encenderá con su conexión a la corriente.
- El uso del PC ya no será necesario. Será opcional en el caso de querer graficar.
- Es posible la conexión simultánea del Arduino a la tensión y al USB del ordenador.
- La interacción con el regulador por teclado necesita confirmación por parte del usuario.
- Ante un cambio de consigna u otro parámetro, el usuario ha de pulsar una tecla de confirmación.
- El sistema ante una pulsación del teclado anula la tensión sobre la señal de control. El sistema entra en reposo.
- Cuando está en modo reposo, el PID no computa los cálculos ocurridos en esas iteraciones. De lo contrario se introducirían valores incorrectos en sus cálculos.

TÍTULO: **IMPLEMENTACIÓN DE UN REGULADOR PID GENÉRICO EMBEBIDO**

PLANOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

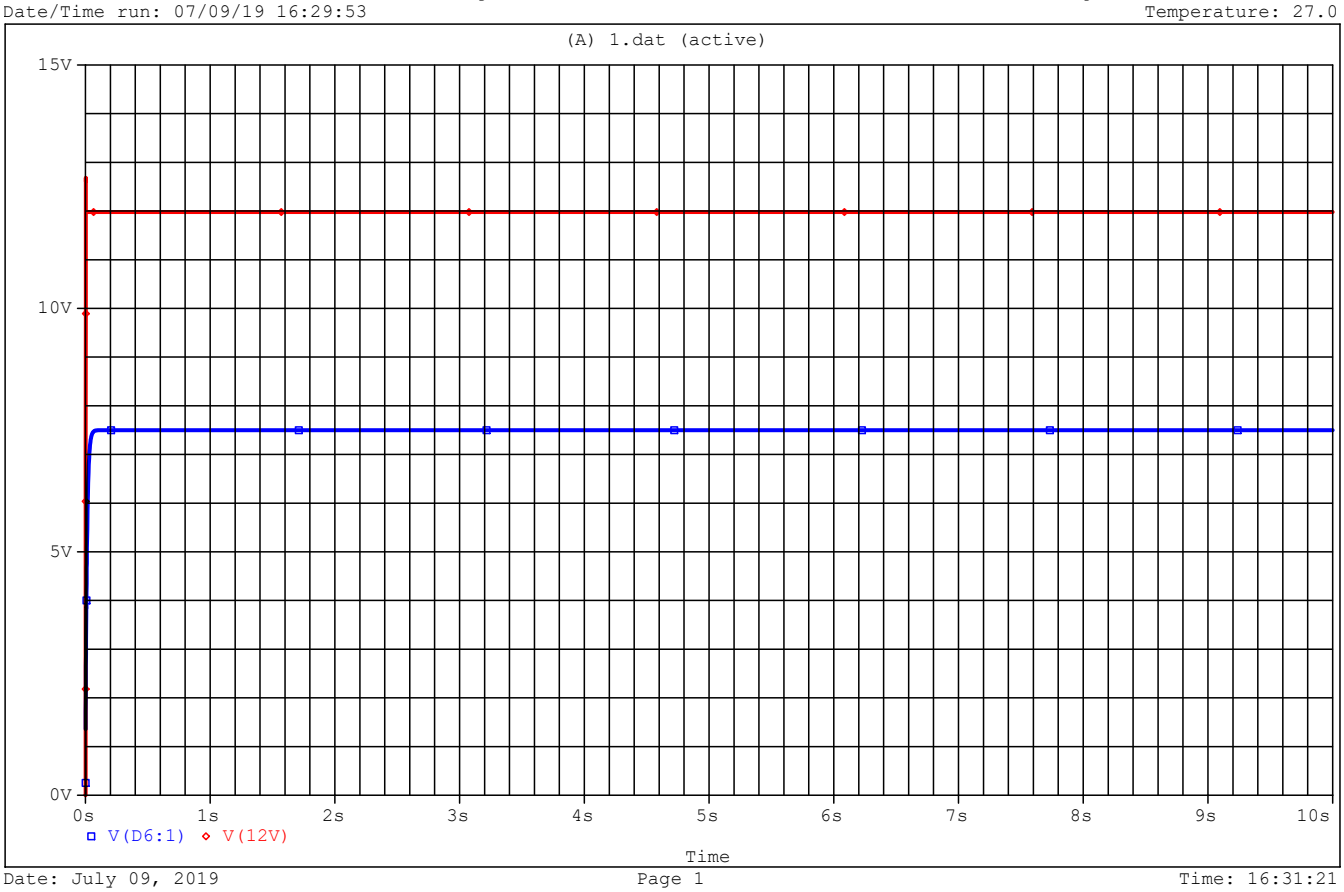
FECHA: SEPTIEMBRE DE 2019

AUTOR: EL ALUMNO

Fdo.: ALEJANDRO FERNÁNDEZ ROMERO

Índice de planos

** Profile: "SCHEMATIC1-1" [C:\Users\Alejandro Fernández\OneDrive - Universidade da Coruña\Tfg\KICAD\Nuevafuen...
Date/Time run: 07/09/19 16:29:53 Temperature: 27.0



Date: July 09, 2019 Page 1 Time: 16:31:21

UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A176

TÍTULO DEL TFG:

IMPLEMENTACIÓN DE UN PID GENÉRICO EMBEBIDO

TÍTULO DEL PLANO:

SIMULACIÓN DE LA FUENTE DE ALIMENTACIÓN

FECHA: SEPTIEMBRE 2019

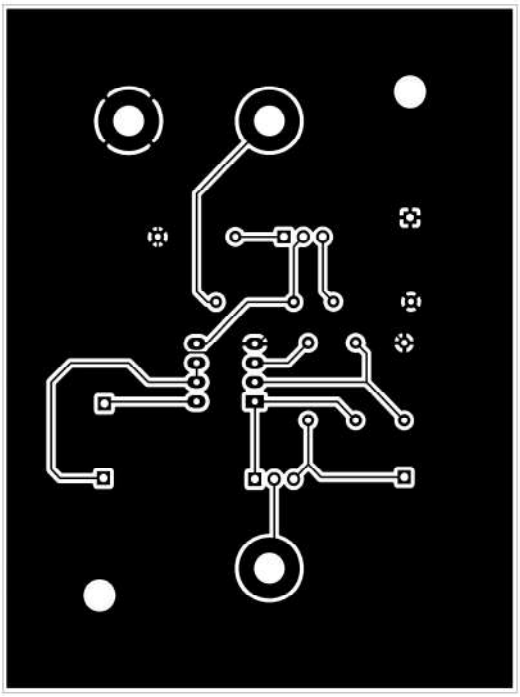
ESCALA: S/E

AUTOR:

ALEJANDRO FERNÁNDEZ ROMERO

FIRMA:

PLANO Nº: 01



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A176

TÍTULO DEL TFG:

IMPLEMENTACIÓN DE UN PID EMBEBIDO GENÉRICO

TÍTULO DEL PLANO:

CIRCUITO IMPRESO DE LA FUENTE

FECHA: SEPTIEMBRE 2019

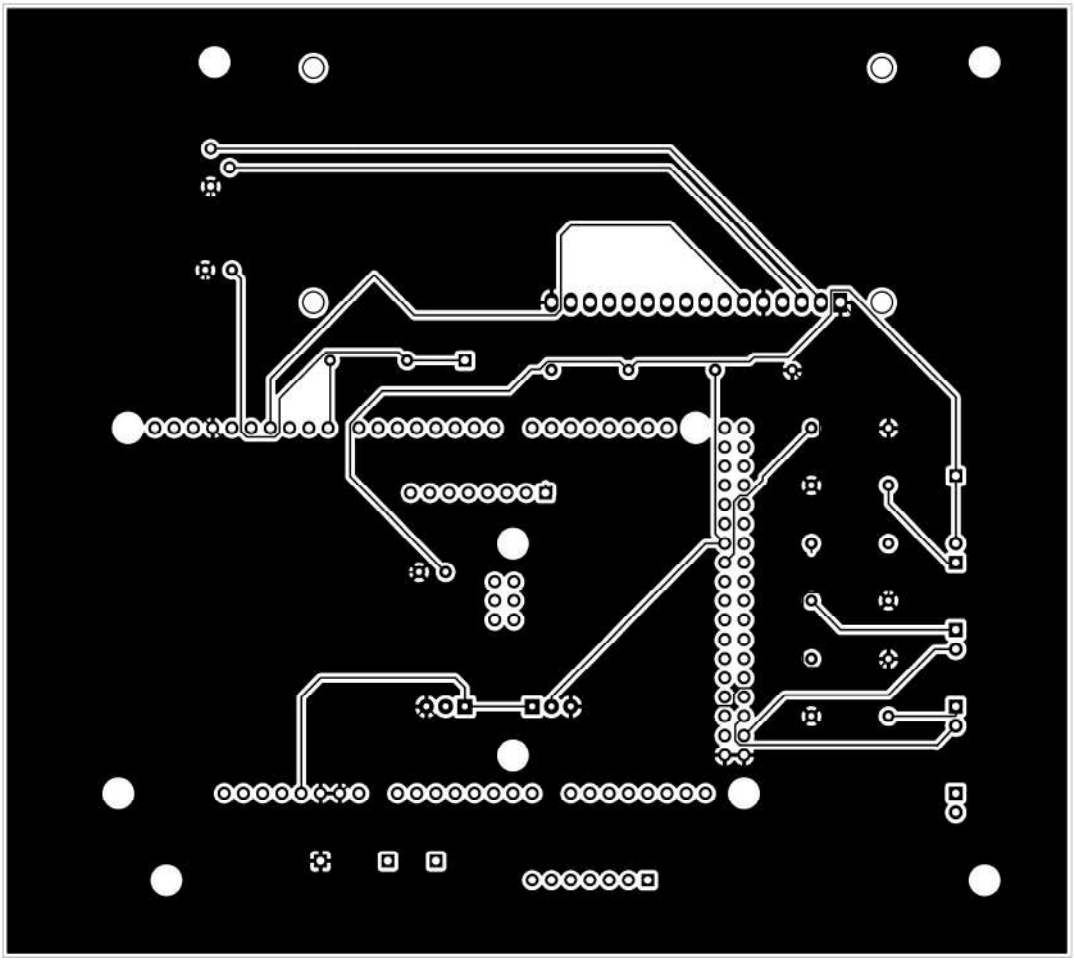
ESCALA: 1:1

AUTOR:

ALEJANDRO FERNÁNDEZ ROMERO

FIRMA:

PLANO Nº: 02



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A176

TÍTULO DEL TFG:

IMPLEMENTACIÓN DE UN PID EMBEBIDO GENÉRICO

TÍTULO DEL PLANO:

CIRCUITO IMPRESO DE LA FUENTE

FECHA: SEPTIEMBRE 2019

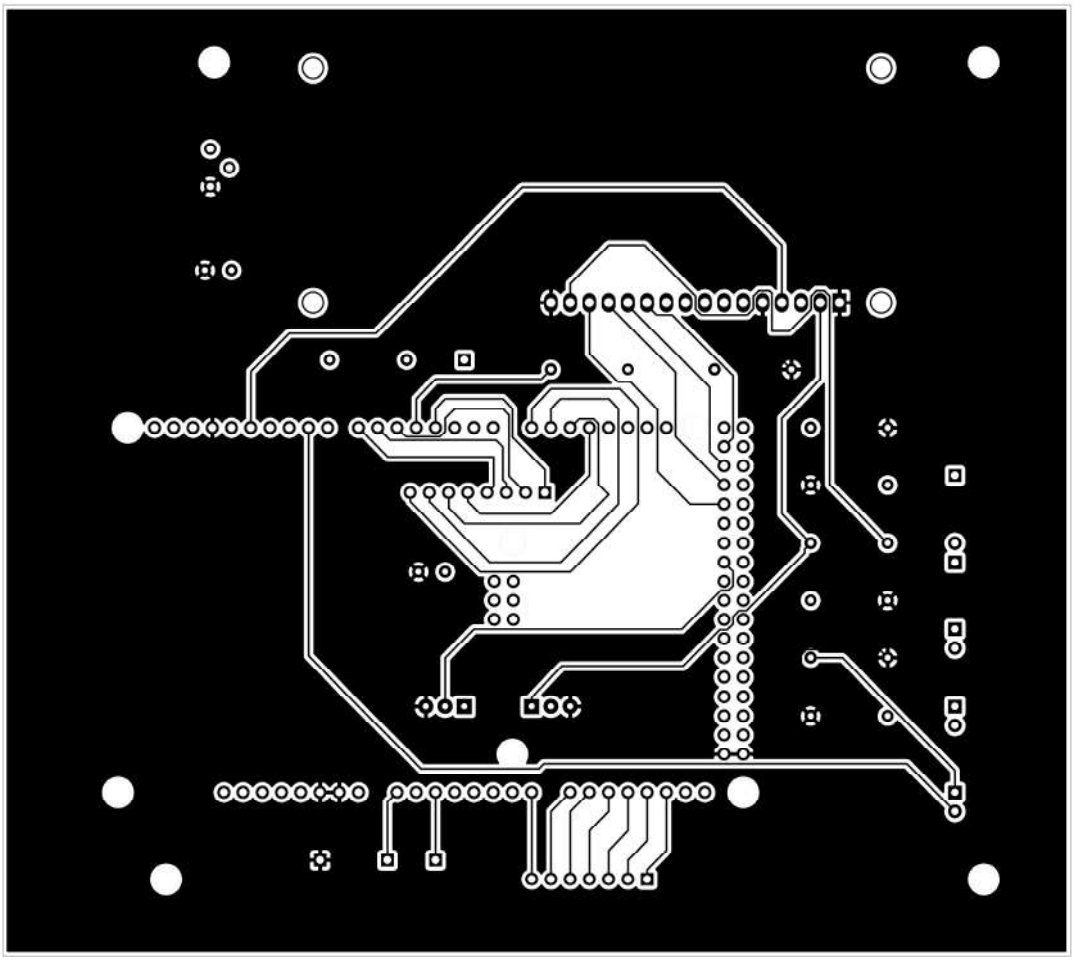
ESCALA: 1:1

AUTOR:

ALEJANDRO FERNÁNDEZ ROMERO

FIRMA:

PLANO Nº: 03



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A176

TÍTULO DEL TFG:

IMPLEMENTACIÓN DE UN PID EMBEBIDO GENÉRICO

TÍTULO DEL PLANO:

CIRCUITO IMPRESO DE LA FUENTE

FECHA: SEPTIEMBRE 2019

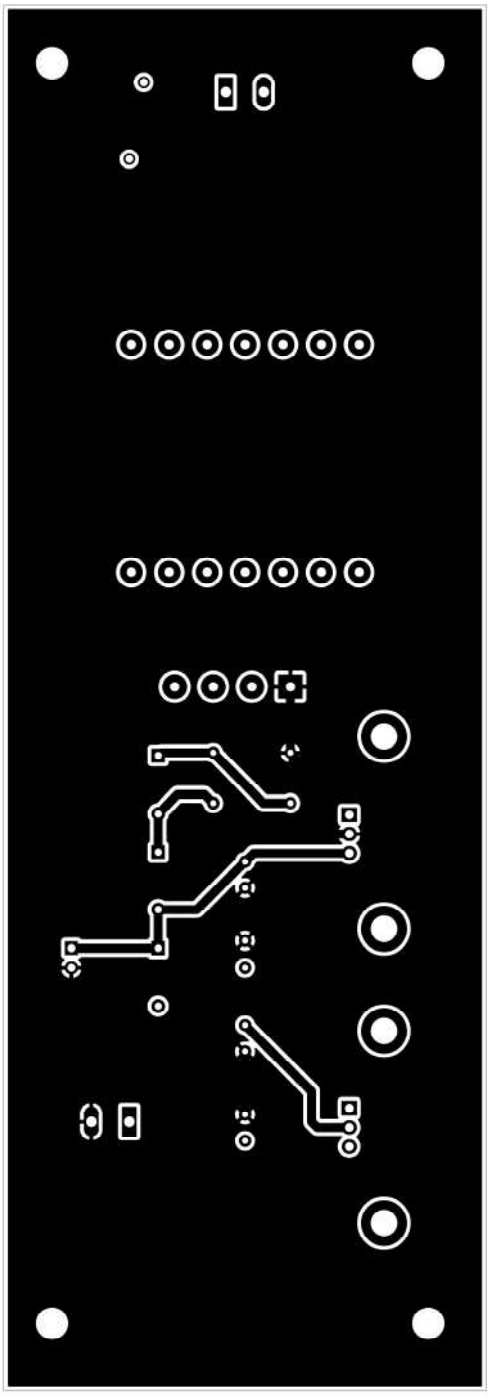
ESCALA: 1:1

AUTOR:

ALEJANDRO FERNÁNDEZ ROMERO

FIRMA:

PLANO Nº: 04



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A176

TÍTULO DEL TFG:

IMPLEMENTACIÓN DE UN PID EMBEBIDO GENÉRICO

TÍTULO DEL PLANO:

CIRCUITO IMPRESO DE LA FUENTE

FECHA: SEPTIEMBRE 2019

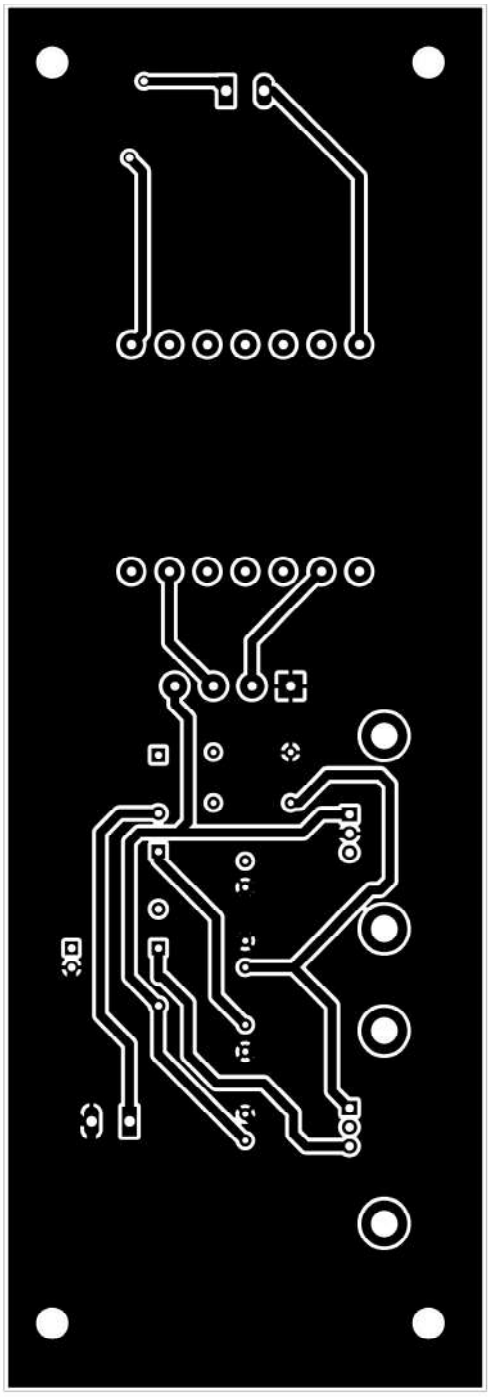
ESCALA: 1:1

AUTOR:

ALEJANDRO FERNÁNDEZ ROMERO

FIRMA:

PLANO Nº: 05



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A176

TÍTULO DEL TFG:

IMPLEMENTACIÓN DE UN PID EMBEBIDO GENÉRICO

TÍTULO DEL PLANO:

CIRCUITO IMPRESO DE LA FUENTE

FECHA: SEPTIEMBRE 2019

ESCALA: 1:1

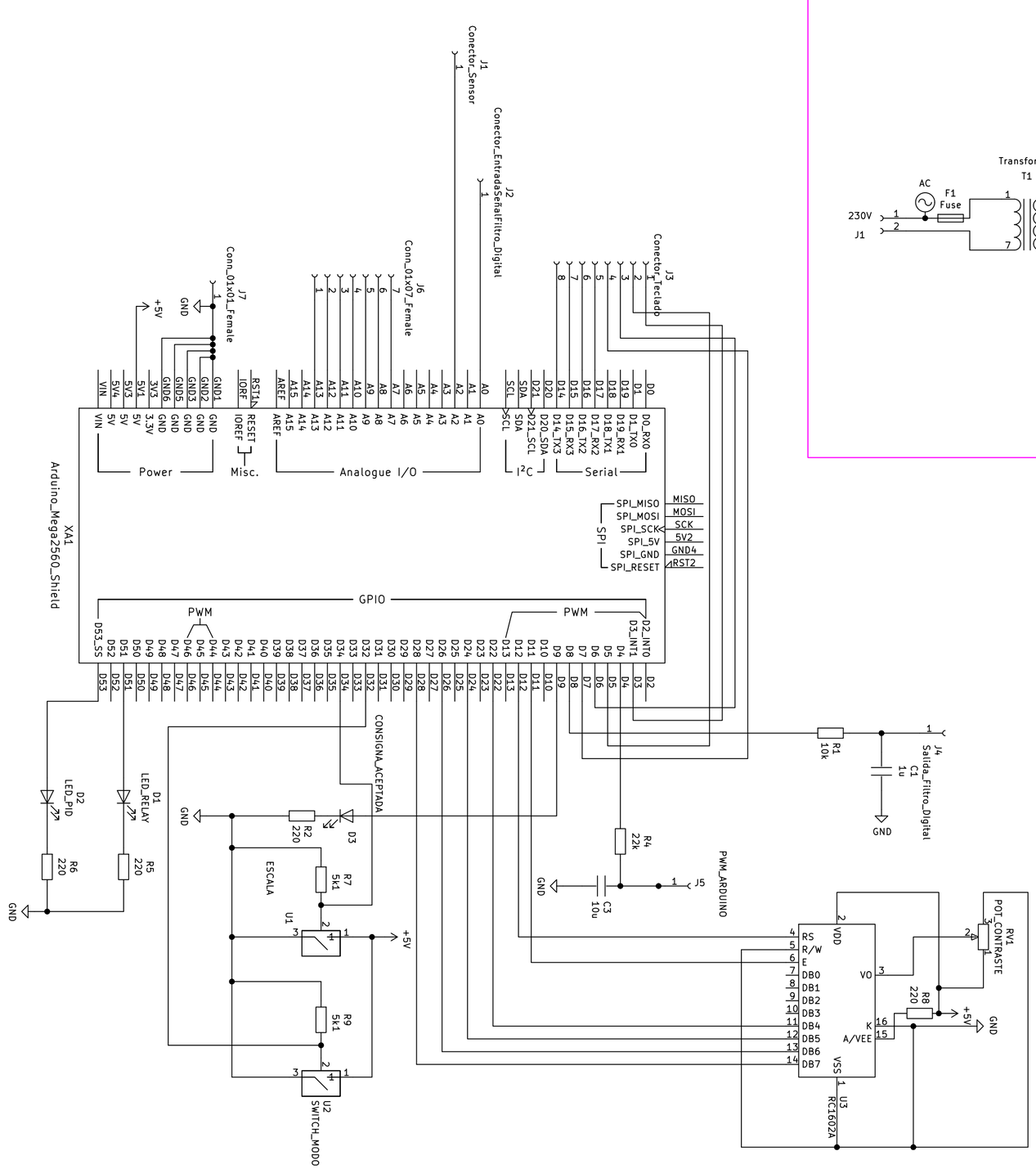
AUTOR:

ALEJANDRO FERNÁNDEZ ROMERO

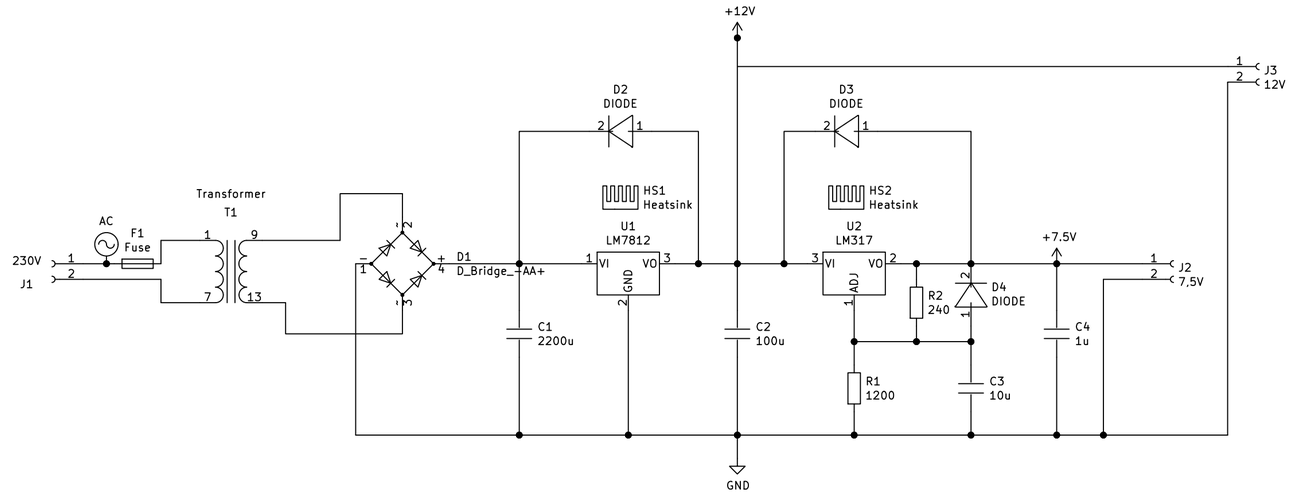
FIRMA:

PLANO Nº: 06

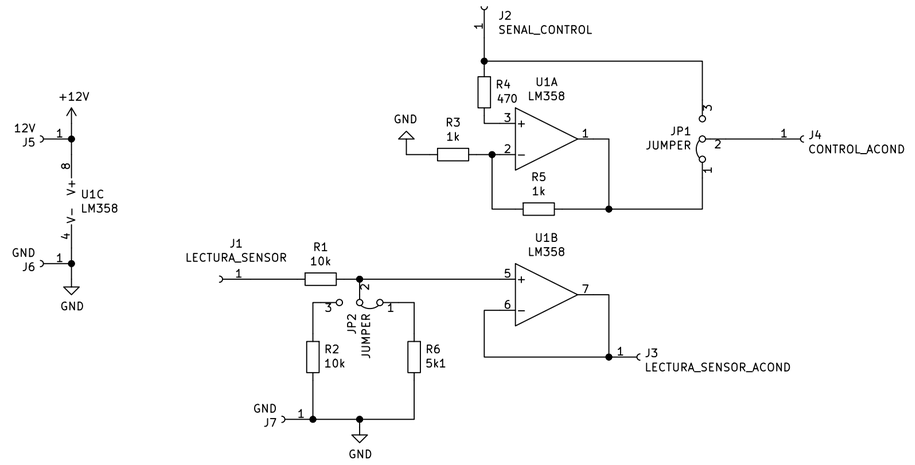
ENTRENADOR



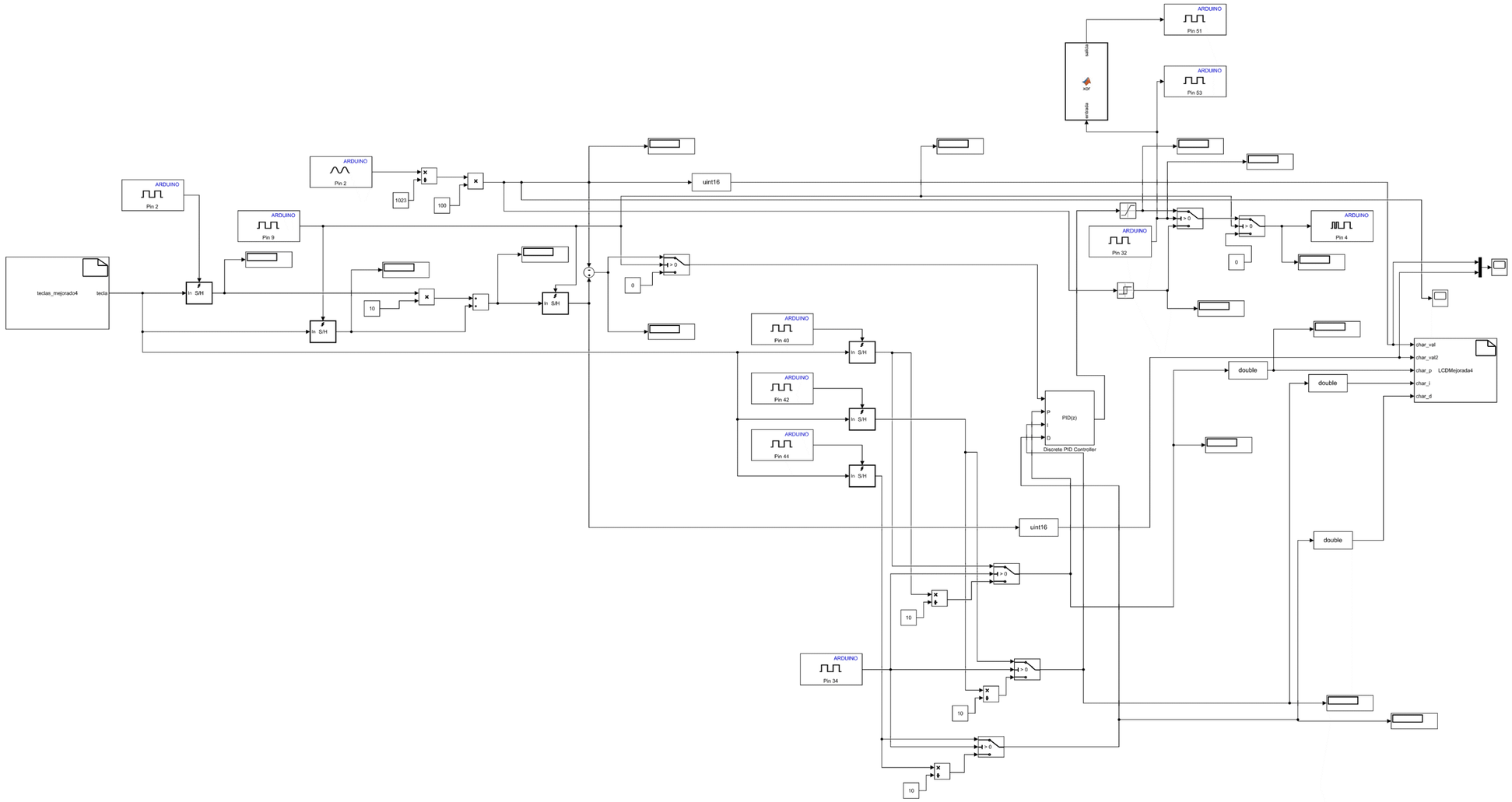
FUENTE LINEAL DE TENSIÓN



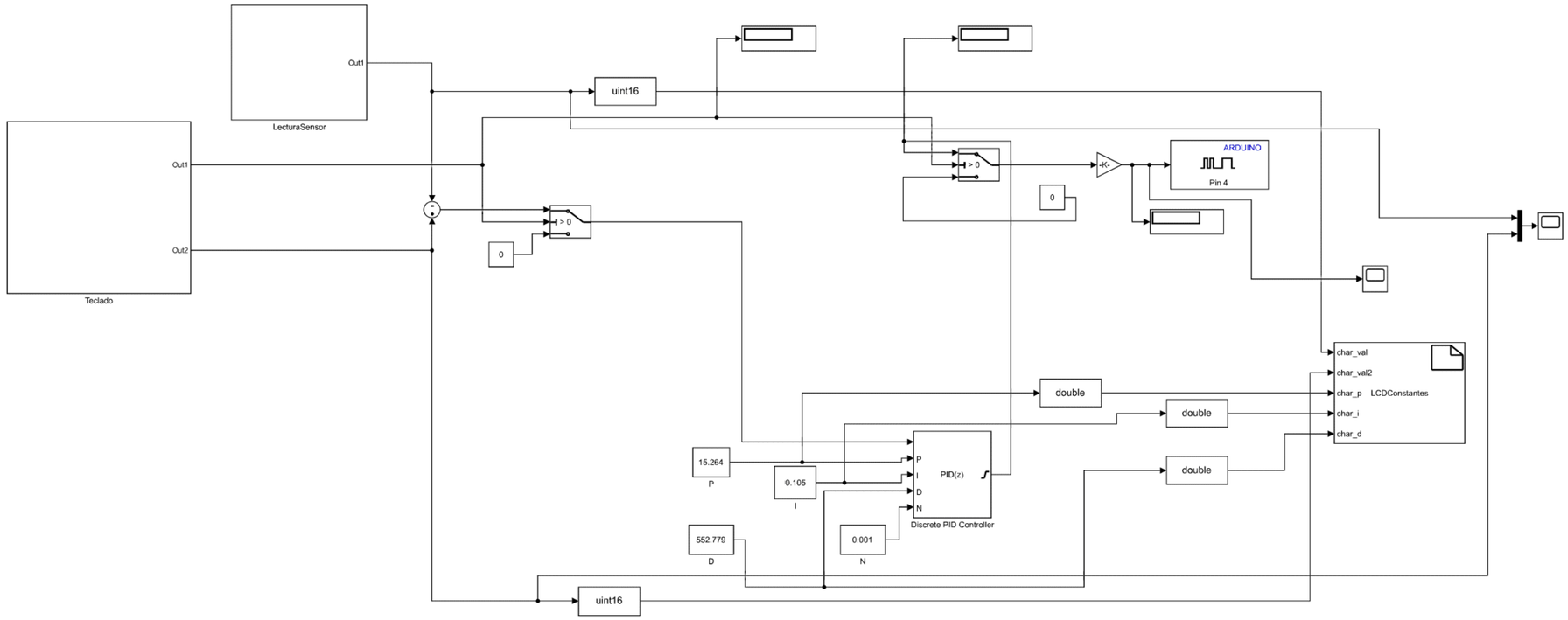
ACONDICIONAMIENTO DE TENSIONES



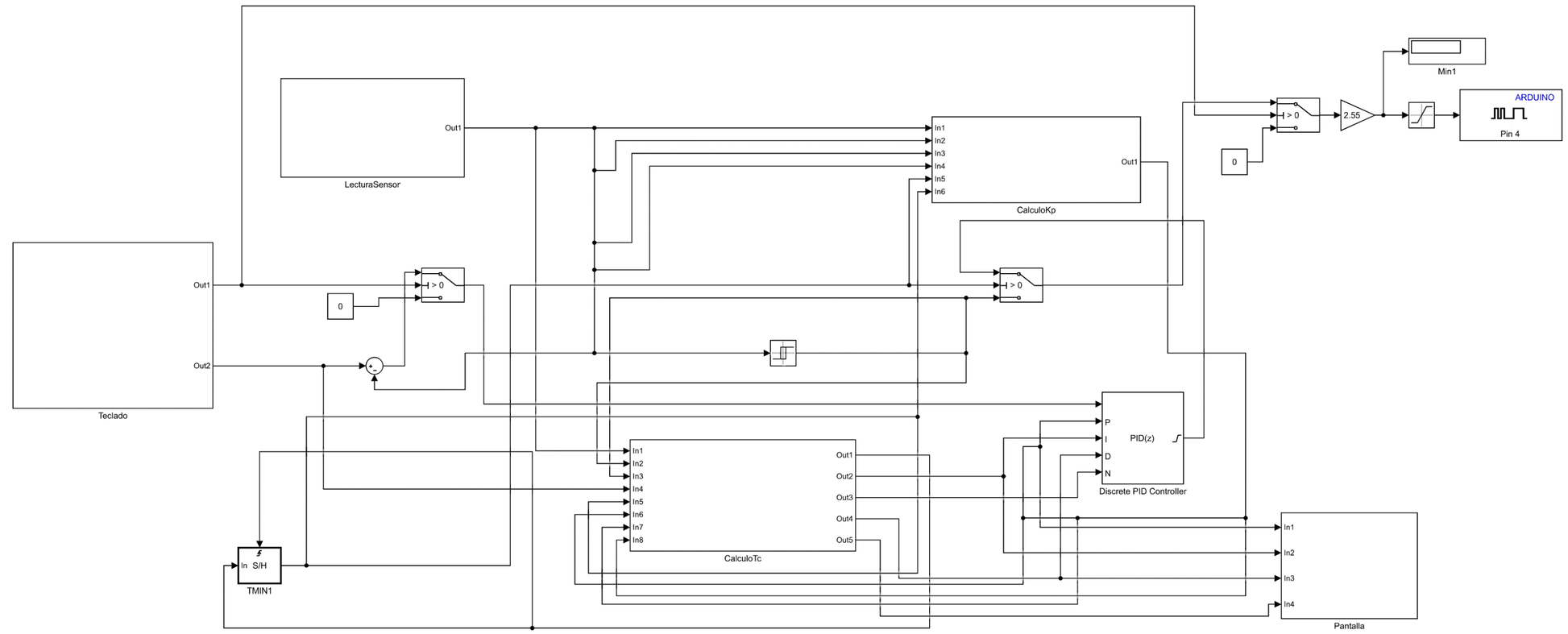
UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA		TFG Nº: *770G01A176*
TÍTULO DEL TFG **IMPLEMENTACIÓN DE UN PID EMBEBIDO GENÉRICO**		
TÍTULO DEL PLANO **DISEÑO DE LOS CIRCUITOS ELECTRÓNICOS**		FECHA: SEPTIEMBRE 2019 ESCALA: S/E
AUTOR **ALEJANDRO FERNANDEZ ROMERO**	FIRMA	PLANO Nº: 07



UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: *770G01A176*
TÍTULO DEL TFG **IMPLEMENTACIÓN DE UN PID EMBEBIDO GENÉRICO**	
TÍTULO DEL PLANO **PROGRAMA DEL PID EMBEBIDO CON CONSTANTES SELECCIONABLES POR EL USUARIO**	FECHA: SEPTIEMBRE 2019 ESCALA: S/E
AUTOR **ALEJANDRO FERNANDEZ ROMERO** FIRMA	PLANO Nº: 08

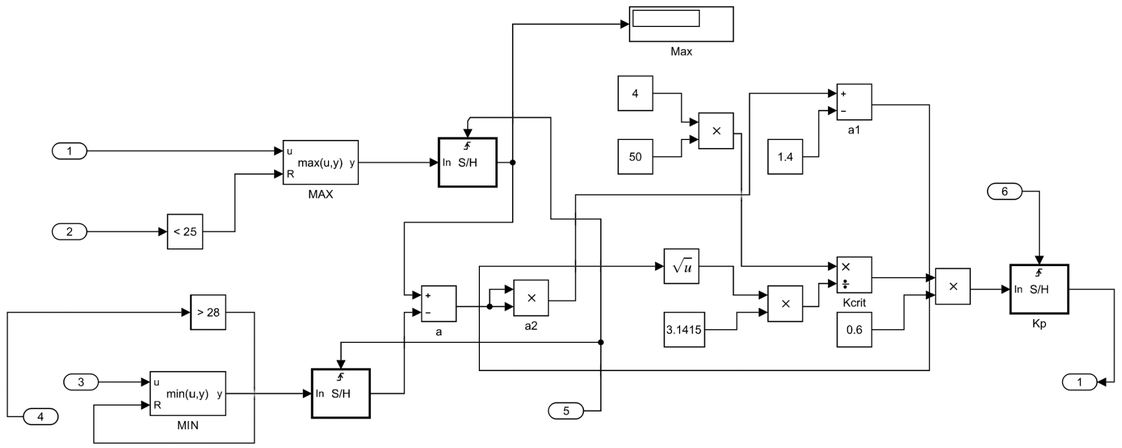


UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: *770G01A176*
TÍTULO DEL TFG **IMPLEMENTACIÓN DE UN PID EMBEBIDO GENÉRICO**	
TÍTULO DEL PLANO **PROGRAMA DEL PID EMBEBIDO CON CONSTANTES PREDETERMINADAS**	FECHA: SEPTIEMBRE 2019 ESCALA: S/E
AUTOR **ALEJANDRO FERNANDEZ ROMERO** FIRMA	PLANO Nº: 09

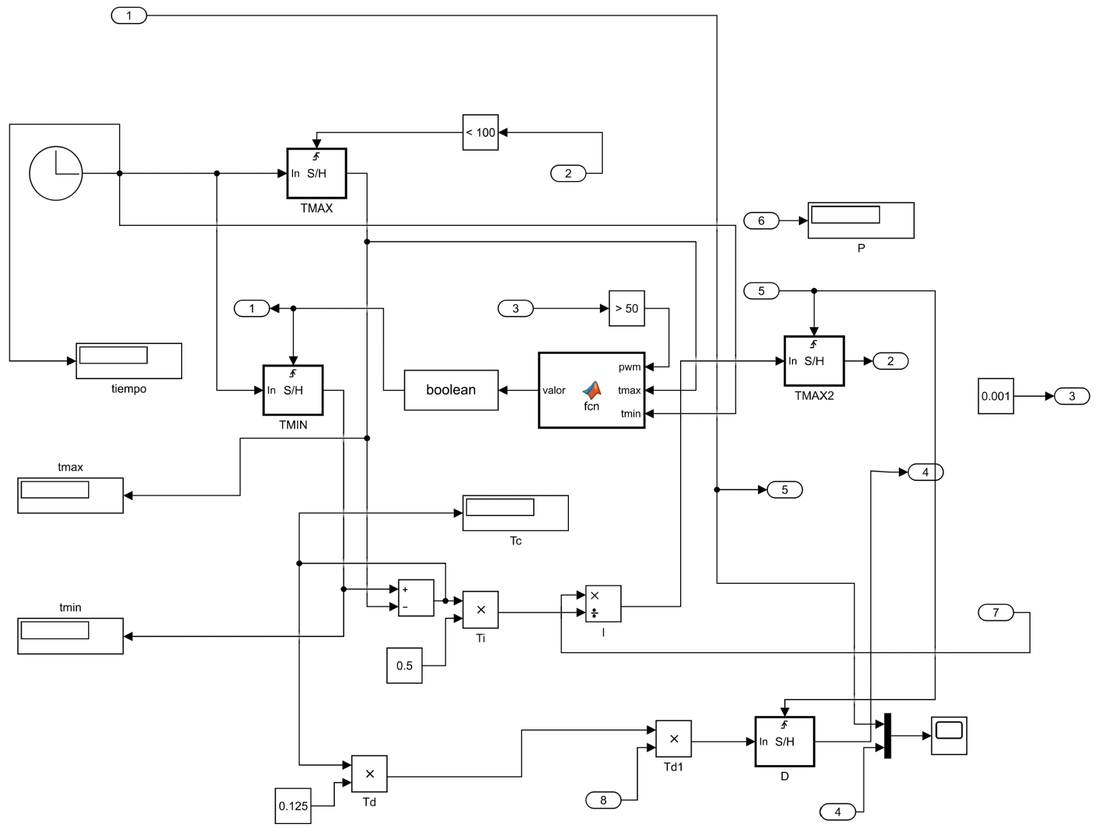


UNIVERSIDADE DA CORUÑA **ESCUELA UNIVERSITARIA POLITÉCNICA** **GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA**	**TFG Nº:** ***770G01A176***
TÍTULO DEL TFG **IMPLEMENTACIÓN DE UN PID EMBEBIDO GENÉRICO**	
TÍTULO DEL PLANO **PROGRAMA DEL PID EMBEBIDO AUTOAJUSTABLE**	**FECHA: SEPTIEMBRE 2019**
	ESCALA: S/E
AUTOR **ALEJANDRO FERNANDEZ ROMERO** / **FIRMA**	**PLANO Nº: 10**

CÁLCULO DE KP



CÁLCULO DE TC



UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: *770G01A176*
TÍTULO DEL TFG **IMPLEMENTACIÓN DE UN PID EMBEBIDO GENÉRICO**	
TÍTULO DEL PLANO **RECORTE DEL PROGRAMA DEL PID EMBEBIDO AUTOAJUSTABLE**	FECHA: SEPTIEMBRE 2019 ESCALA: S/E
AUTOR **ALEJANDRO FERNANDEZ ROMERO** FIRMA	PLANO Nº: 11

TÍTULO: **IMPLEMENTACIÓN DE UN REGULADOR PID GENÉRICO EMBEBIDO**

PLIEGO DE CONDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2019

AUTOR: EL ALUMNO

Fdo.: ALEJANDRO FERNÁNDEZ ROMERO

Índice del documento PLIEGO DE CONDICIONES

14.1. Especificaciones de los materiales

Los componentes que fueron seleccionados para conformar el montaje propuesto, cumplen las necesidades y requisitos del mismo.

14.1.1. Selección de los componentes

El criterio que se ha seguido para seleccionar cada uno de los elementos, fue el estudio de las hojas de características de cada uno. Tras este análisis, y una comparación con los valores nominales sacados de los cálculos efectuados, se llevó a cabo la elección.

Fueron elegidos materiales que se comercializan de forma legal en el mercado, por lo que éstos cumplirán con las normativas correspondientes. Será por lo tanto, responsabilidad de cada uno de los fabricantes, el correcto funcionamiento de cada uno de estos componentes.

14.1.2. Selección de los componentes sustituidos

Si por motivos de que se descatalogue un producto detallado en la lista, la compra no es posible, este se reemplazará por un producto similar/equivalente que cumpla o supere las especificaciones exigidas por el diseño planteado.

14.2. Verificación de los elementos del montaje

Asegurarse del correcto funcionamiento de cada uno de los elementos empleados es de vital importancia.

14.2.1. Circuitos impresos

Para evitar hipotéticos fallos en la fase de producción, que puedan conducir a un “malfuncionamiento eléctrico”, se ha pedido a la empresa productora, que realice un testeo eléctrico de las pistas y conexiones.

TÍTULO: **IMPLEMENTACIÓN DE UN REGULADOR PID GENÉRICO EMBEBIDO**

MEDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2019

AUTOR: EL ALUMNO

Fdo.: ALEJANDRO FERNÁNDEZ ROMERO

Índice del documento MEDICIONES

15 Elementos necesarios

Listado de los materiales que se creían necesarios para una primera fase del proyecto. No representan el estado final de este.

Elemento	Unidades
Ardjuino MEGA 2560	1
Teclado Matricial 4x4	1
Pantalla LCD 16x2	1
Kit de Iniciación Básico Arduino	1
Licencia de MATLAB estudiantil	1
Ordenador Personal	1
Potenciómetro	2
Transformador 12V 16W	1
Disipador 8,6 ºC/W	2
Interruptor PCB	2
Puente de diodos	1
Regulador de tensión LM317	1
Regulador de tensión LM7812	1
Circuito impreso placa de acondicionamiento	1
Circuito impreso fuente de alimentación	1
Circuito impreso placa del entrenador	1

Tabla 15.1 – Elementos necesarios en el momento de planteamiento del proyecto

TÍTULO: **IMPLEMENTACIÓN DE UN REGULADOR PID GENÉRICO EMBEBIDO**

PRESUPUESTO

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2019

AUTOR: EL ALUMNO

Fdo.: ALEJANDRO FERNÁNDEZ ROMERO

Índice del documento PRESUPUESTO

16 PRESUPUESTO

16.1. Mano de obra

Considerando un salario de 30 €/hora correspondiente a un graduado en ingeniería electrónica industrial, se harán los siguientes cálculos:

Actividad	Horas empleadas (h)	Coste Total (€)
Programación	60	1800,00
Búsqueda de información	30	900,00
Montaje	40	1200,00
Pruebas	60	1800,00
Total		**5700,00**

Tabla 16.1 – Precios unitarios de mano de obra

16.2. Software

Software	Coste de la licencia (€)
MatlLAB R2017b (Licencia Estudiantil)	35,00
Orcad 17.2 Lite	0,00
Arduino (IDE) 1.8.9	0,00
Total	**35,00**

Tabla 16.2 – Precios unitarios de Software

16.3. Equipo de trabajo

Elemento individual	Concepto	Cantidad (uds.)	Coste unitario (€/ud.)	Coste Total (€)
HP PAVILION 590-A0000NS DT	Ordenador Personal (PC)	1	230,44	230,44
Windows 10 Home	Licencia Sistema Operativo	1	145	145,00
Total				**375,44**

Tabla 16.3 – Precios unitarios de elementos auxiliares

16.4. Producción y envío de los circuitos impresos

Elemento	Coste Total (€)
Fabricación placas PCB	19,20
Aduanas	16,00
IVA	10,03
Envío	12,76
Total	**57,99**

Tabla 16.4 – Precios unitarios del montaje

16.5. Componentes

Material para la implementación	Concepto	Cantidad (uds)	Coste unitario (€/u)	Coste total (€)
Arduino MEGA 2560 Rev. 3 [29]	Microcontrolador	1	33,58	33,58
Teclado matricial 4x4 [30]	Teclado	1	4,99	4,99
Pantalla LCD 16x2 HD44780 [31]	Pantalla	1	2,90	2,90
UNIROI Kit de Iniciación Arduino Básico (UA001) [32]	Kit con protoboard, resistencias, leds, cables y otros elementos pasivos.	1	14,99	14,99
SODIAL Potenciometro 6mm, 10k ohm Horizontal Variable Cermet UK [33]	Potenciómetro de ajuste	2	0,10	0,20
Transformador encapsulado PCB 12V 16VA [34]	Transformador	1	8,06	8,06
SW50-4G Disipador 8.6℃/W TO-220 [35]	Disipador de calor	2	0,95	1,90
EG1218 Interruptor Deslizante THT, 200 mA [36]	Interruptor Deslizanter	2	1,28	2,56
2KBP02M Diodo Rectificador 1 KV, 10 A 4 Pines [37]	Puente de diodos	1	1,31	1,31
L7812CT 19V a 35V de Entrada 12V, 1 A TO-220-3 [38]	Regulador lineal de tensión	1	0,38	0,38
LM317BTG 1.2V a 37V, 1.5A 12V, 1 A TO-220-3 [39]	Regulador lineal de tensión ajustable	1	0,54	0,54
Total				**75,60**

Tabla 16.5 – Precios unitarios de los componentes

16.6. Coste final

Elemento individual	Coste Total (€)
Mano de Obra	5700,00
Software	35,00
Elementos Auxiliares	375,44
Unidades de Obra	75,60
Diseño y Montaje	57,99
Total	**6244,03**

Tabla 16.6 – Presupuesto total

El presupuesto total está valorado en **6244,03 euros**, (**seis mil doscientos cuarenta y cuatro euros con tres céntimos**).