



**ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA**
Universidad de Córdoba



TRABAJO FIN DE GRADO
Grado en Ingeniería Electrónica Industrial
**DISEÑO E IMPLEMENTACIÓN DE UN ALGORITMO DE
CONTROL PARA UNA TORRE DE REFRIGERACIÓN**

Autor: Francisco Luque Luque
Director: Francisco Táboas Touceda



UNIVERSIDAD DE CÓRDOBA

INFORME DEL DIRECTOR DEL TRABAJO FIN DE GRADO

El alumno D. Francisco Luque Luque ha trabajado en la realización del Trabajo Fin de Grado de forma constante y con aprovechamiento.

El trabajo ha supuesto una dedicación constante a lo largo de los últimos meses, labores de diseño, consulta de documentación, normativa y planos, la obtención de resultados y la elaboración del documento final que ahora se presenta.

El alumno ha asistido regularmente a numerosas tutorías, mostrando un interés y una capacidad de trabajo sobresalientes. Ha demostrado responsabilidad en todas las etapas del trabajo y ha sabido resolver con eficacia todos los problemas que se le han presentado.

Por lo tanto puedo concluir que el alumno ha desarrollado un trabajo excelente en contenido y forma, y éste es apto para su presentación.

Córdoba, a 7 de Septiembre de 2017

El Director

El Alumno

D. Francisco Táboas Touceda

D. Francisco Luque Luque

*Agradecer a Francisco Táboas haberme brindado
la posibilidad de participar en este proyecto.*

*Agradecer a mis padres su apoyo, que ha hecho
posible que me dedique a esta entusiasmante área.*

*Dedicarle a mi hermana este trabajo, que le sirva
de motivación en la nueva etapa que comienza.*

Contenido

Memoria descriptiva

1	Introducción.....	9
1.1	Torre de refrigeración	9
1.1.1	Componentes de la torre	9
1.1.2	Principio de funcionamiento	10
1.1.3	Sistema de control y adquisición de datos	11
1.1.4	Objetivo de la torre de refrigeración.....	11
2	Objetivos.....	12
3	Antecedentes.....	13
4	Descripción de la solución adoptada	14
4.1	Concepto de retraso del ángulo de disparo	14
4.2	Descripción del hardware del sistema	15
4.2.1	Detector de paso por cero	15
4.2.2	Actuador sobre motor.....	16
4.2.3	Placa Arduino Nano.....	17
4.2.4	Sistema en conjunto.....	18
4.2.5	Listado de componentes	21
4.3	Descripción del software del sistema.....	21
4.3.1	Interfaz de usuario del Puerto Serie.....	21
4.3.2	Protección de las bombas por los sensores de nivel	24
4.3.3	Modo de control automático y manual.....	24
4.3.4	Comunicación I2C.....	24
4.3.5	Monitorización de las comunicaciones	25
4.3.6	Estructuración del código.....	25
5	Desarrollo de la solución.....	27

Memoria justificativa

1	Introducción.....	31
2	Detector de paso por cero	31
2.1	Circuito integrado H11AA1.....	31
2.2	Componentes pasivos	32
2.2.1	Resistencia de entrada (R1).....	32
2.2.2	Resistencia de salida (pull-up R2).....	32
3	Actuador sobre motor.....	34

3.1	Círcuito integrado MOC3021	34
3.2	TRIAC BTA16.....	34
3.3	Componentes pasivos	35
3.3.1	Resistencia de entrada (R1).....	35
3.3.2	Resistencia de salida (R2)	35
3.3.3	Círcuito amortiguador (R3 y C1).....	36
3.4	Disipador térmico.....	39
4	Cálculos	42
4.1	Etapa de entrada al microcontrolador	42
4.1.1	R1	42
4.1.2	R2	43
4.2	Etapa de salida del microcontrolador	43
4.2.1	R1	43
4.2.2	R2	43
4.2.3	Círcuito amortiguador (R3 y C1).....	44
4.2.4	Potencia a disipar por R2 y R3.....	45
4.3	Disipador térmico.....	46
4.3.1	Cálculos previos.....	46
4.3.2	Cálculos del disipador.....	46
4.3.3	Elección del disipador.....	46
5	Resultados.....	47
5.1	Círcuito de entrada	47
5.1.1	Valores	47
5.1.2	Esquemático.....	47
5.1.3	Experimentación	47
5.2	Círcuito de salida.....	48
5.2.1	Valores	48
5.2.2	Esquemático.....	50
5.2.3	Experimentación	50
5.3	Disipador térmico.....	52
6	Presupuesto	53
7	Desarrollo del software de control	54
7.1	Evolución temporal del algoritmo de control.....	54
7.2	Comunicación I2C.....	55
7.3	Puerto Serie	55
7.3.1	Identificación del problema	55

7.3.2	Almacenamiento de mensajes en ROM	56
7.3.3	Compilación condicional	56
7.4	Cálculo del tiempo de disparo.....	57
7.5	Linealización de los motores	58
7.5.1	Linealización de la potencia	58
7.5.2	Linealización de la sinusode de la red (50Hz)	59
7.5.3	Linealización del ventilador.....	62
7.5.4	Linealización de la bomba de agua.....	63
7.5.5	Bomba de rociado	65
7.6	Controladores PID	65
7.6.1	Implementación de los PIDs	65
7.6.2	Discretización de PID.....	66
7.6.3	Sintonización de los PIDs.....	66
8	Estudio de la comunicación	67
8.1	Problema de comunicación.....	67
8.2	Unificación de Arduinos	67
8.2.1	Arduino para caudalímetros.....	67
8.2.2	Arduino para temperaturas.....	68
8.2.3	Arduino Sensores	68
8.2.4	Propuesta de unificación.....	68
9	Revisión de los objetivos.....	69
10	Conclusiones	69
	Referencias	70

ANEXOS:

- Referencias de la tabla de presupuestos
- Análisis de la implementación de la librería PID
- Software desarrollado para Arduino Control

ESQUEMAS

Esquema 1: Sistema de adquisición de datos y de control	11
Esquema 2: Esquemático del circuito detector de paso por cero	15
Esquema 3: Esquema para materializar el circuito detector de paso por cero sobre placa de prototipo.....	16
Esquema 4: Esquemático del circuito para actuar sobre un motor	16
Esquema 5: Esquema para materializar el circuito diseñado para variar la potencia entregada a carga monofásica	17
Esquema 6: Esquemático del circuito detector de paso por cero.....	47
Esquema 7: Esquemático para actuar sobre cargas conectadas a una red monofásica mediante un microcontrolador	50
Esquema 8: Circuito equivalente para triac en conducción (izquierda) y triac en NO conducción (derecha).....	50

ILUSTRACIONES

Ilustración 1: Componentes de la torre de refrigeración e interacción entre ellos.....	9
Ilustración 2: Un ciclo de la tensión en la red eléctrica (izquierda) y semiperiodo de un ciclo (derecha) con datos de tensión	14
Ilustración 3: Concepto de retraso en el disparo de la conducción.	14
Ilustración 4: Arduino nano. Imagen obtenida de la página web de arduino (Arduino, 2017) ..	18
Ilustración 5: Lateral izquierdo (imagen superior izquierda), lateral derecho (imagen superior derecha) y lateral inferior de la caja de control.	18
Ilustración 6: Depuración de la caja de control.....	19
Ilustración 7: Ejemplo de la interfaz de usuario por Puerto Serie al producirse la inicialización	23
Ilustración 8: Ejemplo de consigna automática y de bomba bloqueada por la lectura del sensor de nivel	23
Ilustración 9: Circuito integrado H11AA1 internamente (Vishay Semiconductors, 2011)	31
Ilustración 10: Tiempo de propagación de la señal en función de la resistencia de carga del colector (izquierda). Circuito de prueba para la medición del tiempo de propagación (derecha) (Vishay Semiconductors, 2011).....	33
Ilustración 11: Circuito integrado MOC3021, estructura interna (Texas Instruments, 1998)	34
Ilustración 12: Potencia disipada en el componente BTA16 en función de la corriente (STMicroelectronics, 2010)	40
Ilustración 13: Diagrama del montaje componente-disipador y circuito térmico equivalente (Trujillo, Pozo, & Triviño, 2011).....	41

Ilustración 14: Frontal, lateral y vista superior del disipador del circuito comercial (Variador de potencia comercial, 2017).....	42
Ilustración 15: pulso durante el paso por cero (izquierda) y ampliación (derecha).....	47
Ilustración 16: niveles lógicos en Arduino.....	48
Ilustración 17: tensión en bornes de la carga, en la placa de control, para el ventilador. Se muestra desde retraso en disparo de 0 – conducción continua- (esquina izquierda superior) hasta 8 milisegundos (esquina derecha inferior), pasando por: 1,5, 3, 5 y 7 milisegundos.....	51
Ilustración 18: Retraso en el disparo en función de la potencia deseada.....	59
Ilustración 19: Área encerrada en un semiperíodo de un ciclo de la sinusoides de la red eléctrica	59
Ilustración 20: Tiempo de retraso en el disparo en función del área requerida	60
Ilustración 21: Comparativa de la variación del área encerrada en la sinusoides para variaciones del ángulo de disparo de medio milisegundo en las regiones cercanas al paso por cero.....	61
Ilustración 22: Comparativa de la variación del área encerrada en la sinusoides para variaciones del ángulo de disparo de medio milisegundo en la región intermedia (tiempo de disparo desde 4 a 5,5 milisegundos).	61
Ilustración 23: Excitación-respuesta del ventilador	63
Ilustración 24: Región de respuesta lineal del ventilador	63
Ilustración 25: Excitación-respuesta de la bomba de agua a refrigerar.....	64
Ilustración 26: Región de respuesta lineal de la bomba de agua a refrigerar.....	64

Memoria descriptiva

1 Introducción

Se dispone de una torre de refrigeración experimental, de circuito cerrado y ampliamente instrumentalizada. Actualmente, está dotada de varios sensores para la evaluación de temperaturas (6 sensores), caudales (2 sensores), humedad relativa (4 sensores), medidas de presión diferencial (6 sensores) y nivel (2 sensores). Éstos están colocados en puntos de interés de la torre, permitiendo así la monitorización de las variables objetivo y el análisis energético de éstas.

1.1 Torre de refrigeración

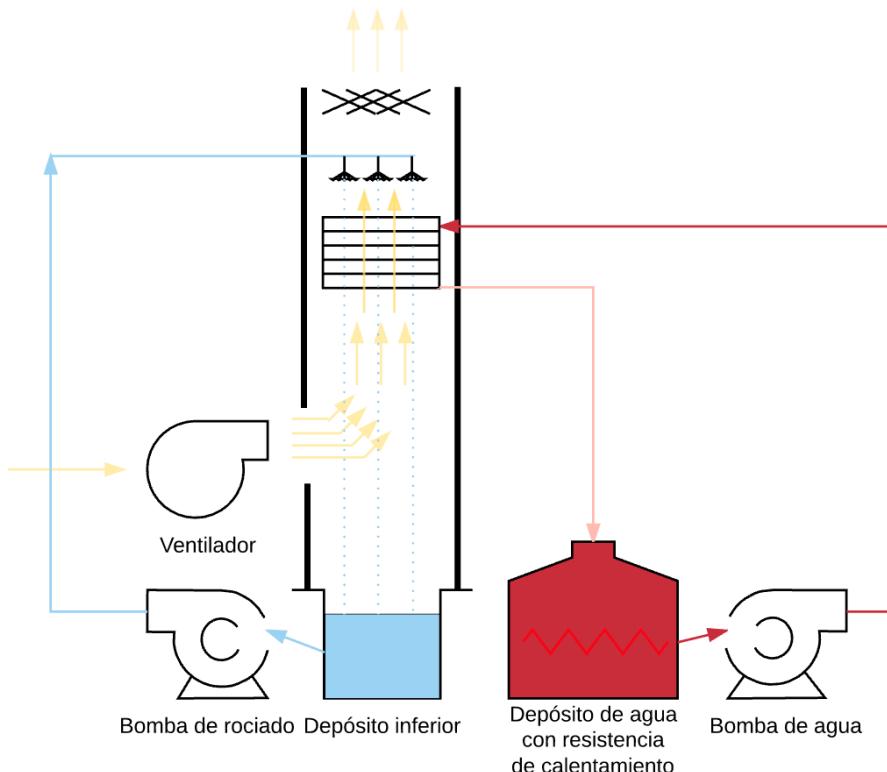


ILUSTRACIÓN 1: COMPONENTES DE LA TORRE DE REFRIGERACIÓN E INTERACCIÓN ENTRE ELLOS.

La torre de refrigeración, como su nombre bien indica, es utilizada para enfriar un fluido. Para explicar el principio de funcionamiento, primero se van a introducir los componentes que la integran y su función.

1.1.1 Componentes de la torre

- **Ventilador:** Introduce aire del exterior de la torre al interior de ésta. Para ello existen dos disposiciones básicas, conocidas como tiro forzado (o mecánico) y tiro inducido, dependiendo de si el ventilador está colocado en la parte inferior de la torre (entrada de aire a la torre) o en la parte superior (salida de aire de la torre), respectivamente. Como puede observarse en la Ilustración 1, en la torre de refrigeración se dispone de tiro forzado, pues el ventilador está colocado en la entrada de aire. Posee una potencia eléctrica de 490W.

- **Bomba de rociado, depósito inferior y rociadores:** Desde el depósito inferior se bombea agua a la parte superior de la torre. Desde allí, es dispersada, mediante rociadores, por todo el interior de la torre. El agua va cayendo, por efecto de la gravedad, hacia el depósito inferior del que fue extraída. Como puede apreciarse, su sentido es a contracorriente del flujo de aire.
La bomba de rociado tiene una potencia eléctrica de 190W.
- **Resistencia de calentamiento:** Se ubica en el depósito de agua. Es gobernada por un relé de estado sólido, y su objetivo es producir el calentamiento, de forma controlada, del agua a refrigerar. Mediante su control se va a poder estudiar el “acercamiento a bulbo húmedo”, que mide la eficiencia de la torre de refrigeración.
Posee una potencia eléctrica de 2kW.
- **Bomba de agua:** Extrae el agua caliente del depósito y la bombea hacia un sistema de tuberías en el interior de la torre. Es en éste donde se produce el intercambio de calor.
Tiene una potencia eléctrica de 450W.
- **Eliminador de gotas:** Está situado en la parte superior de la torre, en la salida de aire. Su función es evitar que salgan gotas de agua en el flujo de aire saliente, reteniéndolas y haciéndolas caer nuevamente hacia el interior de la torre.

1.1.2 Principio de funcionamiento

El enfriamiento se produce por dos fenómenos de transferencia de calor: por convección y transferencia de masa por difusión.

- **Transferencia de calor por convección:**

Debido a que el aire que se introduce en la torre está a diferente temperatura del agua rociada y de las tuberías, se produce una transferencia de calor. Este fenómeno es el de menor importancia en la torre, siendo el siguiente el que más influencia tiene en la transferencia de calor.

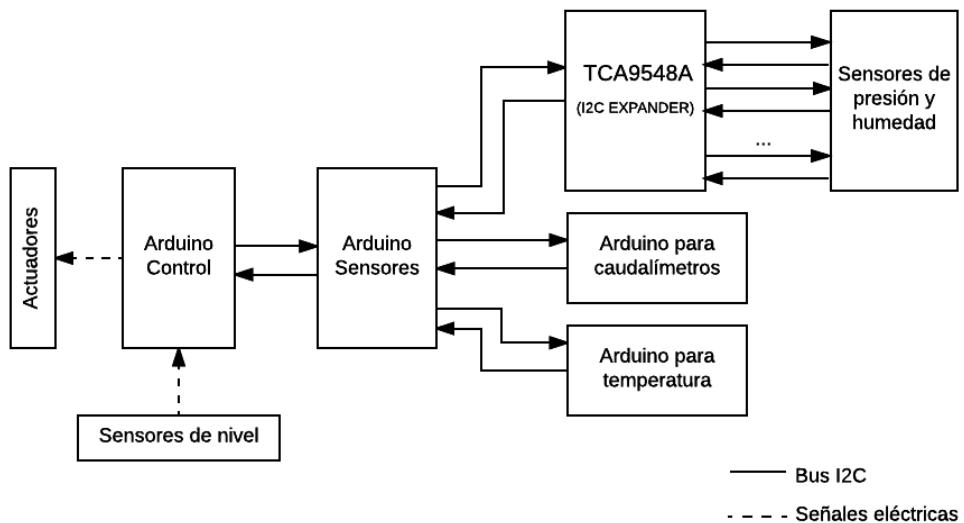
- **Transferencia de calor por difusión:**

Se produce entre el agua rociada y el aire. Es producto de una diferencia de concentración de vapor de agua. El aire que entra a la torre es más seco que el aire que sale, que contiene mayor humedad. Se produce, por tanto, una transferencia de masa del agua al aire. El agua, para su evaporación, necesita un aporte energético. Como no hay ninguna fuente que lo aporte, lo toma de sí misma (de su energía interna), reduciendo su temperatura. Para aumentar la eficiencia, se ha de tener la mayor superficie de contacto con el aire posible. Es por ello que el conjunto de tuberías, por las que circula el fluido a refrigerar, tienen un diámetro pequeño y, colocadas en serpentín, una longitud total muy grande, logrando crear una gran superficie de contacto, tanto con el agua rociada (que se deposita en forma de gotas y película delgada sobre ellas) como con el líquido de su interior. El agua depositada se enfria al evaporarse, reduciendo la temperatura de la tubería y ésta del líquido que circula por su interior.

1.1.3 Sistema de control y adquisición de datos

Hasta la realización del trabajo fin de grado, todo el gabinete de los actuadores era de tipo manual. En el caso de las bombas de rociado y de fluido, la regulación era del tipo todo o nada. En el caso del ventilador se tenía, mediante un banco de relés y resistencias, un control de velocidad discreto (de 16 velocidades).

La plataforma hardware en la que se basa el sistema para realizar la adquisición de datos y el control es Arduino. A continuación se muestra un esquema con las placas que conforman dicho sistema:



ESQUEMA 1: SISTEMA DE ADQUISICIÓN DE DATOS Y DE CONTROL

La comunicación entre las diferentes placas Arduino y con algunos de los sensores se realiza mediante un bus I2C. En éste, la maestra es la placa denominada “Arduino Sensores”, actuando el resto como esclavos. Este diseño permite gran modularidad, separando en varios controladores Arduino las diferentes funcionalidades implementadas, permitiendo la extracción y reconexión de las mismas con suma facilidad y mínima modificación del software.

1.1.4 Objetivo de la torre de refrigeración

La torre ha sido diseñada y construida por el profesor Francisco Táboas Touceda. Es el siguiente paso a un estudio teórico que ha estado realizando sobre la eficiencia energética en las torres de refrigeración, aportando un enfoque diferente al que se ha seguido tradicionalmente. Con ella pretende demostrar la validez de su teoría y marcar un precedente en el diseño de las torres de refrigeración.

Tanto las pérdidas de carga, como los coeficientes de transferencia de calor y masa dependen, fundamentalmente, de los números adimensionales de Reynolds y Prandtl que, a su vez, dependen de la velocidad del flujo y de su temperatura (respectivamente).

Con la realización del trabajo fin de grado se pretende poder controlar las velocidades de los diferentes flujos, para así lograr el número de Reynolds deseado en cada uno de ellos, permitiendo el análisis, en un conjunto variado de puntos de trabajo, de la torre.

2 Objetivos

Se realizará un diseño para el control de la torre de refrigeración. Éste tiene que tener total integración con el sistema actual, y será implementado a través de *Arduino Control* (Esquema 1).

Se diseñará un circuito que, mediante una señal controlada por un microcontrolador, actúe sobre cada uno de los motores, para conseguir variar la potencia entregada.

Por otro lado, se escribirá un programa de control que actúe tanto de forma manual (estableciendo el porcentaje de trabajo) como de forma automática (especificando el parámetro de consigna) para cada uno de los motores. Para ello se deberá gestionar, de forma efectiva, la información de los sensores y el control de la planta dentro de la estructura actual, que está planteada con la incorporación de una placa Arduino exclusivamente para el control.

Objetivos definidos en el anteproyecto:

- Diseño e implementación del algoritmo de control para la torre de refrigeración. Se pretende que este algoritmo:
 - Sea capaz de modificar la potencia transmitida a cada uno de los motores y de accionar la resistencia de calentamiento.
 - Admita la introducción de valores manuales (porcentaje de funcionamiento de cada actuador).
 - Implemente un método de control para seguimiento de referencias.
 - Sea totalmente compatible con el sistema actual (ha de realizarse su integración en él).
- Identificación de los problemas de comunicación actuales:
 - Propuesta (e implementación si procede) de un método de comunicación más eficaz para evitar que el control de la torre quede bloqueado.
 - Propuesta (e implementación si procede) de mejora en la estrategia del sistema de control.

3 Antecedentes

Debido a que el autor no había trabajado con la plataforma Arduino con anterioridad, la realización de este trabajo fin de grado comenzó con una búsqueda de información acerca de la misma. Esta búsqueda preliminar se centró en su página oficial (Arduino, 2017).

Tras ella, los esfuerzos se centraron en el primer objetivo: ser capaz de modificar la potencia transmitida a los motores. Para ello se recurrió a la asignatura *Electrónica de Potencia* cursada en tercer curso, concretamente a los apuntes y al libro (J.Escudero) seguido durante la misma. En este punto se definió la estrategia a seguir: Modificar la potencia entregada a los motores desfasando la entrada en conducción a partir del paso por cero de la señal de red.

Con este objetivo, se realizó una búsqueda, orientada a software, en el subdominio de *Arduino Playground* (Arduino Playground, 2017) dentro de la web oficial (Arduino, 2017). En la sección de librerías se encontró una dedicada a la *modulación PWM basada en el paso por cero* (AC Zero Cross PWM Library, 2017). Sin embargo, el enlace que conduce a la misma no estaba disponible debido a que al autor había retirado el contenido. Se encontró otro recurso, dentro de *Arduino Playground*, que podía servir como referencia para el proyecto: Un control para resistencias basado en paso por cero (AC Phase Control, 2017).

Por otro lado, y aunque no se encontró mucho acerca del software necesario, las búsquedas orientadas al hardware sí fueron algo más fructíferas. Sin ir más lejos, en el recurso anteriormente mencionado (AC Phase Control, 2017) ya se incluye un circuito hardware para la detección del paso por cero y la actuación. La información proporcionada por éste se complementó con la obtenida de otro proyecto (Kotsopodis, 2017) que sí estaba orientado a motores (no como en el caso anterior, sólo a calentador y por tanto a carga resistiva). Además, el control de velocidad del ventilador que estaba implementado en la torre, estaba basado en un circuito comercial (Variador de potencia comercial, 2017), en el cual se había sustituido el potenciómetro por el banco de relés, por lo que también se realizó ingeniería inversa para obtener su esquemático.

Respecto a la implementación de los PID, ésta se ha basado en una librería (Beauregard, 2017) obtenida del subdominio *Playground* de *Arduino* (Arduino Playground, 2017).

4 Descripción de la solución adoptada

En este apartado se van a explicar los principios de funcionamiento del sistema que se ha diseñado e implementado en la torre. Para ello, se mostrará el circuito hardware y sus características y, posteriormente, las del software desarrollado para la plataforma Arduino. Como esta memoria es de tipo descriptivo, en ningún momento se abordarán los cálculos realizados ni se justificarán las decisiones tomadas (para ello se debe consultar la memoria justificativa).

4.1 Concepto de retraso del ángulo de disparo

La energía eléctrica que se distribuye a través de la red eléctrica monofásica, está formada por una sinusode (tensión alterna) de 50Hz y tensión eficaz 230V (pico de 325V).

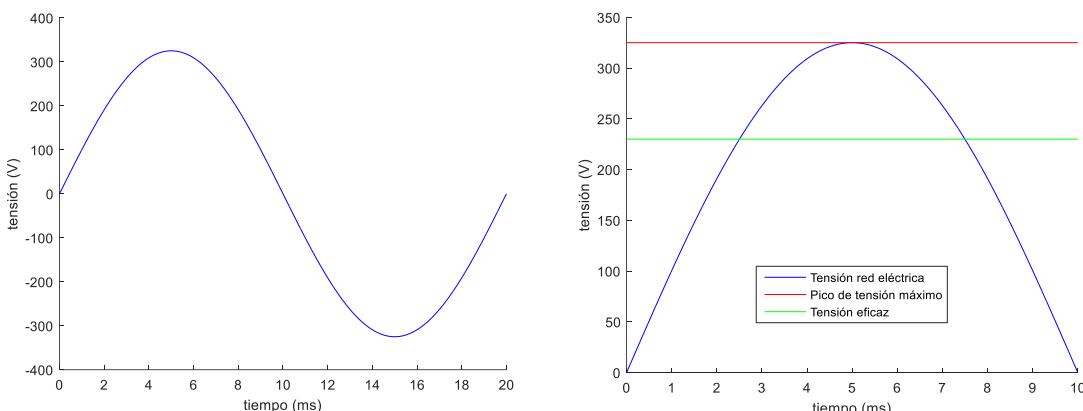


ILUSTRACIÓN 2: UN CICLO DE LA TENSIÓN EN LA RED ELÉCTRICA (IZQUIERDA) Y SEMIPERÍODO DE UN CICLO (DERECHA) CON DATOS DE TENSIÓN

Cuando una carga está conectada a la red, está sometida a esta tensión alterna, cuyas características de tensión pico y tensión eficaz se representan en la Ilustración 2. Con el objetivo de controlar la potencia entregada a los motores (y así modificar la variable sobre la actuación, como es la velocidad del aire o el número de Reynolds del agua), se pretende modificar la tensión que se les suministra. Para ello se va a constar de dos circuitos, uno capaz de detectar el paso por cero de la sinusode de la red eléctrica y otro que, tras recibir una señal de control, permita la conducción del motor. Así, una vez detectado el paso por cero puede temporizarse la cantidad de tiempo de retraso deseada y, tras ello, permitir la conducción del motor, modificando la tensión eficaz que éste recibe. Este concepto se visualiza en la Ilustración 3, donde se contrasta la cantidad de energía recibida (área azul) de una onda completa y una con retraso en disparo.

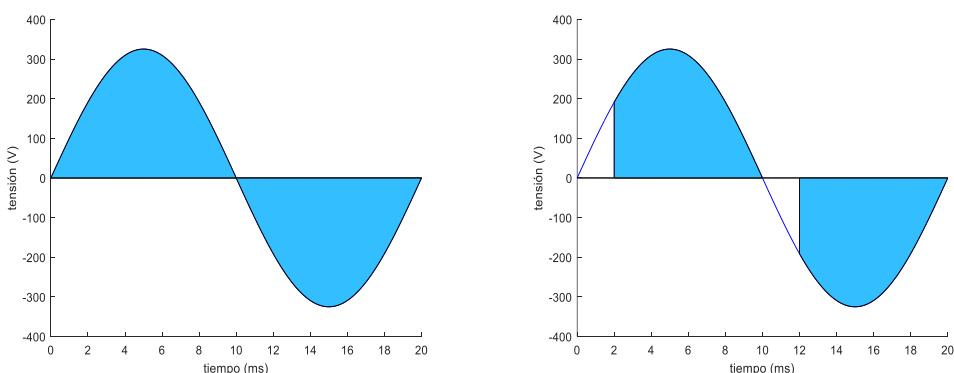


ILUSTRACIÓN 3: CONCEPTO DE RETRASO EN EL DISPARO DE LA CONDUCCIÓN.

4.2 Descripción del hardware del sistema

Es el encargado de efectuar el control de la potencia entregada, actuando como interfaz entre la red eléctrica y los motores. Se distinguen tres partes bien diferenciadas:

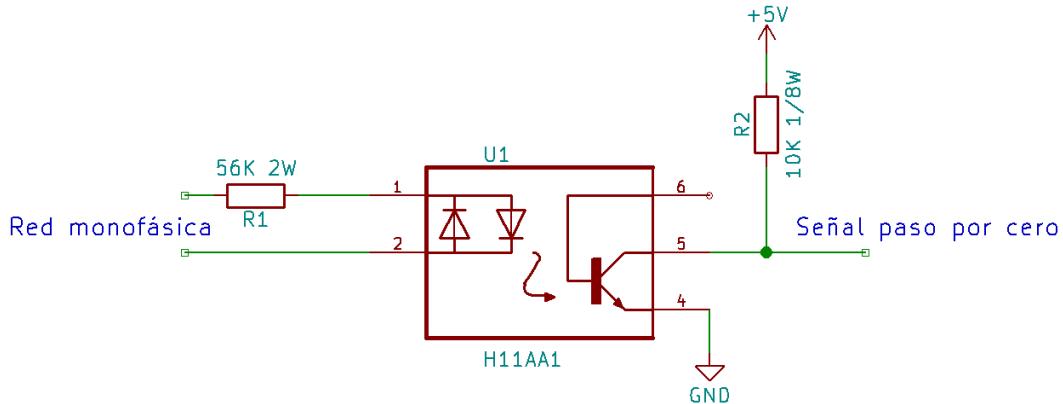
- Detector de paso por cero.
- Actuadores sobre los motores.
- Placa Arduino Nano.

Tanto el detector como los actuadores han sido materializados en placas de tipo matriz, sobre las que se han soldado los componentes y las pistas. Para que el diseño sea completamente modular, todos los circuitos se han elaborado por separado. Para cada circuito se mostrará su esquemático y el esquema de conexión de la placa de prototipo. Éste esquema ha sido elaborado considerando el tamaño real de los elementos (por ejemplo la longitud de las resistencias, variable en función de su potencia de disipación), para su correcto emplazamiento.

4.2.1 Detector de paso por cero

Su función es mandar un pulso positivo al microcontrolador cada vez que se detecta el paso por cero de la sinusoide de la red eléctrica, actuando como pasarela entre ésta y los niveles de tensión que maneja el microcontrolador. Parámetros:

- Entrada: Red monofásica, a través de los terminales indicados.
- Salida: Señal de paso por cero. Se genera un pulso positivo durante el paso por cero.

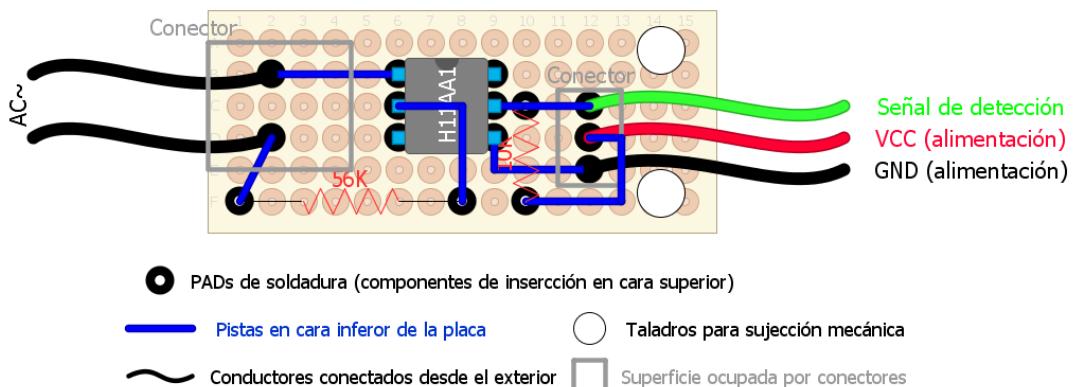


ESQUEMA 2: EQUÍVOCO DEL CIRCUITO DETECTOR DE PASO POR CERO

Lo que caracteriza a la señal de paso por cero, aparte de ser un pulso positivo (la señal es estática a 0V y activa a 5V durante el pulso), es que se activa antes del paso por cero de la sinusoide, y cesa después de este paso, coincidiendo la mitad de la duración del pulso con el paso por cero (aproximadamente). Como esta señal está pensada para que actúe sobre una interrupción externa de un microcontrolador, ésta ha de ser configurada para la detección del flanco de bajada.

A continuación se muestra el esquema que se ha elaborado para realizar la materialización del detector de paso por cero sobre placa de prototipo.

DETECTOR DE PASO POR CERO

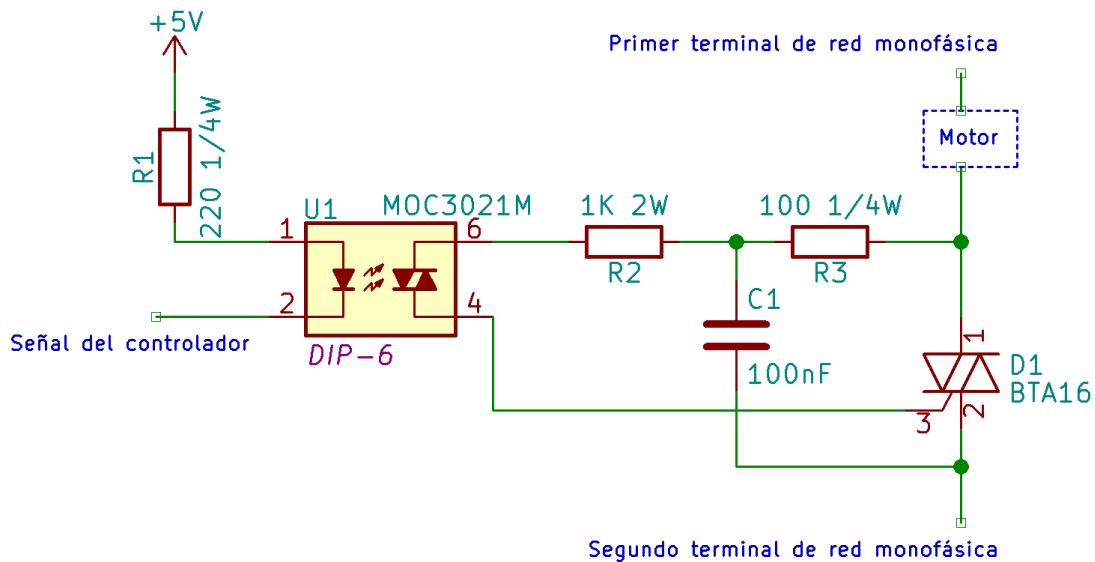


ESQUEMA 3: ESQUEMA PARA MATERIALIZAR EL CIRCUITO DETECTOR DE PASO POR CERO SOBRE PLACA DE PROTOTIPO.

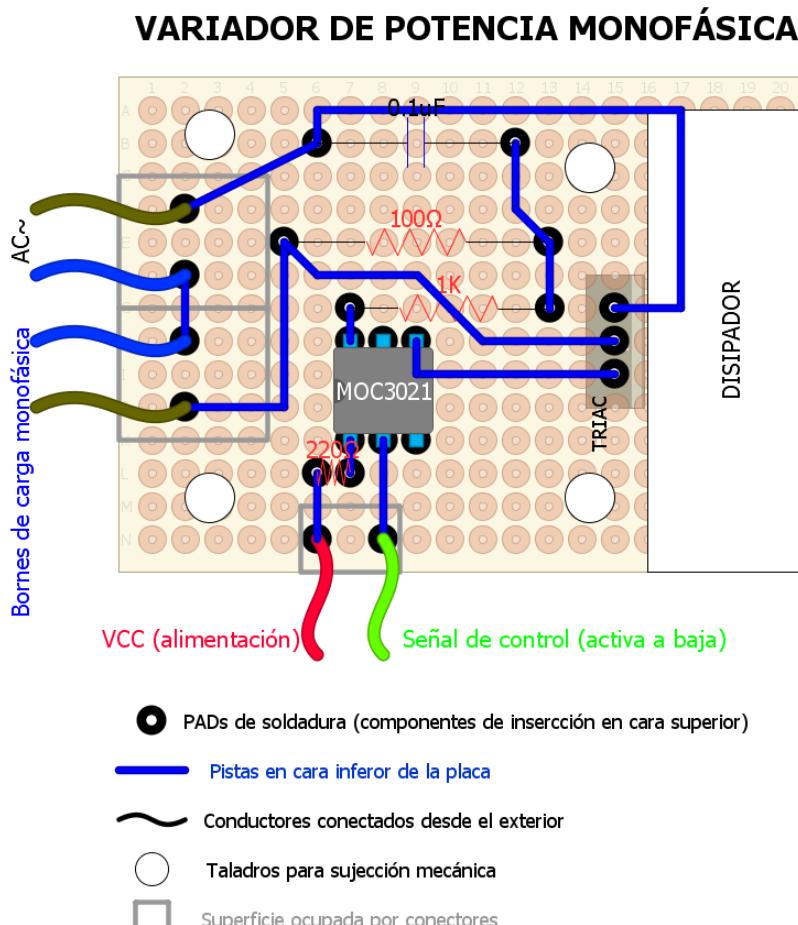
4.2.2 Actuador sobre motor

Este circuito actúa como interfaz entre los niveles de tensión del microcontrolador y la red eléctrica. Así, mediante una señal de niveles discretos 0-5V, se puede atacar a cargas de mucha más potencia conectadas a 230V. Parámetros:

- Entrada: Señal de control del microcontrolador. Es activa a baja (0V).
- Salida: La carga que se pretende controlar ha de estar conectada en serie con los terminales de la red eléctrica monofásica.



ESQUEMA 4: ESQUEMÁTICO DEL CIRCUITO PARA ACTUAR SOBRE UN MOTOR



ESQUEMA 5: EQUÍVOCO PARA MATERIALIZAR EL CIRCUITO DISEÑADO PARA VARIAR LA POTENCIA ENTREGADA A CARGA MONOFÁSICA

Nota: En el Esquema 5, la resistencia de 220Ω (R_1) está colocada de forma vertical.

El circuito está diseñado para ser capaz de operar con convección natural (sin refrigeración forzada) a temperaturas ambiente de 60°C (interior de caja eléctrica junto a otros actuadores) para cargas que demanden, al 100%, una intensidad de $3,5\text{A}$, esto es, para cargas de hasta 800W .

4.2.3 Placa Arduino Nano

Arduino es la plataforma escogida para el desarrollo del software de control. El software desarrollado para el control se encuentra descrito en el apartado 4.3 Descripción del software del sistema. Los pines que están siendo utilizados son:

- Pin D2: Interrupción externa para detectar el paso por cero.
- Pin D4: Señal de control del ventilador.
- Pin D5: Señal de control de la bomba de agua.
- Pin D6: Señal de control de la bomba de rociado.
- Pin D7: Señal de control de la resistencia de calentamiento.
- Pin D10: Señal del sensor de nivel del tanque de agua (fluido a refrigerar).
- Pin D11: Señal del sensor de nivel inferior del tanque de rociado.
- Pin D12: Sin implementar. Se ha dejado adaptado para una futura implementación de sensor de nivel superior del tanque de rociado.

- Pin D13: Led incorporado en Arduino Nano. Utilizado para señalizar que la resistencia de calentamiento está activa.
- PIN A4: Señal SDA de comunicación I2C.
- PIN A5: Señal SCL de comunicación I2C.

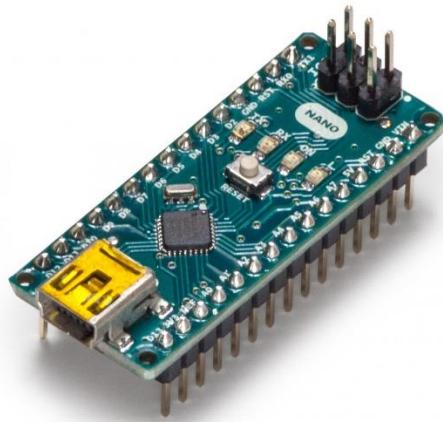


ILUSTRACIÓN 4: ARDUINO NANO. IMAGEN OBTENIDA DE LA PÁGINA WEB DE ARDUINO (ARDUINO, 2017)

4.2.4 Sistema en conjunto

Se muestran fotografías de la caja de control. En el momento de su captura, la tapa estaba abierta y el conexionado aún no era definitivo, pues se estaba realizando la depuración del control con la placa conectada al ordenador (puede apreciarse el cable de conexión USB). En total, se ha utilizado una Arduino nano y se han elaborado tres actuadores sobre motores y un detector de paso por cero, como se aprecia en la Ilustración 6.



ILUSTRACIÓN 5: LATERAL IZQUIERDO (IMAGEN SUPERIOR IZQUIERDA), LATERAL DERECHO (IMAGEN SUPERIOR DERECHA) Y LATERAL INFERIOR DE LA CAJA DE CONTROL.

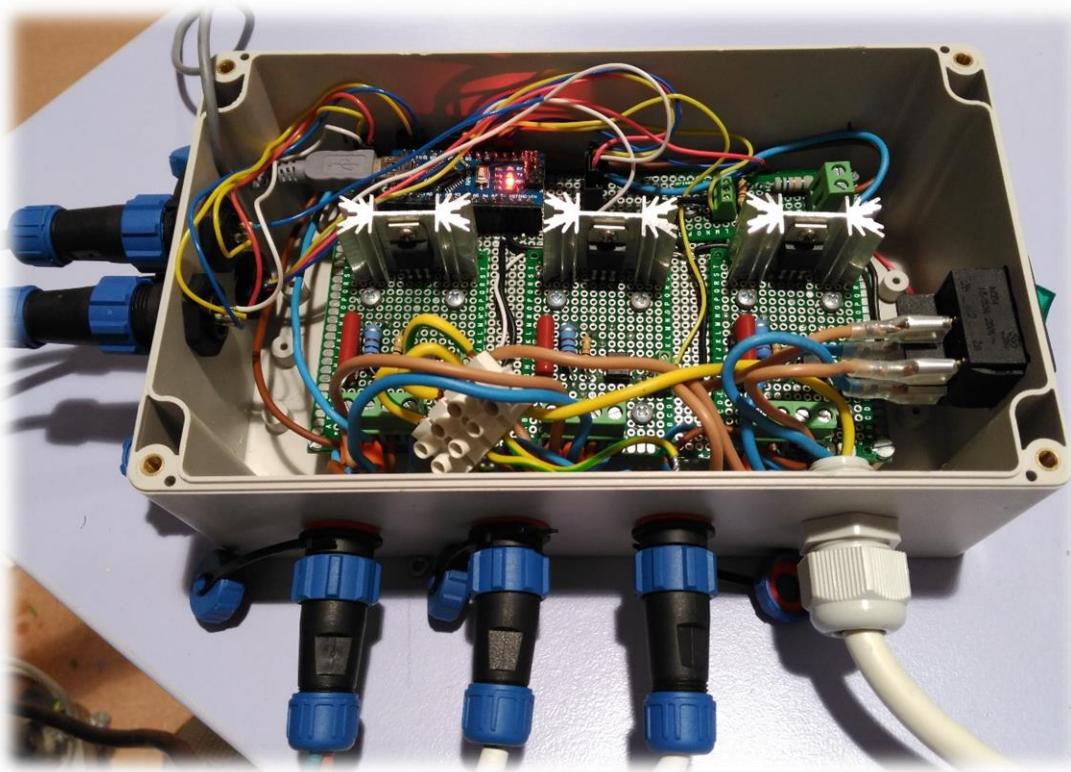


ILUSTRACIÓN 6: DEPURACIÓN DE LA CAJA DE CONTROL.

Como se comentó anteriormente, tanto el detector como los actuadores han sido elaborados sobre placas de tipo matriz, sobre las que se han soldado los componentes y las pistas. Para que el diseño sea completamente modular, todos los circuitos se han materializado por separado y su fijación se ha realizado mediante tornillos, para que puedan ser fácilmente extraídos o reemplazados.

Para la placa Arduino, se ha dispuesto de un zócalo de inserción, para que también pueda ser extraída de la caja. La programación de ésta se realiza a través del puerto mini-B USB del que dispone, mediante el entorno de desarrollo integrado (*IDE, Integrated Development Environment*) de Arduino, disponible para su descarga desde la web oficial (Arduino, 2017). Para cargar el programa, la Arduino puede ser extraída de la caja de control, o bien, puede introducirse el cable directamente dentro de la misma, sin necesidad de extracción previa, pues se ha dejado espacio suficiente a tal efecto (véase Ilustración 6). ATENCIÓN: Por seguridad, la manipulación de cualquier elemento interno de la caja ha de hacerse siempre con ésta desconectada de la red eléctrica.

La caja de control, como va a estar instalada en la torre de refrigeración, se ha confeccionado de modo que sea resistente a salpicaduras. A tal efecto, se le ha dotado de conectores aislantes de agua y humedad. Cada conector, en función de su distribución, tiene la siguiente funcionalidad:

- Lateral izquierdo: Sensores de nivel, alimentación y comunicación I2C.
 - Conector superior trasero (4 conexiones): Alimentación y comunicación I2C.
 - Alimentación de Arduino:
 - Borne 2 (internamente blanco): GND.
 - Borne 3 (internamente amarillo): V_{IN} (5V estables).
 - Comunicación I2C:
 - Borne 1 (internamente rojo): Señal SDA.
 - Borne 4 (internamente azul): Señal SCL.
 - Conector superior delantero (2 conexiones): Control resistencia de calentamiento.
 - Borne 1 (internamente amarillo): GND.
 - Borne 2 (internamente azul): Señal de control de la resistencia.
 - Conector inferior (5 conexiones): Sensores de nivel.
 - Borne 1 (internamente rojo): Alimentación (V_{IN}) para los sensores.
 - Borne 2 (internamente blanco): GND para los sensores
 - Borne 3 (internamente amarillo): Sensor de nivel inferior de agua (fluido a refrigerar).
 - Borne 4 (internamente azul): Sensor de nivel inferior del depósito de rociado.
 - Borne 5: Sin implementar. Posibilidad de añadir sensor de nivel superior del depósito de rociado, permitiendo el relleno de agua automático. Para ello habría que realizar la instalación a una conexión de agua a la torre así como de una válvula de paso todo o nada regulada electrónicamente, y modificar el software de control para añadir esta nueva característica.
- Zona inferior:
 - Primer conector (izquierda): Bomba de agua.
 - Segundo conector (medio): Bomba de rociado.
 - Tercer conector (derecha): Ventilador.

Todos ellos disponen de 3 conexiones y se han realizado de la misma forma:

 - Borne 1: Conducto azul del motor.
 - Borne 2: Conducto marrón del motor.
 - Borne 3: Toma de tierra del motor.
 - Conexión del extremo derecho: Conexión a la red eléctrica monofásica.
- Lateral derecho: Se ha dispuesto de un interruptor que aísla la alimentación de la red y el resto de componentes de la caja de control.

4.2.5 Listado de componentes

Se muestra una tabla resumen de los componentes necesarios para los circuitos detector de paso por cero y actuador sobre motor.

<u>Componente</u>	<u>Unidades</u>
Detector de paso por cero	
56kΩ 2W ±5%	1
10kΩ 1/8 W ±5%	1
H11AA1	1
Actuador sobre motor	
220Ω 1/4W ±5%	1
1kΩ 2W ±5%	1
100Ω 1/4W ±5%	1
0,1µF 400V	1
BTA16-600BW	1
Disipador térmico	1
MOC3021	1

4.3 Descripción del software del sistema

Es el encargado de controlar las placas hardware mediante las señales descritas en el apartado 4.2.3, así como de implementar el resto de características, que se explican a continuación. Se ejecuta sobre la placa Arduino Nano instalada en la caja de control.

4.3.1 Interfaz de usuario del Puerto Serie

Se ha desarrollado una interfaz de usuario mediante la cual se puede efectuar el control de la torre a pie de planta, mediante conexión USB y el software libre de Arduino.

Para ello, se abre el programa de control de la torre (abriendo cualquier fichero .ino de los que lo conforman) y realiza la siguiente selección a través del menú: *Herramientas > Placa "Arduino/Genuino" > Arduino Nano*. Tras realizar la conexión USB con la placa Arduino, ha de seleccionarse en *Herramientas > Puerto > Puerto al que está conectada la placa*.

Para evitar compilación y ejecuciones innecesarias cuando no se dispone de conexión serie, se ha estructurado el programa con compilación condicional, eliminándose todo el código que se encarga de gestionar la comunicación por Puerto Serie. Cuando el programa esté cargado y funcionando de forma definitiva en la torre, sólo se lleva a cabo la comunicación por I2C con la placa *Arduino Sensores*, encargada de comunicarse con el exterior para enviar y recibir datos.

Se habilita el modo depuración para tener acceso a la interfaz del Puerto Serie. Para ello se descomenta la siguiente línea de código dentro del fichero *configuración_software.h*:

```
// CONFIGURACIÓN DEL SISTEMA
#ifndef MODO_DEPURACION 1 // Si está descommentado se habilita comunicación por Puerto Serie
```

Tras ello se recompila el programa y se carga a la placa, pulsando sobre *Sketch > Subir*.

Una vez cargado, se abre el monitor serie (*Herramientas > Monitor Serie*) para tener acceso a la interfaz de usuario. Hay que seleccionar la velocidad de comunicación, ajustándola a 115200 baud, en la esquina inferior derecha de la ventana del monitor serie.

Nota: La interfaz se ha desarrollado en inglés por continuidad con la interfaz MatLab que ya tenía desarrollada el proyecto de la torre, cuyo menú también está en inglés.

Tras la realización de los pasos anteriores, se obtiene la imagen mostrada a continuación (Ilustración 7), donde se diferencian tres partes:

- Valores actuales (*current values*): Se muestra el estado de cada actuador de la torre.
 - Control configurado en modo automático (mediante el PID) o manual. Cuando está configurado en modo automático también se muestra la consigna/referencia marcada al PID (véase Ilustración 8). En el caso de que la bomba esté bloqueada por la lectura del sensor de nivel (que se haya alcanzado el nivel mínimo), no aparece ni modo automático ni manual, sino bloqueada (*blocked*, véase Ilustración 8).
 - Potencia del actuador (rango 0 a 100%), tanto si está en modo manual (se muestra la que se ha establecido) como si está actuando el PID (se muestra el % de la señal de control que manda el PID al actuador).
 - Medición del sensor correspondiente.
 - Velocidad del aire en el caso del ventilador.
 - Reynolds en el caso del agua de rociado y agua a refrigerar.
 - Acercamiento (*approach*) en el caso de la resistencia de calentamiento.
 - Medición de los sensores de nivel del depósito de rociado y de agua ("0" indica que se ha alcanzado el nivel mínimo, "1" que el nivel es correcto).
- Comandos de control disponibles: Se muestran los comandos disponibles. Es indiferente su introducción en minúsculas o mayúsculas. En los comandos en los que sólo se necesita una letra se indica "*no second command needed*". En los que sí requieren una segunda letra, también requerirán de un número (pues se estará indicando la potencia o la consigna).
- Mensaje de ejemplo de introducción de comandos. Se muestra la pauta correcta para introducir comandos (por ejemplo, si se quiere poner el ventilador en modo automático con una referencia de velocidad de 3.2, habría que introducir "*af*" y "*3.2*").

```

Current values:
Fan:           Manual
              Pot: 0.00%  Medicion: 0.00
Spraypump:     Manual
              Pot: 0.00%  Medicion: 0.00
Heating:       Manual
              Pot: 0.00%  Medicion: 0.00
Waterpump:     Manual
              Pot: 0.00%  Medicion: 0.00
Level sensors:
Bottom level Spray sensor: 0  Bottom level Water sensor: 0
Commands:
First command: Mode selection
'S': Show the actual values (no second command needed)
'H' Approach (no second command needed)
'A' Mode: automatic
'M' Mode: manual
'R' Stop all controls (no second command needed)
Second command: Motor selection
'F' Fan power
'W' Waterpump power
'S' Spraypump power
'H' Heating control

Waiting for user input:

```

EXAMPLE: Fan in manual mode with a reference of 50%: 'MF50'
 Instead of the input showed below, is recommended the next: 'MF' '50'
 (what means, write MF and send it. Then do the same with the value 50).

ILUSTRACIÓN 7: EJEMPLO DE LA INTERFAZ DE USUARIO POR PUERTO SERIE AL PRODUCIRSE LA INICIALIZACIÓN

Si se produce la introducción incorrecta de algún comando, se muestra un mensaje de error y, de nuevo, la lista de comandos disponibles.

```

Current values:
Fan:      Automatic      Consigna: 3.00
          Pot: 58.00%  Medicion: 0.00
Spraypump:   Manual
          Pot: 0.00%  Medicion: 0.00
Heating:    Manual
          Pot: 0.00%  Medicion: 0.00
Waterpump:  BLOCKED      Consigna: 0.00
          Pot: 0.00%  Medicion: 0.00
Level sensors:
Bottom level Spray sensor: 1  Bottom level Water sensor: 0

```

ILUSTRACIÓN 8: EJEMPLO DE CONSIGNA AUTOMÁTICA Y DE BOMBA BLOQUEADA POR LA LECTURA DEL SENSOR DE NIVEL

Los valores actuales (*current values*), se actualizan cada vez que se produce uno (o varios) de los siguientes eventos:

- Se produce una actualización de las variables de proceso (se han recibido nuevos datos a través de I2C con mediciones).
- Se produce una actualización de los parámetros control. Ésta puede venir dada por:

- Nueva recepción de consignas (por puerto serie/I2C).
- Nueva recepción de valores manuales (a través de puerto serie/I2C).
- Recálculo de las potencias de los actuadores, por parte de los PIDs, para aproximar o mantener la variable de proceso al valor marcado por la consigna.

4.3.2 Protección de las bombas por los sensores de nivel

La función de los sensores de nivel es indicar que los depósitos, de los cuales las bombas extraen fluido, están a un nivel bajo. Son sensores de todo o nada, dando como lectura un “1” si el nivel es superior a donde está colocado el sensor, y “0” si es inferior.

Cuando los sensores indiquen un “0”, la bomba ha de ser detenida. Pero debido a que están midiendo el nivel de un líquido en un depósito, la superficie del mismo tiene oscilaciones. Cuando el nivel de líquido en el depósito está cercano a la posición del sensor, éste comenzará a dar lecturas oscilantes entre “0” y “1”.

Para evitar el problema de activación y desactivación constante de las bombas debido al problema descrito en el párrafo anterior, se ha desarrollado una función en el software que monitoriza el nivel actual de los sensores y sus estados anteriores. Así, permite establecer un margen de histéresis tras la detección de cambio de nivel. Este margen se especifica mediante el símbolo *HISTERESIS_PROTECCION_BOMBAS* declarado en *configuracion_software.h*. Mediante él se indica el número mínimo de milisegundos (por ejemplo, 5000ms, equivalentes a 5 segundos) que la bomba permanecerá en cada estado, es decir, que si se detecta un nivel bajo (“0”), la bomba se parará durante los milisegundos indicados por *HISTERESIS_PROTECCION_BOMBAS*. Tras este tiempo, se volverá a comprobar el nivel del sensor: si está a nivel bajo (“0”) la bomba permanecerá apagada, pero si está a nivel alto (“1”) se volverá a activar con los parámetros de control que tenía antes de producirse la desactivación por parte del sensor de nivel. Si durante el tiempo que ha permanecido desactiva se reciben (por I2C o Puerto Serie) nuevos parámetros de control (modo automático o manual, consigna...) éstos serán actualizados internamente y, cuando la bomba esté de nuevo en condiciones operativas (el sensor indique “1”), se activará de acuerdo a los mismos.

4.3.3 Modo de control automático y manual

La incorporación de controles automáticos mediante PIDs permite que la torre trabaje de forma independiente, sin la necesidad de un supervisor humano que manipule las variables de forma manual. Así, se pueden programar diferentes puntos de trabajo, y el software de control se encargará de establecer la potencia necesaria en los actuadores para alcanzarlos y mantenerlos, rechazando perturbaciones.

4.3.4 Comunicación I2C

Es la encargada de garantizar la integración con el resto de la planta. La placa *Arduino de Control* está conectada a un bus I2C, ejerciendo como esclava (*Arduino Sensores*, en el Esquema 1, es la que ejerce como maestra del bus). A petición de *Arduino Sensores*, envía y recibe datos.

4.3.4.1 Peticiones de envío de datos

Se realizan dos tipos de envío:

- Envío de datos de control (mayoritario): Se envían los modos de los PID (automático/manual), el nivel de los sensores, y la potencia de cada actuador.
- Envío de consignas (minoritario): Se envían los modos de los PID (automático/manual), el nivel de los sensores, y las consignas actuales de los PID de los diferentes actuadores. Esto ocurre cada 5 envíos de datos de control.

4.3.4.2 Recepción de datos

De nuevo se tienen dos tipos de recepción:

- Recepción de datos del proceso: Se reciben las variables de velocidad, números de Reynolds y acercamiento.
- Recepción de parámetros de control: Se reciben las consignas de los PIDs o el porcentaje de potencia de los motores manuales.

4.3.5 Monitorización de las comunicaciones

Tanto en el caso de la comunicación por puerto serie como por I2C, se produce una monitorización por parte de *watchdogs* (o *perros guardianes*), para añadir seguridad:

- En el caso del puerto serie, si se produce una espera entre la introducción de comandos, de más del tiempo indicado por *TIEMPO_WATCHDOG_PS* (en milisegundos), se cancela la introducción.
- En el caso de I2C, si no se produce la recepción o petición de envío de datos durante más del tiempo (en ms) indicado por *TIEMPO_WATCHDOG_COMUNICACION_I2C*, se realiza la parada de todos los motores, pues se entiende que hay un fallo crítico en la comunicación con *Arduino Sensores*.

4.3.6 Estructuración del código

El código se ha dividido en diferentes ficheros, según su funcionalidad (todo el código, dividido en los respectivos ficheros, está incluido como anexo). Así, los ficheros resultantes de la división son:

- *vX_Y_arduino_control.ino*: Versión del software X.Y. Es el fichero principal, que contiene las funciones características de Arduino *setup()* y *loop()*, y a partir del cual el IDE de Arduino realiza la compilación.
- *configuracion_software.h*: Esta cabecera de librería es incluida en el fichero anterior. En ella se incluyen las librerías utilizadas y se declaran los símbolos, las variables globales y los mensajes almacenados en la memoria ROM del microcontrolador.
- *calculo_valores.ino*: En éste se encuentra el código que compone las funciones dedicadas al cálculo de los valores de disparo en función de la potencia introducida para cada motor.
- *comunicacion_I2C.ino*: Contiene el código relacionado con la comunicación I2C, así como los eventos de envío y recepción.
- *configuraciones.ino*: En él se declara la función que configura e inicializa los pines y la que se encarga del reseteo de las variables.

- *control_PID.ino*: En este fichero se encuentran las funciones relacionadas con las inicializaciones de todos los PIDs y la discretización del PID de la resistencia.
- *interrupciones.ino*: En ella se encuentran definidas las actuaciones al detectarse una interrupción externa (el paso por cero) e interrupción por temporización hardware (del *timer1*, utilizado para temporizar el retraso del ángulo de disparo).
- *puerto_serie.ino*: Todas las funciones relacionadas con la gestión del envío y recepción por Puerto Serie. El fichero entero es compilado sólo si se está en modo depuración, lo cual se consigue descomentando la definición de *MODO_DEPURACION* dentro de la cabecera *configuracion_software.h*.
- *sensores.ino*: Código de lectura de sensores de nivel y protección de las bombas.
- *temporizador.ino*: Funciones relacionadas con el temporizador 1 programadas para realizar las temporizaciones necesarias, obtenidas tras el cálculo oportuno, para retrasar el disparo y así conseguir la potencia requerida.

5 Desarrollo de la solución

Este apartado es la continuación de uno anterior: Antecedentes. En él se explican los primeros pasos seguidos y se hace referencia a las fuentes y trabajos que se han tomado como base para la solución hardware adoptada. Sin embargo, en estas referencias no se hace ninguna mención a los cálculos de los valores de los componentes mostrados en los respectivos esquemáticos que aparecen en ellas, e incluso faltaban datos importantes, como la potencia disipada por cada componente, límites de potencia máximos soportados por el circuito, conveniencia de utilizar disipadores térmicos acoplados a los TRIAC (encargados de gobernar la carga eléctrica), etc. A pesar de ello, le sirvieron al autor para obtener una visión global: número y tipo de componentes a utilizar.

Primero, se decide realizar el circuito hardware. Se comienza buscando componentes, alternativos a los usados en los antecedentes, que puedan aportar mejoras al proyecto (por ejemplo, el uso del circuito integrado H11AA1, en lugar del 4N25, que hace innecesario el uso de un circuito rectificador).

Tras disponer de todos los componentes se realizan pruebas sobre una placa de prototipo. Una vez comprobado el funcionamiento, se diseña la disposición de los componentes en las placas de tipo matriz, y se lleva a cabo su soldadura.

Para la comprobación del funcionamiento, fue necesario escribir y depurar la primera versión software del controlador. Esta versión era muy básica, y sólo se encargaba de detectar el paso por cero y, tras él, actuar sobre un motor con la potencia establecida a través del puerto serie. A esta primera versión le sigue la inclusión de varios motores, con el recálculo de tiempos cada vez que se modifica una potencia, y temporizaciones relativas.

Tras los pasos anteriores, el software se vuelve más complejo, por lo que, para tener el código organizado según su funcionalidad, se lleva a cabo la división en ficheros. Se incorpora la comunicación I2C para integración con el resto de la planta, y se siguen añadiendo características a la comunicación del puerto serie. Se añaden funciones de seguridad del tipo *watchdog* para ambas comunicaciones (serie e I2C). Posteriormente, se incorpora la lectura de los sensores y se añade la protección, de tipo margen de histéresis temporal, a las bombas. Se incorpora la resistencia de calentamiento y los PIDs, y se efectúan las linealizaciones de potencia de los motores. Por último, en lo que respecta a software, se lleva a cabo una reestructuración completa del código y renombrado de funciones y variables para esclarecer su papel en el código.

Finalmente, se adapta una caja eléctrica para incorporar los circuitos de actuación, detección y la placa controladora.

Memoria justificativa

1 Introducción

Los antecedentes mencionados en la memoria descriptiva sirvieron como base para la elaboración de la solución hardware. En éstos se ofrecen esquemáticos, pero no se justifica, en ninguno, la elección de los componentes utilizados. Con el desarrollo de esta memoria se pretende justificar la elección de todos los componentes (cuyos valores finales difieren de los que emplean los esquemáticos utilizados como base), así como explicar los aspectos técnicos del software de control desarrollado.

2 Detector de paso por cero

La etapa de entrada al microcontrolador, que se encargará de realizar las temporizaciones y mandar las señales de control, consiste en la detección del paso por cero de la senoide de la red eléctrica. Ésta le será transmitida mediante un pulso positivo a través del terminal indicado como “señal paso por cero”.

2.1 Circuito integrado H11AA1

La elección de este componente se ha basado en las siguientes características (Vishay Semiconductors, 2011):

- Entrada bidireccional, por lo que permite detectar cada paso por cero de la señal, a diferencia de otros opto-acopladores, como los de la familia 4N25 (Vishay Semiconductors, 2010) empleada en el proyecto al que se referencia en los antecedentes (Kotsopoulos, 2017), cuya entrada unidireccional hacen necesaria la incorporación de un puente rectificador a su entrada.
- Encapsulado DIP estándar de 6 pines que permite su incorporación a la placa que se elabore e instale en la planta de refrigeración.
- Alto valor de aislamiento (5,3 kV pico) que permite la protección del microcontrolador y del resto de componentes conectados a éste.

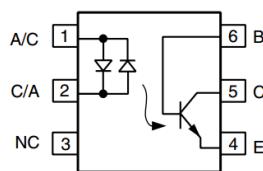


ILUSTRACIÓN 9: CIRCUITO INTEGRADO H11AA1 INTERNAMENTE (VISHAY SEMICONDUCTORS, 2011)

Internamente el componente consta en la entrada de dos LEDs infrarrojos conectados en paralelo y enfrentados entre sí. Ambos están acoplados a la base de un fototransistor conectado a la salida. Cuando la senoide de la red eléctrica (que estará conectada entre los terminales 1 y 2 mediante la fase activa y el neutro) pase por cero, el fototransistor dejará de conducir, por lo que, conectando el terminal cinco mediante una resistencia pull-up a una fuente de tensión, y el terminal 4 a GND, se consigue un pulso a alta en cada paso por cero.

2.2 Componentes pasivos

2.2.1 Resistencia de entrada (R1)

El objetivo de esta resistencia, referenciada en el esquemático como R1, es limitar la corriente de entrada al circuito integrado *H11AA1*. Se ha dimensionado con un valor elevado para disminuir la pérdida de potencia que se produce en la misma. De la hoja de características (Vishay Semiconductors, 2011) se obtienen los siguientes valores en la entrada del circuito integrado *H11AA1*:

- Corriente máxima admitida: $\pm 60\text{mA}$.
- Potencia de disipación máxima: 100mW .
- Caída de tensión máxima en el diodo¹ (conducción directa, corriente de prueba inferior a la que circula en el circuito diseñado): $1,5\text{V}$

Los valores de entrada actuarán de forma restrictiva para el dimensionamiento de la resistencia, estableciendo el valor mínimo que ha de tener. De todos modos, como se pretende que la pérdida de potencia en ella sea lo más pequeña posible, se le dará un valor grande que cumplirá holgadamente estos requisitos. El cálculo de los valores de intensidad en el circuito diseñado se realiza de la siguiente forma²:

$$I_{rms, \text{entrada } H11AA1} = \frac{V_{rms}}{R} \quad I_{\text{máx.,entrada } H11AA1} = \frac{V_{rms}\sqrt{2}}{R}$$

Una vez conocidos estos valores se puede realizar el cálculo de la potencia disipada en el circuito integrado:

$$P_{H11AA1, \text{entrada}} = I_{rms, \text{entrada } H11AA1} \cdot V_{\text{caída diodo } H11AA1}$$

Y en la resistencia:

$$P_{R1} = (I_{rms, \text{entrada } H11AA1})^2 \cdot R_1$$

2.2.2 Resistencia de salida (pull-up R2)

La función de esta resistencia es actuar como elevadora de tensión en la salida. Así, mientras el fototransistor de la etapa de salida del circuito integrado *H11AA1* no esté saturado, se tendrá un nivel de 5V en el extremo que lo conecta con la resistencia, y cuando sí lo esté se tendrá una tensión máxima de $0,4\text{V}$ (Vishay Semiconductors, 2011).

Como se ha comentado anteriormente, la generación del pulso se basa en que los diodos de entrada del circuito integrado *H11AA1* dejen de conducir. Esto provoca que el pulso se produzca antes del paso por cero y cese después del mismo, pues los diodos cortarán su estado de

¹ Tanto para este cálculo como para el resto de la memoria, los valores escogidos de las hojas de características son los que ofrecen una mayor restricción para el diseño. Así pues, en este caso, es el máximo ($1,5\text{V}$), y no el típico ($1,2\text{V}$), ya que con el primero el valor de potencia a disipar será superior, y es con éste con el que se ha de realizar el diseño (por su mayor restricción), pues el diodo del componente que se utilice para materializar el circuito puede tener esta caída de tensión máxima en lugar de la típica, y uno de los objetivos del diseñador es garantizar la operatividad, durabilidad y seguridad del circuito diseñado.

² No se ha tenido en cuenta la caída de tensión que se produce en el diodo de la etapa de entrada del circuito integrado *H11AA1*, debido a su pequeña magnitud en comparación con la tensión de red.

conducción cuando la tensión en sus extremos sea inferior a 1,5V (Vishay Semiconductors, 2011).

Debido a las características de generación del pulso descritas, interesa que el controlador sea sensible a la detección del flanco de bajada de la señal generada por el detector de paso por cero. Así, se asegura que la actuación se realiza tras el paso por cero de la senoide, y no antes. Con este objetivo en mente, se pretende hacer un diseño en el que el retardo de la fase baja a alta de la señal sea pequeño o moderado (para asegurar que se alcanza cuanto antes y así tener una anchura de pulso adecuada) y, por otro lado, que el retardo de fase alta a fase baja sea el mínimo posible, para que tras el paso por cero se produzca la detección del flanco de bajada y, por tanto, comience la actuación por parte del microcontrolador lo antes posible.

En base a la Ilustración 10, que se muestra a continuación, el mínimo valor admisible para la resistencia de salida R_2 sería de $5\text{k}\Omega$, valor para el cual se produce el mínimo t_{pHL} (tiempo de propagación para la tensión de salida V_O del estado de alta a estado de baja). Por otro lado, t_{PLH} , valor opuesto al anterior (esto es, de 0V a 5V), ofrece un tiempo aceptable, en torno a $40\mu\text{s}$.

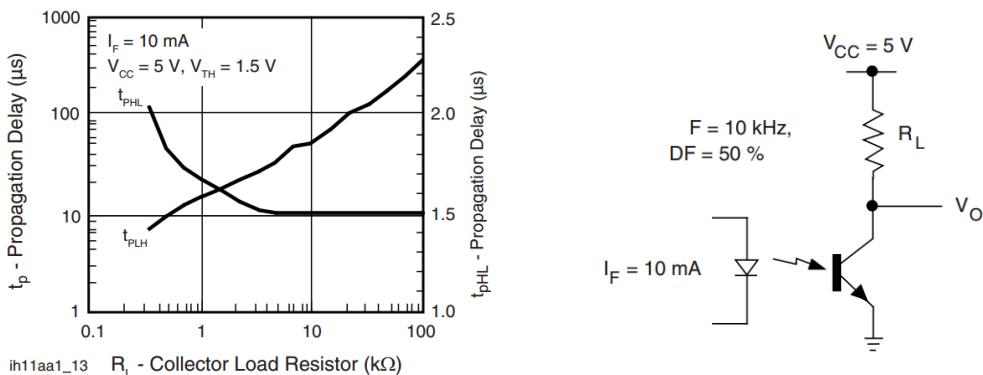


ILUSTRACIÓN 10: TIEMPO DE PROPAGACIÓN DE LA SEÑAL EN FUNCIÓN DE LA RESISTENCIA DE CARGA DEL COLECTOR (IZQUIERDA).
CIRCUITO DE PRUEBA PARA LA MEDICIÓN DEL TIEMPO DE PROPAGACIÓN (DERECHA) (VISHAY SEMICONDUCTORS, 2011)

Debido a que en el circuito diseñado se tendrá una corriente inferior a la del circuito de prueba de la ilustración anterior, así como al carácter sinusoidal de la tensión empleada (en lugar de pulsada) y frecuencia enormemente inferior, la irradiación del LED disminuirá de forma progresiva, permitiendo una variación de corriente menos agresiva, hasta su entrada en corte, y por tanto disminuyéndose el retardo de propagación t_{PLH} respecto al mostrado en la Ilustración 10. Por ello, la resistencia elegida es de $10\text{k}\Omega$, pues ofrece unos tiempos de propagación adecuados (según lo anteriormente expuesto, mínima propagación en el flanco de bajada y aceptable en el de subida) así como una disipación de potencia muy reducida. La intensidad que circulará por la etapa de salida (desde la fuente de 5 V hasta tierra pasando por la resistencia y el transistor), así como la potencia disipada en el transistor (potencia disipada en la salida del circuito integrado) y la potencia disipada en la resistencia son:

$$I_{rms,salida\ H11AA1} = \frac{V_{caída\ R2}}{R_2} = \frac{V_{alimentación} - V_{transistor,saturación}}{R_2}$$

$$P_{H11AA1,salida} = I_{rms,salida\ H11AA1} \cdot V_{transistor,saturación}$$

$$P_{R2} = (I_{rms,salida\ H11AA1})^2 \cdot R_2$$

3 Actuador sobre motor

La etapa de salida del microcontrolador es la encargada de actuar sobre el motor a través de un TRIAC (BTA16). Para su gobierno se emplea un opto-acoplador (MOC3021) que recibe la señal de control del microcontrolador (ésta es activa a baja) y se la comunica a la base del BTA16. El circuito amortiguador, conformado por una resistencia y un condensador, cumple la función de limitar la variación de tensión, con el objetivo de evitar el disparo de los TRIACs (tanto del BTA16 como el de salida del opto-acoplador MOC3021) debido a una elevada tasa de cambio de la tensión en sus bornes.

3.1 Circuito integrado MOC3021

Es un opto-acoplador que dispone de un diodo infrarrojo y un TRIAC cuya base está ópticamente acoplada a este. Entre sus aplicaciones más típicas se encuentra la de actuar de interfaz entre microcontroladores y periféricos de 115/240 V_{AC}, pues su TRIAC de salida soporta tensiones pico de hasta 400V y ofrece aislamiento para tensiones de hasta 7,5kV.

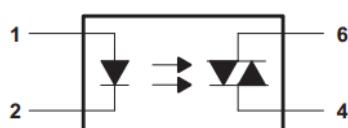


ILUSTRACIÓN 11: CIRCUITO INTEGRADO MOC3021, ESTRUCTURA INTERNA (TEXAS INSTRUMENTS, 1998)

Otra aplicación típica es el control de motores, que es para lo que se utilizará en este diseño. Su TRIAC de salida no está pensado para gobernar directamente cargas, sino para actuar sobre la puerta de otro TRIAC externo, que sí sea capaz de manejar mayor potencia y que esté conectado en serie con la carga. El integrado elegido, por su inmediata disponibilidad, ha sido del fabricante Texas Instruments y, a partir de sus características (Texas Instruments, 1998), se realizan los cálculos de las resistencias R₁, R₂ y del circuito amortiguador, R₃ y C₁ (véanse estas referencias en el Esquema 7).

3.2 TRIAC BTA16

Es el actuador, encargado de permitir la conducción a través del motor (está conectado en serie con él), al aplicar una intensidad a su puerta. El componente elegido es el BTA16-600BW. Éste dispositivo es una versión *snubberless*, por lo que no necesita circuito amortiguador, ya que presenta muy buenas características en cuanto a las tasas de variación de tensión en sus extremos. Sin embargo, en nuestro diseño sí que se empleará un circuito amortiguador, pues es necesario para evitar el auto-disparo del TRIAC de la etapa de salida del MOC3021. A parte de buenas características en tasa de variación, presenta valores adecuados en ratios máximos soportados y en sus características eléctricas:

- Corriente rms en estado de conducción de 16A. Aunque para la aplicación se requiere un máximo de 3,2A durante el arranque, el sobredimensionamiento ayuda a evitar un calentamiento excesivo (véanse los criterios térmicos en el apartado de Introducción de este documento). Además, también presenta, en el encapsulado TO-220AB que se pretende utilizar, la arquitectura adecuada para añadir un dissipador térmico, y como es de la serie BTA, tiene la pestaña aislada eléctricamente del resto del componente.
- Pico de tensión repetitiva máxima soportada (en directo e inverso) (STMicroelectronics, 2008): mínimo de 600 V.

- Utilización especialmente recomendada en cargas inductivas (debido a su alto rendimiento en conmutación) y para aplicaciones de desfase de ángulo.

3.3 Componentes pasivos

3.3.1 Resistencia de entrada (R1)

Limita la corriente de entrada al LED infrarrojo. De la hoja de características (Texas Instruments, 1998) se tiene que la máxima corriente continua soportada es de 50 mA, y la corriente que produce la activación del TRIAC de salida (parámetro I_{FT}) es de 15 mA. Por tanto, se tiene que la condición para activarlo es:

$$15mA \leq I_{R1} \leq 50mA$$

Y la corriente que circula por la misma es calculada de la siguiente forma:

$$I_{R1} = I_{\text{entrada},MOC3021} = \frac{V_{\text{alimentación}} - V_{\text{caída,diodo entrada MOC}}}{R_1}$$

Donde la alimentación será de 5V y la caída de tensión (máxima) en el LED infrarrojo de 1,5V.

Una vez conocida la intensidad se puede calcular la potencia disipada, tanto en la etapa de entrada del integrado:

$$P_{MOC3021,\text{entrada}} = I_{\text{entrada},MOC3021} \cdot V_{\text{caída,diodo entrada MOC}}$$

Como en la resistencia:

$$P_{R1} = (I_{\text{entrada},MOC3021})^2 \cdot R_1$$

3.3.2 Resistencia de salida (R2)

A la salida se tiene conexión a 230V en alterna, por lo que los valores de corriente serán mucho mayores que los de la entrada. Una vez que se manda la señal de control para que se produzca la activación del TRIAC que controla al motor, el TRIAC de salida del MOC3021 conduce durante un breve periodo de tiempo (pues tras la activación del BTA16, éste presenta una resistencia interna mucho menor a R_2 y la conexión de ambos es en paralelo). Por tanto, debido a que este TRIAC (de salida del MOC3021) sólo soportará pulsos de corriente, el parámetro clave para el cálculo de R_2 es el “pico de corriente no repetitiva, en estado de conducción, para el controlador de salida” (*Output driver nonrepetitive peak on-state current*):

$$R_{2,\text{valor mínimo}} = \frac{V_{\text{pico}}}{I_{\text{máx}}}$$

Siendo éste el valor mínimo de resistencia admitido para no superar el pico máximo de corriente dado por la hoja de características (Texas Instruments, 1998).

Una vez establecido el límite inferior del valor de R_2 se analiza otro parámetro influyente en la misma: la corriente de puerta para el TRIAC BTA16. Éste necesita una corriente mínima de 2,5mA (según especifica hoja de características (STMicroelectronics, 2010), un 5% de la corriente de puerta máxima, 50 mA), y una tensión aplicada a la puerta (V_{GT}) de 1,3V, por lo que del valor de

R_2 depende que el TRIAC pueda disparar en un momento dado o no (pues no se cumplen los requisitos). La sinusode de la red eléctrica viene dada por:

$$V(t) = V_{pico} \cdot \sin(2\pi f \cdot t)$$

Si se establece que se pueda disparar a partir de una tensión de red x , se tiene que el mínimo retraso temporal tras el paso por cero a partir del que se puede disparar es de:

$$x V = V_{pico} \cdot \sin(2\pi f \cdot t) V$$

$$t_{disparo,mínimo} = \frac{\sin^{-1}\left(\frac{x V}{V_{pico}}\right)}{2\pi f}$$

Y para que se pueda disparar a una tensión de red de x V, el valor de R_2 máximo ha de ser:

$$x V = V_{R2} + V_{GT} = I_{G,mínima} \cdot R_2 + V_{GT}$$

$$R_{2,valor\ máx} = \frac{x - V_{GT}}{I_{G,mínima}}$$

Por otro lado, la corriente máxima que circularía a través del TRIAC de salida del MOC3021, ignorando el valor de R_3 (que será mucho menor que el valor de R_2) sería:

$$I_{máx.pulsada,MOC3021,salida} = \frac{V_{máx}}{R_2}$$

Esta corriente se aplica durante un periodo de tiempo inferior a la de 1,2A mostrada en la hoja de características (Texas Instruments, 1998), pues ésta es indicada con una frecuencia de repetición de 10ms y una duración de 1ms, y en el circuito diseñado no se aplicará más allá de decenas de microsegundos.

Respecto a la potencia disipada en el foto-TRIAC de la salida del MOC3021, su cálculo no es necesario si se cumple que la intensidad es de carácter pulsado y menor a la máxima de 1,2A mencionada con anterioridad.

3.3.3 Circuito amortiguador (R_3 y $C1$)

El circuito amortiguador es el encargado de suavizar la tasa de cambio y hacerla menor a la máxima tasa de cambio admitida por los componentes. Esta es (para cada componente sensible a tasa de cambio) la siguiente:

- BTA16BW: 1000 V/ μ s (STMicroelectronics, 2010)
- MOC3021: TRIAC en la salida, 100 V/ μ s (Texas Instruments, 1998)

De los valores anteriores se escoge el último, correspondiente MOC3021, por ser el más restrictivo. Para mayor seguridad, se establece en los cálculos que la máxima tasa de cambio admisible sea de **50 V/ μ s**.

Existen dos causas por las que la tensión a la que están sometidos estos componentes experimenta un ratio de cambio. La primera de ellas es el carácter sinusoidal de la tensión de red. La señal de la red eléctrica viene dada por:

$$V(t) = V_{pico} \cdot \sin(2\pi f \cdot t) V$$

Y la tasa de cambio se obtiene derivando respecto al tiempo la ecuación anterior. Ésta será máxima cuando se produzca el paso por cero de la sinusoida, punto en el que la pendiente es máxima:

$$\frac{dV(t)}{dt} = V_{pico} \cdot 2\pi f \cdot \cos(2\pi f \cdot t) \text{ V/s}$$

$$\left. \frac{dV(t)}{dt} \right|_{máx} = V_{pico} \cdot 2\pi f \cdot \cos(2\pi f \cdot t) \Big|_{t=0} \text{ V/s}$$

Por otro lado, en una aplicación donde se pretende el control de motores, que se caracterizan por ser cargas de tipo resistivo-inductivo, existe un desfase entre la tensión aplicada y la corriente que circula por ellos. Esta es la segunda de las causas que provoca un ratio de cambio de la tensión a la que están sometidos los componentes.

Una vez comenzada la conducción del TRIAC, tras haberse producido el disparo, éste no dejará de conducir hasta que se haya extinguido la corriente que circula por él. Si la carga fuese puramente resistiva, se tendría la certeza de que esto ocurriría en los pasos por cero de la tensión. Sin embargo, con carga de tipo resistivo-inductiva, dependerá del momento en el que se produzca el disparo. El peor de los casos, en el que la corriente se extinga cuando la sinusoida está en el punto de máxima amplitud, es cuando se producirá mayor tasa de variación de la tensión y es el empleado para el cálculo del circuito amortiguador.

Con la inclusión de R_3 y C_1 (circuito amortiguador) en el diseño, se tiene que la tasa de variación de la tensión, en el peor de los casos (que la corriente se extinga cuando la tensión aplicada es máxima), es (Fairchild Semiconductor, 2006):

$$\frac{dV(t)}{dt} = \frac{V_{pico}}{\tau} = \frac{V_{pico}}{R_3 C_1}$$

Por lo que

$$\tau_{deseada} = R_3 C_1 = \frac{V_{pico}}{(dV/dt)_{deseado}}$$

Para el cálculo del circuito amortiguador, una vez calculada la constante de tiempo deseada (en función del ratio de cambio máximo deseado), se fija uno de los valores (el de la resistencia o condensador) y se despeja el otro. Es aconsejable fijar el valor del condensador (por existir menos valores disponibles en el mercado y ser un componente de mayor costo) y calcular el valor de la resistencia a partir de éste.

$$R_3 = \frac{\tau_{deseada}}{C_1}$$

La impedancia que presenta el circuito amortiguador es:

$$Z_{R3,C1} = R_3 + \frac{1}{wC_1} = R_3 + \frac{1}{2\pi f \cdot C_1}$$

Como el circuito amortiguador está en paralelo con el actuador (TRIAC), sólo circulará corriente por el amortiguador cuando el actuador esté bloqueando el paso de corriente, pues cuando el TRIAC esté activado presentará una impedancia prácticamente nula. La corriente que circulará a través del circuito amortiguador, mientras el actuador está bloqueado, es:

$$I_{rms,circuito\ amortiguador} = \frac{V_{rms}}{Z_{R3,C1}}$$

Y la potencia disipada en la resistencia que forma parte del circuito amortiguador será de:

$$P_{R3,circuito\ amortiguador} = I_{rms,circuito\ amortiguador}^2 \cdot R_3$$

Sin embargo, por esta resistencia no solo circula corriente debida al circuito amortiguador, también circula por ella una corriente de mayor magnitud, en forma de pulso, durante la activación del TRIAC. Esta corriente circula desde la red eléctrica a través de la resistencia R_3 , posteriormente por la resistencia R_2 , seguidamente por el circuito integrado MOC3021 y finalmente por la puerta del TRIAC BTA16 hacia el otro terminal de la red eléctrica.

Debido a que la potencia disipada en la resistencia R_3 por la corriente del circuito amortiguador es muy pequeña, cabe esperar que la disipación de potencia total sea más restrictiva por la corriente debida al pulso de activación del TRIAC.

Conocidos el valor de R_2 y R_3 , se vuelve a calcular la intensidad máxima que circula a través del TRIAC de salida del MOC3021:

$$I_{máx,MOC3021} = \frac{V_{pico}}{R_2 + R_3}$$

Para calcular la potencia de disipación que produce esta corriente, hay que tener en cuenta que es de carácter pulsado. La potencia total a disipar en las resistencias, en caso de que fuese corriente continua, y considerando el peor de los casos (que se produzca el disparo, y por tanto el pulso de activación del TRIAC, en el momento en el que la sinusoides está en su valor pico de tensión) sería:

$$P_{R2,R3,continua} = V_{máx} \cdot I_{máx,MOC3021}$$

Y teniendo en cuenta su carácter pulsado:

$$P_{R2,R3,pulsada} = \frac{P \cdot t_{pulso}}{T}$$

Siendo:

- t_{pulso} la duración del pulso, es decir, el tiempo que tarda en activarse el TRIAC BTA16 desde que se aplica el pulso en su puerta. Se ha considerado un valor, extremadamente conservador, a falta de indicaciones en su hoja de características (STMicroelectronics, 2010), de 100μs, estimándose que el valor real debe de ser en torno a 1 orden de magnitud inferior.

- T el periodo de repetición. Como la sinusoide de la red eléctrica es de 50 Hz, y ésta pasa por cero dos veces, se tiene una frecuencia de disparo de 100 Hz o, lo que es lo mismo, de 10 ms de periodo.
- P la potencia continua disipada (si no fuese pulsada).
- $P_{R2,R3}$ la potencia disipada por el pulso en la resistencia equivalente a R_2 y R_3 (conectadas en serie).

Como el valor de R_2 es mucho mayor que el de R_3 , será en R_2 en la que se produzca la mayor disipación de potencia, siendo el valor de la potencia disipada en las resistencias calculado como sigue. Potencia disipada en R_3 debido a la corriente pulsada:

$$P_{R3,pulsada} = P_{R2,R3,pulsada} \cdot \frac{R_3}{R_2 + R_3}$$

Potencia disipada en R_2 (por ella sólo circula la correspondiente al pulso):

$$P_{R2,total} = P_{R2,R3,pulsada} - P_{R3,pulsada}$$

Potencia total disipada en R_3 (debido a la acción conjunta de la corriente pulsada y de la corriente del circuito amortiguador):

$$P_{R3,total} = P_{R3,pulsada} + P_{R3,circuito\ amortiguador}$$

3.4 Disipador térmico

Este apartado tiene como objetivo analizar la conveniencia de añadir un disipador térmico al TRIAC BTA16. Para su análisis, así como para la elección de la potencia máxima a disipar por el resto de componentes pasivos (resistencias), se han aplicado los siguientes criterios:

- Todo el hardware (circuito detector, actuadores y controlador) estará instalado en el interior de una caja eléctrica, denominada caja de control.
- El espacio dentro de la caja de control será reducido, y no existirá ningún tipo de refrigeración más allá de la pasiva, por lo que la temperatura en su interior será más elevada que la temperatura ambiente (sobre todo por la acción de los actuadores).
- La localización de la instalación es en Córdoba. La temperatura ambiente de trabajo puede llegar a ser muy elevada, sobre todo en los meses de verano.

Considerando estos criterios, se concluye que la temperatura ambiente máxima será de 60°C.

3.4.1.1 Datos previos

Se hace una recopilación de datos de la hoja de características (STMicroelectronics, 2010):

- Temperatura máxima de la unión (T_j): 125°C. Se añade un margen de seguridad del 10% a esta temperatura, por lo que se limita a 112,5°C en los cálculos.
- Resistencia térmica entre la unión y el aire circundante ($R_{th(j-a)}$): 60°C/W

Y de las condiciones de trabajo del componente:

- Temperatura ambiente: 60°C (véase criterios térmicos en la Introducción)
- Máxima potencia disipada en el componente (ver Ilustración 12): 2W (correspondiente al ventilador, que es el motor de más potencia de la torre,).

Figure 1. Maximum power dissipation versus on-state rms current (full cycle)

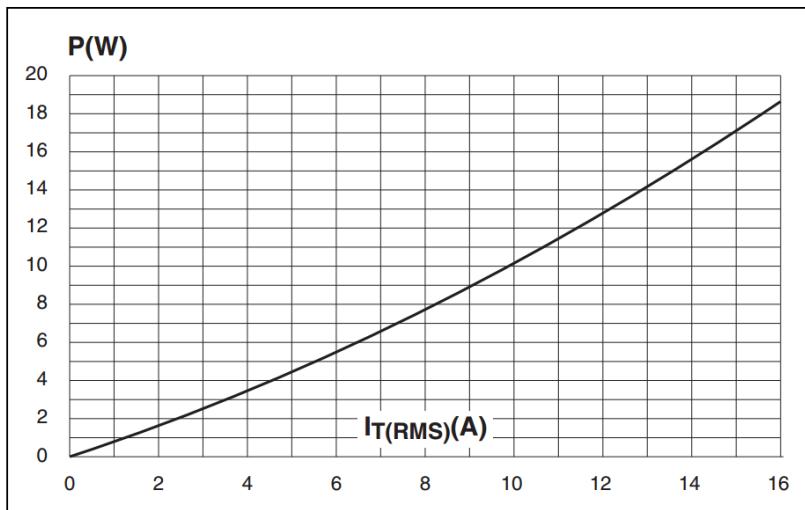


ILUSTRACIÓN 12: POTENCIA DISIPADA EN EL COMPONENTE BTA16 EN FUNCIÓN DE LA CORRIENTE (STMICROELECTRONICS, 2010)

3.4.1.2 Cálculos previos

Esta fórmula (Trujillo, Pozo, & Triviño, 2011) permite saber si es necesaria la utilización de un dissipador térmico en un componente con encapsulado del tipo TO220, como el BTA16 utilizado en el diseño. Para ello, estima la temperatura de la unión a través de los datos recopilados en el apartado anterior.

$$T_{j,estimada} = P \cdot R_{th(j-a)} + T_a$$

Donde P es la potencia disipada en el componente, $R_{th(j-a)}$ es la resistencia térmica entre la unión semiconductora y el aire circundante y T_a es la temperatura ambiente.

En caso de que la temperatura estimada de la unión sea superior a la temperatura máxima dada por el fabricante, será necesaria la utilización de un dissipador.

3.4.1.3 Cálculos del disipador

El montaje del disipador que se va a utilizar en el circuito tendrá la siguiente configuración:

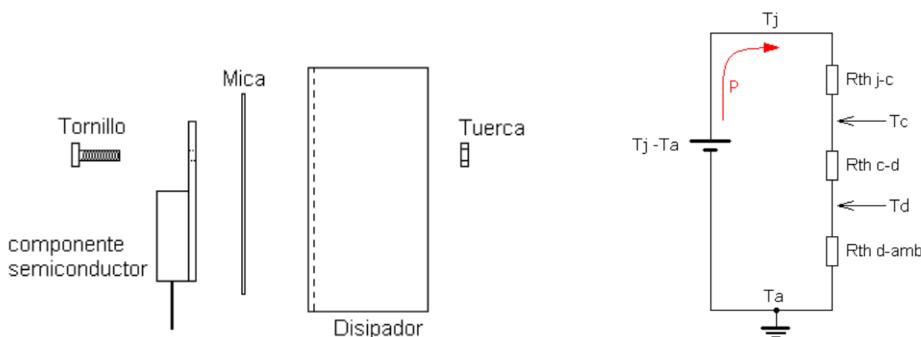


ILUSTRACIÓN 13: DIAGRAMA DEL MONTAJE COMPONENTE-DISIPADOR Y CIRCUITO TÉRMICO EQUIVALENTE (TRUJILLO, POZO, & TRIVIÑO, 2011)

En nuestro diseño, sin embargo, no es necesaria la utilización de la mica aislante, puesto que, como se comentó anteriormente en las características del TRIAC BTA16, su pestaña está aislada eléctricamente. Respecto al circuito térmico equivalente, se tiene:

- T_j : Temperatura de la unión semiconductor.
- T_a : Temperatura ambiente.
- T_c : Temperatura del encapsulado del TRIAC.
- T_d : Temperatura del disipador.
- $R_{th\ j-c}$: Resistencia térmica entre la unión y el encapsulado. Según la hoja de características (STMicroelectronics, 2010), es de $2,1^\circ\text{C}/\text{W}$.
- $R_{th\ c-d}$: Resistencia térmica entre el encapsulado y el disipador. Para una unión directa, y sin silicona termo-conductora, se estima de $1,5^\circ\text{C}/\text{W}$.
- $R_{th\ d-a}$: Resistencia térmica entre el disipador y el ambiente.
- P : Potencia disipada en el TRIAC.

Para calcular la máxima resistencia térmica permitida (del disipador al ambiente), se utiliza la siguiente fórmula (Trujillo, Pozo, & Triviño, 2011):

$$R_{th\ d-a} \leq \left(\frac{T_{j,máx} - T_a}{P} - (R_{th\ j-c} + R_{th\ c-d}) \right)$$

Una vez conocida ésta, se procede a la elección de un disipador que tenga una resistencia térmica igual o menor a la calculada.

3.4.1.4 Elección del disipador

Debido a que se disponía de varios circuitos (Variador de potencia comercial, 2017) mencionados en los antecedentes, los cuales incluyen disipador, se reutilizaron éstos para incorporarlos al diseño. El disipador en cuestión es el siguiente:



ILUSTRACIÓN 14: FRONTAL, LATERAL Y VISTA SUPERIOR DEL DISIPADOR DEL CIRCUITO COMERCIAL (VARIADOR DE POTENCIA COMERCIAL, 2017)

Como no se disponía del valor exacto de la resistencia térmica del disipador, se consultaron varios catálogos para determinar, en función de la forma y dimensiones, un valor estimado, llegándose a la conclusión de que la resistencia térmica era de 16,7°C/W (Aavid Thermalloy). Por tanto, este disipador puede emplearse con el circuito diseñado.

Una vez realizados los cálculos, en el caso de que se disponga de margen entre la resistencia térmica del disipador utilizado y la resistencia térmica máxima permitida para el circuito diseñado, significa que el circuito se puede emplear para controlar potencias mayores a la potencia de diseño. En ese caso, se calcula la potencia máxima soportada.

4 Cálculos

Los valores escogidos de las hojas de características son los que ofrecen una mayor restricción para el diseño (peor caso posible) para garantizar la operatividad, durabilidad y seguridad del circuito diseñado.

Todos los cálculos aquí realizados, y las fórmulas empleadas, se explican en el correspondiente apartado justificativo (2. Detector de paso por cero y 3. Actuador sobre motor).

4.1 Etapa de entrada al microcontrolador

4.1.1 R1

Para R_1 de 56k Ω , la corriente de entrada (eficaz y máxima) es:

$$I_{rms, \text{entrada } H11AA1} = \frac{230 \text{ V}}{56 \text{ k}\Omega} = 4,12 \text{ mA}$$

$$I_{máx., \text{entrada } H11AA1} = \frac{325 \text{ V}}{56 \text{ k}\Omega} = 5,8 \text{ mA}$$

Y la potencia disipada tanto en la etapa de entrada del integrado como en la resistencia es:

$$P_{H11AA1,entrada} = 4,12 \text{ mA} \cdot 1,5 \text{ V} = 6,18 \text{ mW}$$

$$P_{R1} = (4,12 \text{ mA})^2 \cdot 56 \text{ k}\Omega = 0,95 \text{ W}$$

4.1.2 R2

Para R₂ de 10kΩ, la intensidad eficaz que circula por ella es:

$$I_{rms,salida H11AA1} = \frac{(5 - 0,4) \text{ V}}{10 \text{ k}\Omega} = 0,46 \text{ mA}$$

Y la potencia disipada tanto en la etapa de salida del integrado como en la resistencia es:

$$P_{H11AA1,salida} = 0,46 \text{ mA} \cdot 0,4 \text{ V} = 0,184 \text{ mW}$$

$$P_{R2} = (0,46 \text{ mA})^2 \cdot 10 \text{ k}\Omega = 2,12 \text{ mW}$$

4.2 Etapa de salida del microcontrolador

4.2.1 R1

Para R₁ de 220Ω, la intensidad que circula por ella es:

$$I_{R1} = \frac{(5 - 1,5) \text{ V}}{220 \text{ }\Omega} = 15,91 \text{ mA}$$

Valor muy cercano al valor mínimo de 15mA establecido como requisito. Teniendo en cuenta la tolerancia ±5% de la resistencia, el peor caso (mínima intensidad) sería:

$$I_{R1} = \frac{(5 - 1,5) \text{ V}}{(220 + 5\%220) \text{ }\Omega} = \frac{3,5 \text{ V}}{231 \text{ }\Omega} = 15,15 \text{ mA}$$

La potencia a disipar en la resistencia vendría dada por:

$$P_{R1} = (15,91 \text{ mA})^2 \cdot 220\Omega = 55,7 \text{ mW}$$

Y para esta corriente de entrada, se tendría una disipación en el LED infrarrojo de:

$$P_{MOC3021,entrada} = 15,91 \text{ mA} \cdot 1,5 \text{ V} = 23,87 \text{ mW}$$

4.2.2 R2

Para R₂ el **valor mínimo** admisible es:

$$R_{2,valor \text{ } mínimo} = \frac{V_{pico}}{I_{máx}} = \frac{230\sqrt{2} \text{ V}}{1,2 \text{ A}} = 271,1 \text{ }\Omega$$

Para el cálculo del **valor máximo**, es necesario conocer la tensión de red:

$$V(t) = V_{pico} \cdot \sin(2\pi f \cdot t) = 230\sqrt{2} \sin(100\pi \cdot t) V$$

Y establecer la mínima tensión a partir de la cual se pueda disparar el TRIAC. Para una tensión de 4V, el tiempo de disparo mínimo a partir del cual se puede disparar es:

$$x = V_{pico} \cdot \sin(2\pi f \cdot t)$$

$$4 V = 230\sqrt{2} \sin(100\pi \cdot t) V$$

$$t_{disparo,mínimo} = \frac{\sin^{-1}\left(\frac{x}{V_{pico}}\right)}{2\pi f} = \frac{\sin^{-1}\left(\frac{4 V}{230\sqrt{2} V}\right)}{100\pi \text{ Hz}} = 39,15 \mu\text{s}$$

Por otro lado, para que se pueda disparar a una tensión de red de 4V, el valor de R_2 máximo ha de ser:

$$V_{red} = V_{R2} + V_{GT} = I_{G,mínima} \cdot R_2 + V_{GT} = 2,5 \text{ mA} \cdot R_2 + 1,3 V$$

$$R_{2,valor\ máx} = \frac{x - V_{GT}}{I_{G,mínima}} = \frac{4 V - 1,3 V}{2,5 \text{ mA}} = 1,08 \text{ k}\Omega$$

Para R_2 de 1kΩ:

$$I_{máx.pulsada,MOC3021,salida} = \frac{V_{máx}}{R_2} = \frac{V_{red} \cdot \sqrt{2}}{R_2} = \frac{230\sqrt{2}}{1 \text{ k}\Omega} = 325,27 \text{ mA}$$

4.2.3 Circuito amortiguador (R3 y C1)

Señal de la red eléctrica:

$$V(t) = V_{pico} \cdot \sin(2\pi f \cdot t) = 230\sqrt{2} \sin(100\pi \cdot t) V$$

Tasa de cambio máxima de la tensión de red:

$$\frac{dV(t)}{dt} = V_{pico} \cdot 2\pi f \cdot \cos(2\pi f \cdot t) = 230\sqrt{2} \cdot 100\pi \cdot \cos(100\pi \cdot t) V/\text{s}$$

$$\left. \frac{dV(t)}{dt} \right|_{máx} = 230\sqrt{2} \cdot 100\pi \cdot \cos(100\pi \cdot t) \Big|_{t=0} V/\text{s} = 230\sqrt{2} \cdot 100\pi \frac{V}{s} = 102186,3 V/\text{s}$$

Variación de la tensión con la inclusión del circuito amortiguador:

$$\frac{dV(t)}{dt} = \frac{V_{pico}}{\tau} = \frac{V_{pico}}{R_3 C_1}$$

$$\tau_{deseada} = R_3 C_1 = \frac{V_{pico}}{(dV/dt)_{deseado}} = \frac{230\sqrt{2} V}{50 V/\mu s} = 6,51 \mu s$$

Fijando C_1 a $0,1 \mu F$:

$$R_3 = \frac{\tau_{deseada}}{C_1} = \frac{6,51 \mu s}{0,1 \mu F} = 65,1 \Omega$$

Para un valor estándar de resistencia de 100Ω , se tiene que la constante de tiempo del conjunto RC es:

$$R_3 C_1 = 0,1 \mu F \cdot 100 \Omega = 10 \mu s$$

Valor que es superior a los $6,51 \mu s$ calculados, por lo que el circuito amortiguador cumple su función. La impedancia que presenta es:

$$Z_{R3,C1} = R_3 + \frac{1}{wC_1} = 100 \Omega + \frac{1}{2\pi 50 Hz \cdot 0,1 \mu F} = 31,93 k\Omega$$

La corriente que circula por él:

$$I_{rms,circuito\ amortiguador} = \frac{V_{rms}}{Z_{R3,C1}} = \frac{230 V}{31,9 k\Omega} = 7,21 mA$$

Y la potencia disipada en la resistencia por efecto de la corriente que circula a través del circuito amortiguador:

$$P_{R3,circuito\ amortiguador} = I_{rms,circuito\ amortiguador}^2 \cdot R_3 = (7,21 mA)^2 \cdot 100\Omega = 5,2 mW$$

4.2.4 Potencia a disipar por R2 y R3

Intensidad que circula por ellas teniendo en cuenta su conexión en serie:

$$I_{máx,MOC3021} = \frac{V_{pico}}{R_2 + R_3} = \frac{230\sqrt{2}}{1,1 k\Omega} = 295,7 mA$$

Potencia a disipar si la corriente fuese continua:

$$P_{R2,R3,continua} = V_{máx} \cdot I_{máx,MOC3021} = 230\sqrt{2} \cdot 296 mA = 96,3 W$$

Potencia a disipar teniendo en cuenta el carácter pulsado de la corriente:

$$P_{R2,R3,pulsada} = \frac{P \cdot t_{pulso}}{T} = \frac{96,3 W \cdot 100 \mu s}{10 ms} = 0,96 W$$

Potencia disipada en R_3 debido al pulso de corriente:

$$P_{R3,pulsada} = P_{R2,R3,pulsada} \cdot \frac{R_3}{R_2 + R_3} = 0,96 \text{ W} \cdot \frac{100 \Omega}{1 \text{ k}\Omega + 100 \Omega} = 88 \text{ mW}$$

Potencia total disipada en R_2 :

$$P_{R2,total} = P_{R2,R3,pulsada} - P_{R3,pulsada} = (960 - 88) \text{ mW} = 875 \text{ mW}$$

Potencia total disipada en R_3 :

$$P_{R3,total} = P_{R3,pulsada} + P_{R3,circuito amortiguador} = (88 + 5,2) \text{ mW} = 93,2 \text{ mW}$$

4.3 Disipador térmico

4.3.1 Cálculos previos

Temperatura estimada de la unión semiconductor:

$$T_{j,estimada} = P \cdot R_{th(j-a)} + T_a = 2\text{W} \cdot 60^\circ\text{C}/\text{W} + 60^\circ\text{C} = 180^\circ\text{C}$$

4.3.2 Cálculos del disipador

Máxima resistencia térmica permitida (del disipador al ambiente):

$$\begin{aligned} R_{th\ d-a} &\leq \left(\frac{T_{j,máx} - T_a}{P} - (R_{th\ j-c} + R_{th\ c-d}) \right) \rightarrow \\ R_{th\ d-a} &\leq \left(\frac{112,5^\circ\text{C} - 60^\circ\text{C}}{2\text{W}} - \left(\frac{2,1^\circ\text{C}}{\text{W}} + \frac{1,5^\circ\text{C}}{\text{W}} \right) \right) \rightarrow R_{th\ d-a} \leq 22,65^\circ\text{C}/\text{W} \end{aligned}$$

4.3.3 Elección del disipador

Se procede a calcular la potencia máxima de disipación del componente para la cual sería seguro emplear las placas de control de los motores (con los márgenes de seguridad térmicos especificados en el apartado 3.4), puesto que se tiene un margen entre la resistencia térmica del disipador empleado y de la máxima resistencia térmica admisible para la aplicación. Empleando un disipador con una resistencia térmica de $16,7^\circ\text{C}/\text{W}$, se tiene que:

$$R_{th\ d-a} \leq \left(\frac{112,5^\circ\text{C} - 60^\circ\text{C}}{P} - \left(\frac{2,1^\circ\text{C}}{\text{W}} + \frac{1,5^\circ\text{C}}{\text{W}} \right) \right) \rightarrow$$

$$\left(\frac{112,5^\circ\text{C} - 60^\circ\text{C}}{P} - \left(\frac{2,1^\circ\text{C}}{\text{W}} + \frac{1,5^\circ\text{C}}{\text{W}} \right) \right) \leq 16,7^\circ\text{C}/\text{W}$$

$$P \leq \frac{112,5^\circ\text{C} - 60^\circ\text{C}}{20,3^\circ\text{C}/\text{W}} = 2,6\text{W}$$

5 Resultados

5.1 Circuito de entrada

5.1.1 Valores

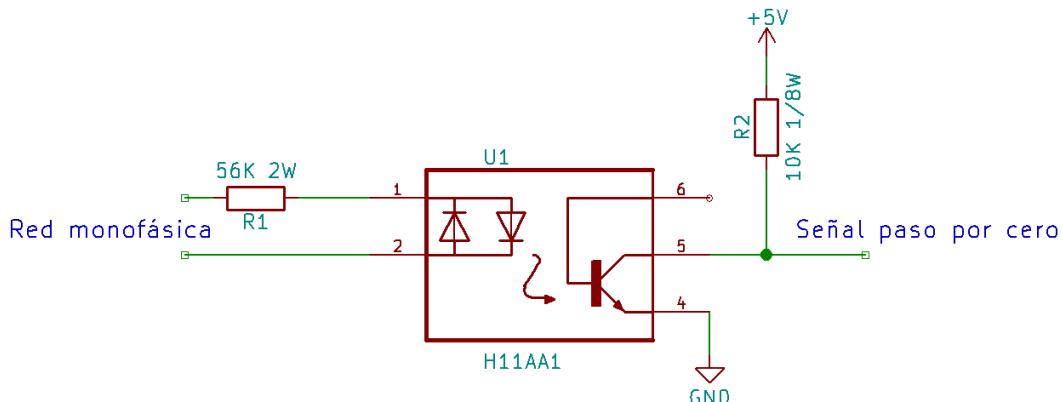
5.1.1.1 Resistencia de entrada (R_1)

Se elige una resistencia estándar de **56kΩ y 2W**, ya que la elección de una resistencia capaz de soportar una disipación máxima de 1W estaría muy cercana a la potencia de disipación calculada y no cumpliría con el margen de seguridad basado en los criterios establecidos previamente (véase el apartado de Introducción de este documento).

5.1.1.2 Resistencia de salida (R_2)

Se elige un valor estándar para R_2 de **10kΩ y 1/8W** (mínimo valor posible) pues la potencia disipada en ella es muy pequeña.

5.1.2 Esquemático



ESQUEMA 6: EQUÍMÁTICO DEL CIRCUITO DETECTOR DE PASO POR CERO

5.1.3 Experimentación

La Ilustración 15 muestra la medición del pulso producido en la detección del paso por cero.

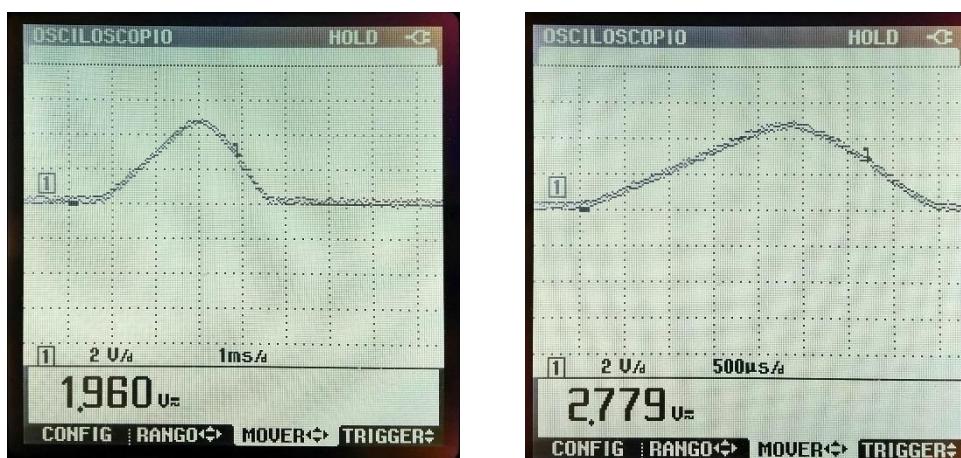


ILUSTRACIÓN 15: PULSO DURANTE EL PASO POR CERO (IZQUIERDA) Y AMPLIACIÓN (DERECHA).

Como se previó en el diseño (véase apartado 2.2.2), el flanco de bajada tiene una duración menor al flanco de subida. En el software, la interrupción externa se configura para la detección del flanco de bajada, para evitar problemas de disparo antes del paso por cero (pues el flanco de subida se produce instantes antes del paso, y el flanco de bajada tras el paso).

Para el análisis del tiempo de retraso entre el paso por cero y la detección por el flanco de bajada, es necesario conocer el nivel de tensión que el microcontrolador considera como “nivel bajo” (y por lo tanto, tras alcanzarlo, considere que ha habido flanco de bajada y se active la interrupción). La Ilustración 16 (Sparkfun, 2017) representa, de forma muy visual, los datos ofrecidos por la hoja de características del microcontrolador ATmega328 (Atmel, 2016), donde se indica que el valor máximo para el nivel lógico bajo es de 0,3 veces la tensión de alimentación. Como se está alimentando la placa Arduino con 5V, el valor V_{IL} es, exactamente, de 1,5V.

De la medición ampliada de la Ilustración 15 se puede obtener diferentes magnitudes (tanto de tiempo como de tensión):

- El pico, de casi 5V, se produce en el instante de paso por cero de la sinusoide, en torno a 2,5 milisegundos tras comenzar el pulso.
- El flanco de bajada provoca la interrupción externa 1 milisegundo después de haberse producido el paso por cero.

Este análisis refleja que no se va a poder disparar en el milisegundo posterior al paso por cero. Es no es un problema, puesto que la potencia que se entrega, si se dispara un milisegundo después del paso por cero, con respecto a la conducción continua, es prácticamente igual. Así, cuando se lleguen a potencias en torno a 85-90%, es preferible mantener el pin de control permanentemente activado, entregándose la máxima potencia, pues mejora los armónicos de red y la salida medida va a tener la misma magnitud en ambos casos. Además, en los valores cercanos a los extremos, la tensión eficaz que se entrega a la carga no varía de forma lineal, como se analiza en el apartado 7.5.2).

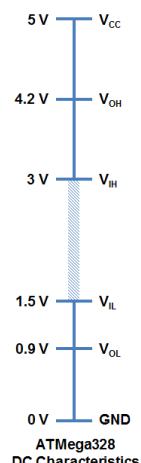


ILUSTRACIÓN 16:
NIVELES LÓGICOS EN
ARDUINO

5.2 Circuito de salida

5.2.1 Valores

5.2.1.1 Resistencia de entrada (R_1)

Un valor de resistencia de 220Ω y tolerancia $\pm 5\%$ cumple con el requisito mínimo de corriente. Como la potencia máxima soportada es de 100mW en conducción continua (en el circuito diseñado ésta es pulsada³, por lo que el límite es superior, cumpliéndose aún más holgadamente), con la elección de una resistencia $220\Omega \pm 5\% \text{ y } \frac{1}{4} \text{ W}$ se superan todos los requisitos.

³ La señal de control será pulsada la mayor parte del tiempo, pues se activará en el momento necesario y se desactivará unas decenas de microsegundos después. Sólo será continua cuando se pretenda que el motor esté al 100%, caso en el que permanecerá a 0V, o el motor al 0% [apagado], estando en este caso la señal de control a 5V constantemente.

5.2.1.2 Resistencia de salida (R_2)

Si se fija que se pueda disparar a partir de una tensión de red de 4V, se tiene que el menor tiempo de disparo que el usuario es capaz de disparar es de $39,15\mu s$, valor que representa el 0,4% del total de 10ms que dura la mitad de la senoide de la red (y del que se dispone para realizar el disparo). Se considera este valor bueno pues el diseñador entiende que pretender regular un motor al 99,6% de su potencia total no tiene sentido, pues la variación en su respuesta será nula.

Así pues, con una resistencia de valor de $1k\Omega$ se asegura que puede dispararse la conducción cuando la tensión de red es aún inferior a 4V.

Por último, el valor resultante de intensidad pulsada obtenido (325mA) es muy inferior a la máxima de 1,2A permitida, considerándose como válido el valor de $1k\Omega$ establecido. Para calcular la potencia a disipar se necesita conocer el valor de intensidad que circula por ella debido al circuito amortiguador, por lo que se calcula más adelante.

5.2.1.3 Circuito amortiguador (R_3 y C_1)

El ratio de cambio de la tensión de red debido a su frecuencia de 50Hz es de tan solo $0,1 V/\mu s$, por lo que por éste no habrá ningún problema. La restricción proviene del corte de la conducción cuando la senoide esté en su punto de máxima tensión.

Se añade un circuito amortiguador formado por R_3 y C_1 que permite amortiguar la tasa de cambio que se produciría en caso de cesar la conducción cuando la tensión de red es máxima. La constante de tiempo de éste es de $10\mu s$, superior a la mínima necesaria para que el foto-triac del integrado MOC3021 no se auto-dispare.

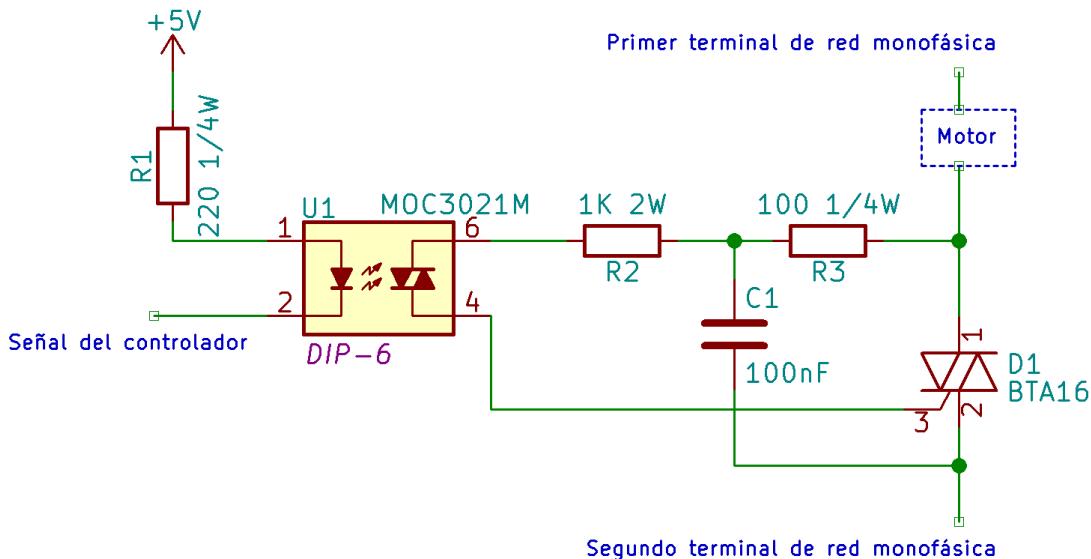
El valor del condensador C_1 es de $0,1\mu F$ y la tensión máxima que soporta es de **400V**.

Tras el cálculo de la potencia a disipar por R_2 y R_3 , se escogen los valores finales siguientes:

$$R_2: 1k\Omega, 2W \quad R_3: 100\Omega, \frac{1}{4}W$$

Como se aprecia, para R_2 se ha aplicado un margen de seguridad, eligiéndose el valor inmediatamente superior al de 1W. Debido a que el valor a disipar de 0,9W está muy cercano a 1W, los criterios térmicos (explicados en la parte justificativa del Disipador térmico) no iban a ser satisfechos de manera adecuada, por ello se ha procedido al sobredimensionamiento de la potencia capaz de disipar.

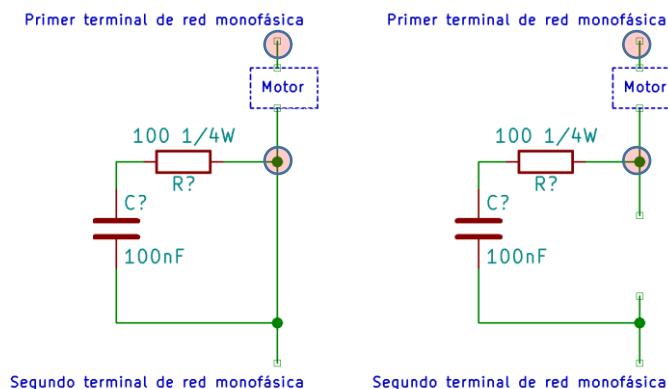
5.2.2 Esquemático



ESQUEMA 7: ESQUEMÁTICO PARA ACTUAR SOBRE CARGAS CONECTADAS A UNA RED MONOFÁSICA MEDIANTE UN MICROCONTROLADOR

5.2.3 Experimentación

Los puntos de medición son los marcados como “bornes de carga monofásica”, en el Esquema 5, correspondientes a los círculos rojos del Esquema 8. Por tanto, cuando el motor esté en conducción, se muestra la tensión en sus bornes y, cuando no lo esté, se muestra una tensión parásita producida por el circuito amortiguador.



ESQUEMA 8: CIRCUITO EQUIVALENTE PARA TRIAC EN CONDUCCIÓN (IZQUIERDA) Y TRIAC EN NO CONDUCCIÓN (DERECHA).

En los transitorios de conducción a no conducción del triac, en la Ilustración 17, es cuando la corriente comienza a circular a través del circuito amortiguador, observándose una respuesta subamortiguada ante esta excitación. Ésta se debe al circuito RLC formado por la inductancia y la resistencia del motor, y el condensador y la resistencia del circuito amortiguador. La condición para que se dé esta respuesta subamortiguada es:

$$R^2 < 4L/C$$

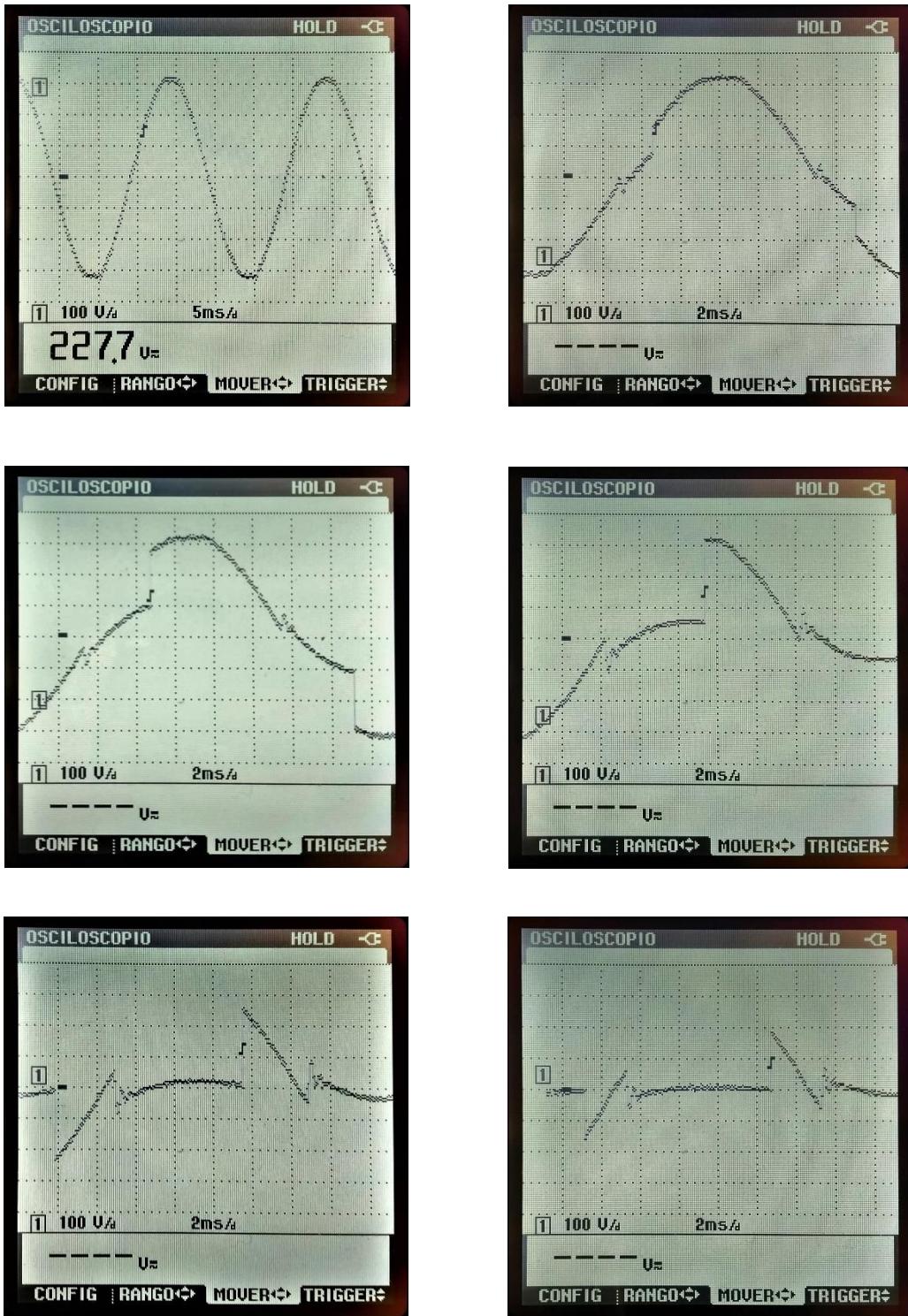


ILUSTRACIÓN 17: TENSIÓN EN BORNES DE LA CARGA, EN LA PLACA DE CONTROL, PARA EL VENTILADOR. SE MUESTRA DESDE RETRASO EN DISPARO DE 0 – CONDUCCIÓN CONTINUA- (ESQUINA IZQUIERDA SUPERIOR) HASTA 8 MILISEGUNDOS (ESQUINA DERECHA INFERIOR), PASANDO POR: 1,5, 3, 5 Y 7 MILISEGUNDOS.

5.3 Disipador térmico

Como los cálculos previos predicen una temperatura de la unión de 180°C, superior a la de 112,5°C dada por el fabricante (aplicando un margen del 10%, para añadir mayor seguridad al diseño), es necesaria la incorporación de un disipador térmico.

Como la resistencia térmica de los disipadores de los que se dispone es de 16,7°C/W, y la máxima resistencia térmica permitida calculada es de 22,65°C/W, se pueden usar estos disipadores en el circuito diseñado. Además, también permiten utilizar el circuito con motores de potencias superiores al máximo empleado en el circuito, debido al margen entre la resistencia térmica del disipador usado y la máxima calculada de 22,65°C/W. Esta potencia máxima capaz de disipar el componente se ha calculado (apartado 4.3.3), resultando de 2,6W, potencia que corresponde con una corriente de 3,5A (Ilustración 12), equivalente a una **carga con una potencia de 800W**. Si el circuito se emplease fuera de la caja de control, entonces la temperatura ambiente no llegaría a 60°C, como se dispuso en el apartado 3.4, sino que sería inferior, siendo el circuito capaz de controlar cargas de aún más potencia.

6 Presupuesto

Este presupuesto es de carácter orientativo y no refleja los costes del equipo que se ha elaborado puesto que ya se poseían algunos de los componentes (placa Arduino, disipadores, resistencias, condensadores...).

Este presupuesto está pensado para la **elaboración de 10 placas y que 3 de ellas sean instaladas en la torre** (en un caja eléctrica impermeable, con conexiones impermeables), ya que los componentes suelen venderse en conjunto y no individuales. A pesar de que no todos los componentes son iguales a los utilizados en los cálculos, las hojas características de éstos han sido revisadas y ofrecen las mismas características (o superiores) a los utilizados.

Producto y unidades	Precio del conjunto	Precio equipo en torre
Disipadores térmicos (10 unidades) [1]	9,65€	2,90€
BTA16-800BWRG (10 unidades) [2]	10,90€	3,27€
MOC3021 (10 unidades) [3]	3,96€	1,18€
H11AA1 (pedido mínimo, 5 unidades) [4]	3,00€	0,6€
Placas matriz, diferentes tamaños [5]	13,99€	7€
Arduino Nano (3 unidades) [6]	10,99€	3,67€
Condensadores 100nF (10 unidades) [7]	7,36€	2,2€
Conjunto de resistencias [8]	2,99€	0,90€
Interruptor balancín iluminado (1 unidad) [9]	2,11€	2,11€
Caja eléctrica impermeable (1 unidad) [10]	7,99€	7,99€
5 conectores impermeables 3 pines [11]	12,13€	12,13€
Terminal PCB 3 vías (10 unidades) [12]	8,88€	3,55€
Terminal PCB 2 vías (20 unidades) [13]	13,48€	4,71€
Presupuesto	Total: 107,43€	Caja control montada en torre: 52,22€

Las referencias a cada uno de los productos de la tabla se encuentran en el anexo “Referencias de la tabla de presupuestos”.

7 Desarrollo del software de control

El software de control se ha desarrollado para la plataforma Arduino. En este apartado se realiza un repaso de la evolución temporal en el desarrollo del software y, a continuación, se describen detalladamente las características que se han implementado.

7.1 Evolución temporal del algoritmo de control

Se comenzó con una implementación inicial, muy rudimentaria y escrita como prueba de concepto, consistente en la detección del paso por cero y la temporización, mediante la función *delayMicroseconds()* de Arduino, del tiempo de retraso (cada 1% de retraso representaba 100 μ s de retraso temporal en la activación) y, tras la activación de la señal de control, otra temporización (de 100 μ s) para su desactivación.

Tras este primer acercamiento, se empieza la elaboración del control de los tres motores. La introducción de temporizaciones relativas complica mucho el diseño, principalmente la gestión de las temporizaciones para las desactivaciones de las señales de control. Debido a que en la plataforma Arduino toda depuración ha de realizarse a través del Puerto Serie, se comienza a elaborar la interfaz de usuario (mediante Puerto Serie), incorporando el menú de selección y las potencias de los diferentes motores en modo manual. La incorporación de cada nuevo elemento lleva asociado el desarrollo, en paralelo, del código de Puerto Serie para actuar como interfaz. En esta primera fase del algoritmo de control todas la temporizaciones se siguen realizando con la función *delayMicroseconds()*, pues permitía ofrecer una solución funcional y en un tiempo de desarrollo más pequeño.

Una vez depurado el funcionamiento de la placa con la ejecución del código de control para los tres actuadores, se incorporó la resistencia de calentamiento, y después los controles automáticos (PIDs). Por último, se desarrolló la protección de las bombas por los sensores de nivel y se programó la comunicación I2C.

Una vez avanzado el proyecto, con todas las funcionalidades implementadas y con una interfaz de usuario completa que requiere mayor carga computacional, se hace necesaria la búsqueda de una alternativa a las temporizaciones mediante *delay* (que desperdicia, durante el tiempo que se ejecuta la temporización, tiempo de programa, pues el microcontrolador no ejecuta ninguna otra línea de código). Se realiza la incorporación de las temporizaciones hardware, utilizando el temporizador 1 del microcontrolador (Atmel, 2016). Esto permite “parallelizar” la actuación sobre los motores con el resto de programa, quedando *Arduino de Control* muy liberada (computacionalmente hablando), lo que permite la incorporación de mucho más software, si fuese necesario, en un futuro.

Tras esto, el software de control estaba lo suficientemente maduro para realizar su instalación en la torre.

7.2 Comunicación I2C

Para la programación de la comunicación I2C el desarrollo se ha basado en la librería *Wire* de Arduino. Además, como se envían datos no nativos (de tipo *float*), también se han usado las funciones *I2CAnything* de Nick Gammon, para gestionar este tipo de envíos y recepciones.

Para la integración con el resto de la planta, se consensuó con el director del proyecto que la trama de datos consistiría en:

- 1 octeto de cabecera: Condiciona los *floats* de datos.
- 4 *floats* de datos: Contienen o información de velocidad, Reynolds... o bien una nueva consigna para un PID o un nuevo valor manual, en función de la información de la cabecera.

Tras cada recepción (se ejecuta la interrupción *repcion_I2C* cuando se ha recibido la totalidad de la trama), se almacenan los valores recibidos y se activa una bandera que indica que se tiene una nueva recepción. Esta es evaluada en el bucle *loop()* y, en caso de que se haya recibido algún dato, se llama a una función de evaluación de la comunicación recibida(*tratamiento_repcion_I2C*), que identifica, en función de la cabecera, si se trata de una trama de datos (incluye la velocidad del ventilador, Reynolds de agua y acercamiento) o de control (nuevo setpoint o nueva potencia manual para alguno de los motores).

No se ha establecido ningún mecanismo de comprobación de errores en la comunicación, simplemente una comprobación de que efectivamente la longitud de la trama recibida es equivalente a la trama establecida (1 byte + 4 *floats*).

También se incorpora una función (*watchdog_comunicacion_I2C*) que monitoriza la comunicación I2C. Ésta, dependiendo de su parámetro de entrada, resetea el *watchdog* o realiza la comprobación de si el tiempo que ha pasado desde la última recepción/petición I2C es superior al establecido como límite.

7.3 Puerto Serie

Debido a la forma mediante la cual Arduino gestiona el envío de caracteres de texto por el Puerto Serie, la interfaz de usuario, así como diversos mensajes destinados a la depuración, consume una gran cantidad de memoria dinámica (RAM) de Arduino. En este apartado se describe cómo se identificó el problema y la solución dada.

7.3.1 Identificación del problema

El compilador de Arduino ofrece información acerca de la memoria usada por el programa en el microcontrolador. Ésta la ofrece, en forma de retroalimentación, cada vez que es ejecutada una compilación (junto a los mensajes de errores y de alerta, si es que éstos existen), en la parte inferior de la *IDE* de Arduino. Durante el desarrollo del software de control, se estaba llegando al límite de la memoria RAM: 97% ocupado. Además, el compilador ofrecía una advertencia de que podrían ocurrir problemas de estabilidad, y así fue durante las pruebas: ocurrían errores aleatorios en partes del código que ya habían sido depuradas.

El problema es identificado debido al aumento de la memoria dinámica entre compilaciones, cuando ni tan siquiera se añadían nuevas variables al programa, pero sí se añadían mensajes para depuración a través del Puerto Serie.

La forma de gestionar, por parte de Arduino, los mensajes de texto es bastante pobre, teniendo en cuenta que el Puerto Serie es su medio de depuración. El problema viene dado por la forma de almacenar los mensajes de texto: Guarda cada carácter como un byte en la memoria dinámica. Por tanto, incluso con el envío de un mensaje muy simple, ya se está incrementando, en gran medida, el uso de la memoria RAM del microcontrolador. Además, este espacio ocupado por un envío, no es liberado una vez que el mensaje se ha enviado, permaneciendo la memoria ocupada. En ninguna de la documentación consultada sobre Arduino se ha encontrado mención a esto, cuando es un aspecto muy interesante y a tener en cuenta de cara al desarrollador.

En lo que respecta al Puerto Serie se llevan a cabo dos acciones, una para paliar este uso excesivo de memoria RAM y otra de cara al programa final que estará instalado en la torre (que no necesita comunicación por Puerto Serie, sólo I2C).

7.3.2 Almacenamiento de mensajes en ROM

Mediante el almacenamiento de mensajes en la memoria ROM del microcontrolador, la cual es de un tamaño mucho mayor, de 32kB, frente a los 2kB de la dinámica (Atmel, 2016), se consigue evitar que éstos ocupen memoria RAM. El inconveniente es que, para su envío a través del Puerto Serie, es necesario la extracción del mensaje de la memoria ROM y su almacenamiento en el búfer de salida del Puerto Serie. A tal efecto se desarrolla una función (*mostrar_mensaje_almacenado_en_ROM*) que extrae un mensaje almacenado previamente en la memoria de programa y lo almacena en el búfer del Puerto Serie.

Una vez desarrollada y depurada esta herramienta, se almacenaron los mensajes de mayor tamaño en la memoria ROM (como el menú de comandos disponibles y el ejemplo mostrado al iniciarse el puerto serie), lo cual liberó una gran cantidad de memoria dinámica.

A pesar de disponer de este nuevo procedimiento, no todos los mensajes han sido pasados a la memoria de programa, sólo los de mayor tamaño, ya que el tiempo de desarrollo se incrementa (pues se hace necesario declarar el mensaje en la memoria ROM y, además, la mayoría de mensajes ya estaban implementados por el otro método). Así, se ha conseguido un equilibrio entre tiempo de desarrollo y estabilidad de Arduino, pues el espacio de memoria dinámica que ocupa el programa es de en torno al 64%, y la memoria de programa ocupada es del 54%.

7.3.3 Compilación condicional

Se realiza la compilación condicional de todo el código que tiene relación con el Puerto Serie. Así se logra una disminución enorme de la memoria dinámica usada. Por otro lado, la carga del microcontrolador se ve aligerada, ya que no tiene que ejecutar ninguna instrucción relativa al Puerto Serie. Esto es útil de cara a que la caja de control esté instalada, de forma definitiva, en la torre, pues su comunicación con el exterior se hará mediante I2C (actuando *Arduino Sensores* como pasarela).

Para elegir si las funciones relacionadas con el Puerto Serie son compiladas o no, dentro del fichero *configuracion_software.h*, ha de modificarse la siguiente línea de código:

```
#define MODO_DEPURACION 1
```

Estando comentada, se ha diseñado el código para que el compilador no tenga en cuenta el código relacionado con el Puerto Serie. Si está descomentada, entonces MODO_DEPURACION toma valor 1 y sí se produce la compilación de todo el código.

7.4 Cálculo del tiempo de disparo

Para modificar la potencia de un motor, hay que efectuar una llamada a la función (*calculo_valores_actuacion*). Admite dos parámetros de entrada, el primero que indica el identificador del motor ('F' (Fan) para el ventilador, 'W' (Water) para la bomba de agua, y 'S' (Spray) para la bomba de rociado), y el segundo indica la nueva potencia deseada.

El algoritmo, una vez identificado el motor, calcula el nuevo tiempo de disparo (a partir de la potencia proporcionada) y lo guarda en la variable global correspondiente al motor modificado. Para el cálculo se tiene en cuenta el aspecto analizado en el apartado 5.1.3 y se aplica una corrección en el tiempo calculado.

Tras esto, se hace una ordenación de tiempos de disparo. Existe un vector que contiene los pines a los que corresponde cada motor, y al hacer la ordenación de los tiempos de disparo, se ordena, paralelamente, este vector de pines, conservándose la correspondencia entre tiempo de disparo y motor a activar.

Hasta este punto se tienen dos vectores principales: el que contiene los pines (*pines_ordenados*) y el de tiempos (*v_times*), ambos ordenados. Ahora se introducen otro par de vectores (*v_estado_pines_durante_activacion* y *v_estado_pines_durante_desactivacion*) que son los que contienen el valor de la señal de control durante la activación y la desactivación. Así, si para el elemento X del vector ambos son 0, significa que *pines_ordenados[X]* se activará permanentemente. Si ambos son 1, se desactivará permanentemente, y si el de activación es 0 y el de desactivación 1, su potencia está siendo controlada. Estos vectores se han introducido para integrar la activación y desactivación permanente con la temporización del tiempo de disparo. Los criterios que se siguen son:

- Activación permanente del motor: Se requiere para aquellos motores cuyo tiempo de disparo absoluto está comprendido entre 0 y *TIEMPO_CORRECCION_PASO POR CERO*. En este margen, además de no ser interesante porque no se obtiene variación de la salida medida (al ser la potencia entregada muy parecida), no puede temporizarse debido al desfase de la detección de paso por cero.
- Desactivación permanente: Se requiere para aquellos motores cuyo tiempo de disparo absoluto es mayor a los 8500 microsegundos (85% de retraso de disparo). La potencia entregada en este punto es tan pequeña que el motor no consigue girar o el efecto sobre la variable medida es nulo (como si estuviese parado).

Tras ello, se realizan los cálculos para las temporizaciones relativas. Una vez detectado el paso por cero y temporizado el primer retardo (correspondiente al motor trabajando con mayor potencia), ha de realizarse la temporización del segundo retardo, pero debido a que ya se ha realizado una, la nueva temporización ha de realizarse conforme a la anterior. Además, para los motores desactivados no se realiza una temporización de 10 milisegundos, si no que directamente se leen los vectores introducidos en la activación/desactivación permanente y se dejan desactivados.

Finalmente, han de realizarse los cálculos para realizar las desactivaciones de las señales de control. Una vez activada, permanecerá en ese estado durante el tiempo marcado por *TIEMPO_MANTENER_ACTIVO_PINES*. Tras éste, ha de desactivarse. Si la temporización relativa para activar un segundo motor es inferior al tiempo que la señal de control ha de estar activa (es decir, inferior a *TIEMPO_MANTENER_ACTIVO_PINES*), entonces la señal de control del primer motor continúa activa hasta que haya que desactivarse la señal de control del segundo

motor, permaneciendo activa la primera señal unas decenas de microsegundos más de las que le corresponde. Esto no supone ningún problema (por la naturaleza de que la activación del segundo motor se ha producido muy cercana a la del primero). Este método se apoya en otro vector, *desactivacion_de_motor*, que indica si la siguiente temporización relativa es mayor al tiempo de activación de la señal de control.

En último lugar, una vez obtenidos los resultados de todas las operaciones que intervienen en el proceso de cálculo, almacenadas en vectores y variables que terminan con el sufijo *_tmp*, se vuelcan sobre otros vectores que estarán a la espera de que se produzca una interrupción externa (los que terminan con el sufijo *_esperando_interrupcion*). Este volcado debe realizarse con las interrupciones desactivas, para evitar que se produzca una interrupción de paso por cero cuando sólo la mitad de los valores han sido actualizados. Por otra parte, cuando se produzca una interrupción externa, se volcarán de nuevo los valores, que estaban a la espera de este evento, sobre las variables que definitivamente actuarán sobre la señal de control. El volcado ha de realizarse en este punto (al comienzo de nuevas temporizaciones) para que no afecte a las temporizaciones relativas que ya se estaban llevando a cabo en el ciclo anterior, lo que provocaría un mal control durante ese ciclo de onda de red.

7.5 Linealización de los motores

Para que los PIDs realicen su función de forma correcta, es necesario que la variable medida responda de forma lineal a la señal de control, es decir, que los cambios que se produzcan en ella para una variación dada de la señal de control sean los mismos, independientemente del punto de trabajo del actuador. Por ejemplo, si estando el actuador al 55% la medición es de 2.7, y al aumentar su potencia al 60% la medición es de 2.9, cuando el actuador esté al 75%, si se aumenta su potencia en un 5%, debe de experimentarse un aumento de 0.2 en la medición.

7.5.1 Linealización de la potencia

El tiempo de disparo se calcula de forma inversa al porcentaje introducido. Así, para un 100% de potencia, no se aplica ningún retraso en el disparo, y conforme disminuye la potencia el tiempo de disparo va aumentando, hasta que la potencia sea del 0% y el tiempo de disparo máximo, de 10 milisegundos. Matemáticamente, este concepto se expresa como sigue:

$$t_{disparo} = 100 \cdot (100 - \%_{potencia\ deseada}) \mu s$$

Y su representación:

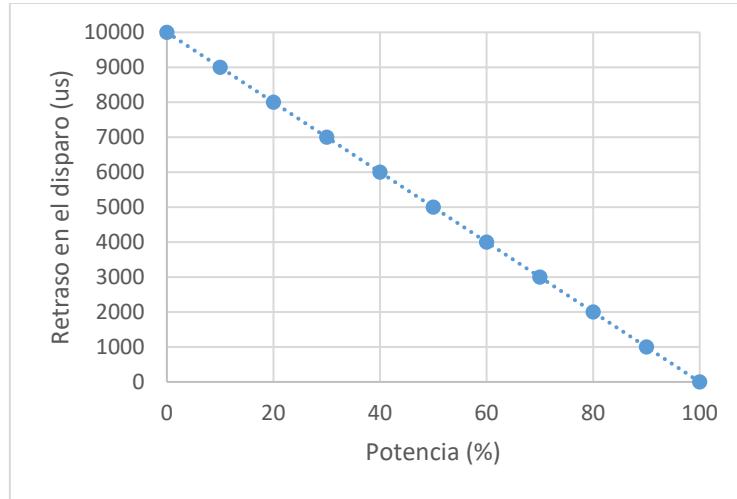


ILUSTRACIÓN 18: RETRASO EN EL DISPARO EN FUNCIÓN DE LA POTENCIA DESEADA.

Esta ecuación permite tener un control muy bueno del tiempo de disparo, por su relación totalmente lineal, y es la implementada para realizar la experimentación. Mediante ella, se recogen los valores necesarios para realizar la linealización de los motores.

7.5.2 Linealización de la sinusoide de la red (50Hz)

La tensión eficaz que recibe la carga, y por tanto la potencia entregada, tiene que ver con el área encerrada bajo la sinusoide. Los cálculos matemáticos de área, por mayor simplicidad, van a realizarse para una sinusoide genérica, en lugar de para la sinusoide de la red eléctrica, pues no interesa conocer el valor del área, sino la variación de ésta. El área encerrada bajo un semiperíodo de una sinusoide genérica es:

$$\int_0^{\pi} \sin(x) = -\cos(\pi) + \cos(0) = 2$$

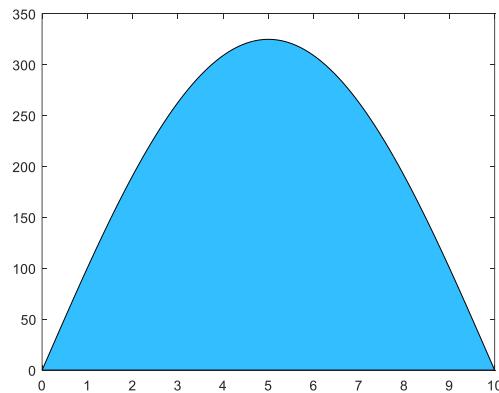


ILUSTRACIÓN 19: ÁREA ENCERRADA BAJO UN SEMIPERÍODO DE UN CICLO DE LA SINUSOIDE DE LA RED ELÉCTRICA.

Si se dispara transcurrido un tiempo desde el paso por cero, se está modificando el área encerrada bajo la sinusoide (véase Ilustración 3). Siendo $x_{disparo}$ el ángulo de disparo, entre 0 y π , el área que se tiene en función de $x_{disparo}$ es:

$$\int_{x_{disparo}}^{\pi} \sin(x) = [-\cos(x)]_{x_{disparo}}^{\pi} = -\cos(\pi) + \cos(x_{disparo}) = 1 + \cos(x_{disparo})$$

$$Area_{requerida} = 1 + \cos(x_{disparo})$$

Y el ángulo de disparo en función del área requerida:

$$x_{disparo} = \cos^{-1}(Area_{requerida} - 1) \text{ rad}$$

Y el paso del ángulo de disparo a tiempo de retraso de disparo, en milisegundos, para la sinusoide de red, viene dado por:

$$t_{disparo} = x_{disparo} \cdot \frac{10}{\pi} \text{ ms}$$

Siendo la fórmula resultante:

$$t_{disparo} = \frac{10}{\pi} \cdot \cos^{-1}(Area_{requerida} - 1) \text{ ms}$$

Esta área requerida varía de 0 a 2. A continuación se representa (Ilustración 20) el tiempo de disparo en función del área requerida y, para una mejor visualización, se ha representado ésta variación de área en porcentaje, de 0 a 100%. Como era de esperar, debido a que la sinusoide sufre su máxima variación (pendiente máxima) en los pasos por cero, cuando está cercana a estos puntos la tasa de variación del área es más pronunciada, comportándose de forma diferente al área central.

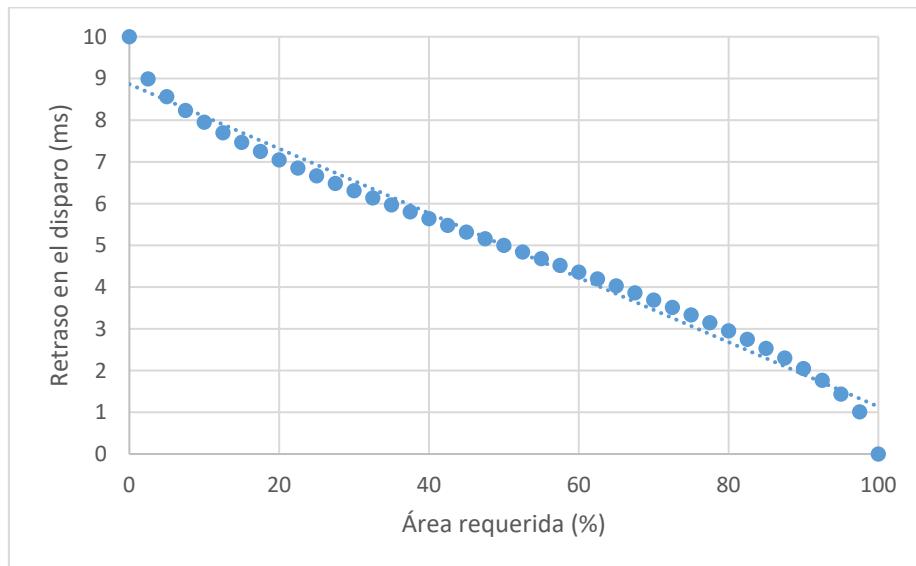


ILUSTRACIÓN 20: TIEMPO DE RETRASO EN EL DISPARO EN FUNCIÓN DEL ÁREA REQUERIDA

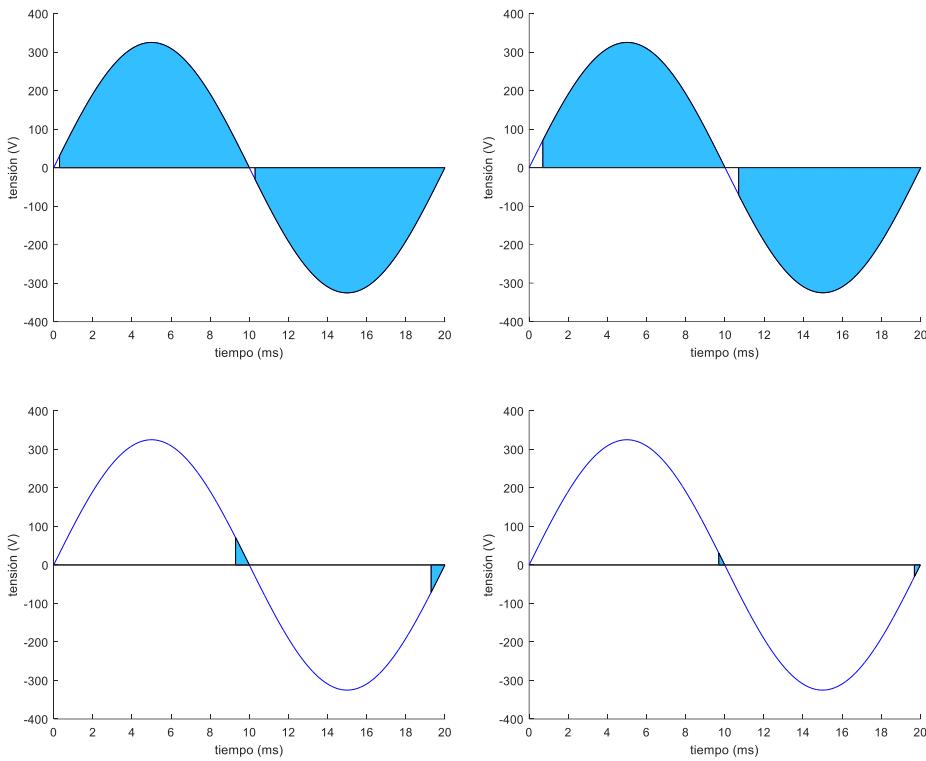


ILUSTRACIÓN 21: COMPARATIVA DE LA VARIACIÓN DEL ÁREA ENCERRADA EN LA SINUSOIDE PARA VARIACIONES DEL ÁNGULO DE DISPARO DE MEDIO MILISEGUNDO EN LAS REGIONES CERCANAS AL PASO POR CERO.

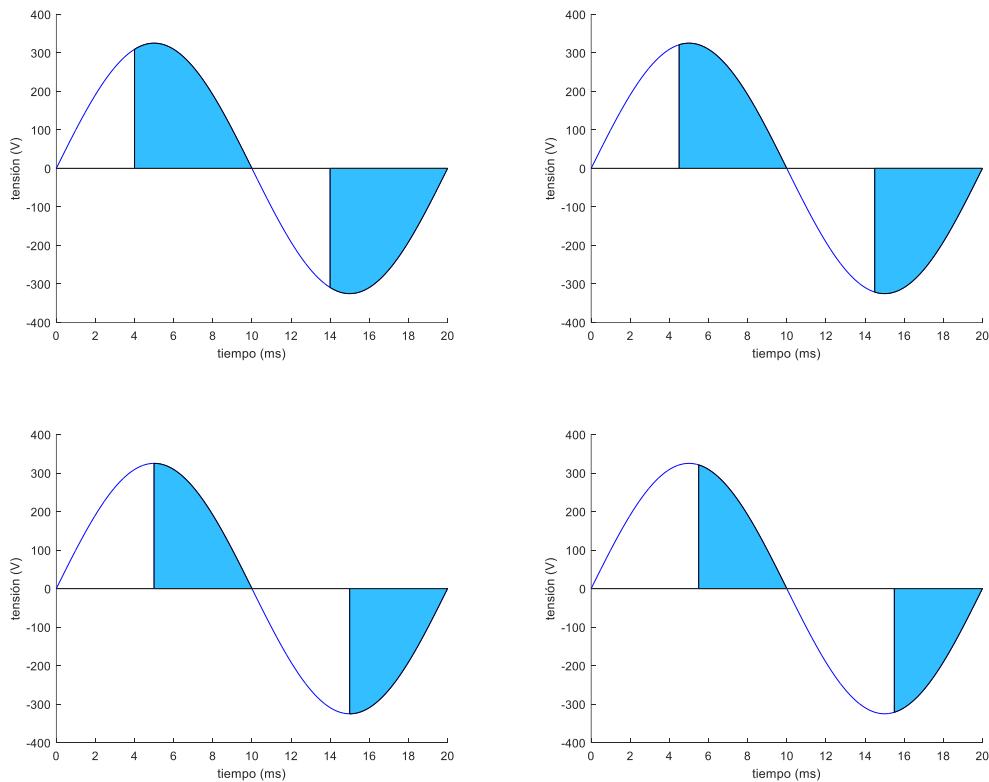


ILUSTRACIÓN 22: COMPARATIVA DE LA VARIACIÓN DEL ÁREA ENCERRADA EN LA SINUSOIDE PARA VARIACIONES DEL ÁNGULO DE DISPARO DE MEDIO MILISEGUNDO EN LA REGIÓN INTERMEDIA (TIEMPO DE DISPARO DESDE 4 A 5,5 MILISEGUNDOS).

En base a la Ilustración 20, Ilustración 21 e Ilustración 22, se puede sacar la conclusión de que existen dos regiones lineales, una en la parte central y otra en los extremos. Las zonas cercanas a los pasos por cero, como ya se comentó en el apartado 7.4, no son interesantes desde el punto de vista del control de los motores. Por tanto, se evitará en todo momento trabajar en ellas, programándose el software de control a tal efecto (véase activación-desactivación permanente de la señal de control en el apartado 7.4).

Este análisis de la sinusode de la red eléctrica es aplicable a las cargas de tipo puramente resistivo. Si en algún momento se pretende controlar alguna con este tipo de variador de potencia, éstas experimentaran un comportamiento muy lineal para áreas comprendidas entre el 10% y el 90% del área total de la sinusode.

Por otro lado, para cargas con componente inductiva, como son los motores que se pretenden controlar, la onda no es tan perfecta como en las resistivas, pues existe desfase entre tensión y corriente. En la carga resistiva, la corriente se extingue en el siguiente paso por cero, tras el disparo, de la sinusode. Sin embargo, en una carga de tipo inductivo, la corriente no se extingue inmediatamente, conmutando la tensión de red de valor (de positivo a negativo o viceversa) y manteniéndose la conducción, como se explicó en el apartado 3.3.3 (Circuito amortiguador (R3 y C1)), hasta que se extinga la corriente, siendo ese momento cuando el TRIAC deje de conducir.

7.5.3 Linealización del ventilador

Se procede a realizar la identificación de la respuesta del ventilador. Para ello se excita introduciendo diferentes valores de tiempo de disparo, midiéndose, una vez estabilizada, la salida.

Potencia (%)	Tiempo de disparo equivalente (μ s)	Velocidad (m/s)
100	0	3.36
85	1500	3.31
70	3000	3.13
50	5000	2.44
30	7000	0.87
20	8000	0.13

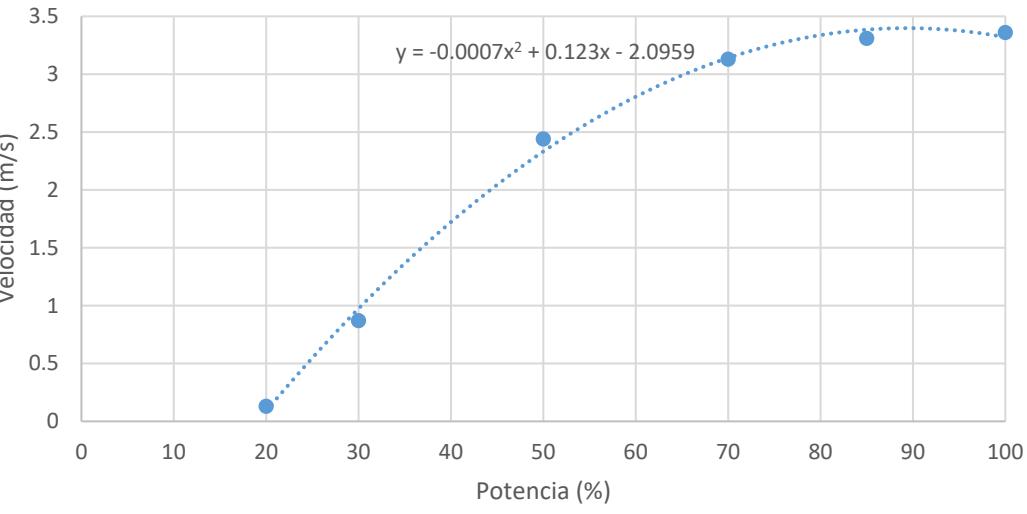


ILUSTRACIÓN 23: EXCITACIÓN-RESPUESTA DEL VENTILADOR

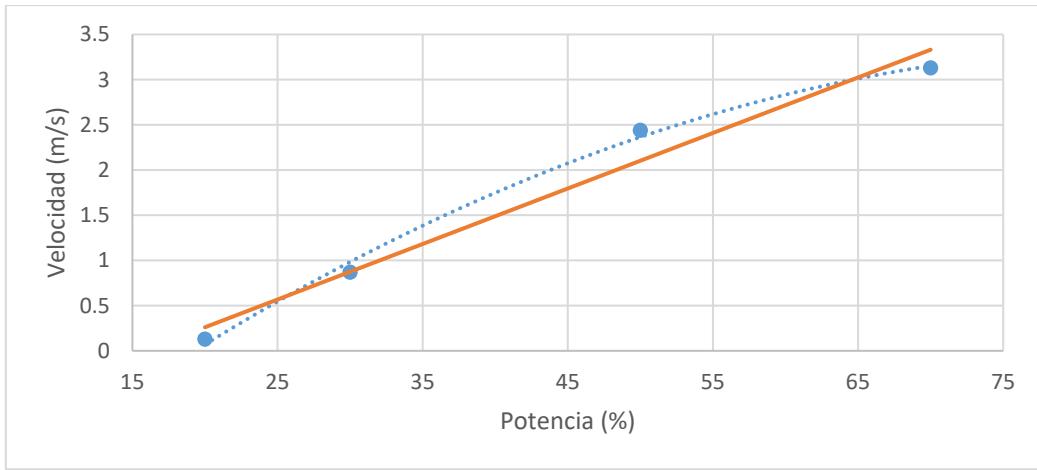


ILUSTRACIÓN 24: REGIÓN DE RESPUESTA LINEAL DEL VENTILADOR

Basándose en la Ilustración 24: Región de respuesta lineal del ventilador, se puede decir que el ventilador presenta una respuesta lineal bastante buena, pues se puede conseguir casi todo el rango de velocidades que abarca de manera lineal.

7.5.4 Linealización de la bomba de agua

Se realiza el mismo procedimiento seguido para el ventilador.

Potencia (%)	Tiempo de disparo equivalente (μ s)	Número de Reynolds
100	0	5374
85	1500	5380
70	3000	5300
65	3500	5375
60	4000	5080
55	4500	4574

50	5000	3480
45	5500	2800
40	6000	2050
35	6500	1356
30	7000	8

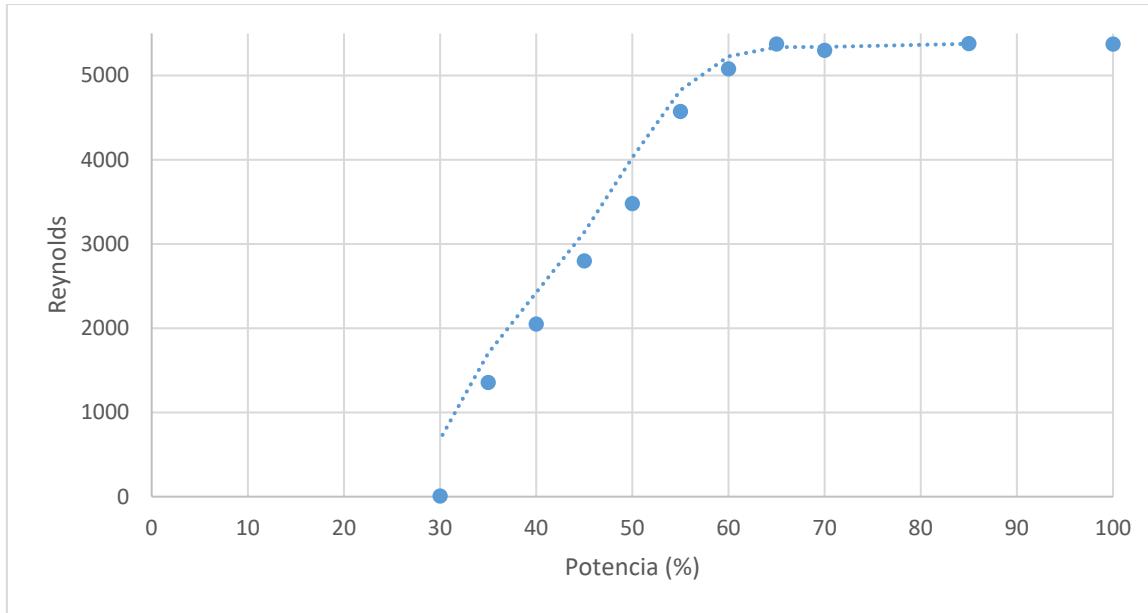


ILUSTRACIÓN 25: EXCITACIÓN-RESPUESTA DE LA BOMBA DE AGUA A REFRIGERAR.

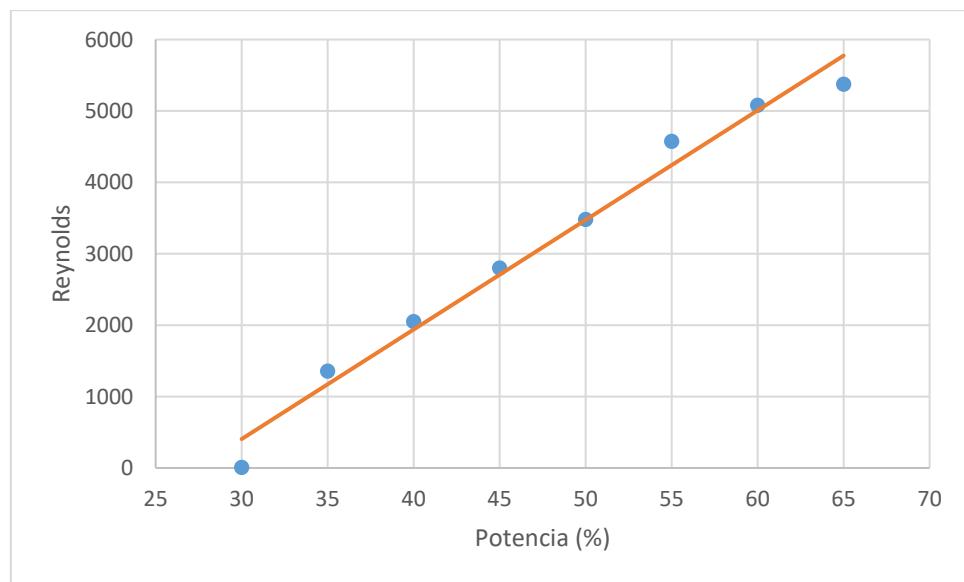


ILUSTRACIÓN 26: REGIÓN DE RESPUESTA LINEAL DE LA BOMBA DE AGUA A REFRIGERAR.

Obteniéndose, en base a la Ilustración 26, una respuesta lineal en todo el rango de número de Reynolds que puede conseguir.

7.5.5 Bomba de rociado

El número de Reynolds del agua de rociado es muy inferior al del agua a refrigerar (el valor máximo del primero es en torno a 10, mientras que el segundo sobrepasa 5300). Sin embargo, los caudalímetros instalados en ambas son iguales, y mientras que para la bomba de agua proporciona muy buenos resultados en todo el rango de variación, no proporciona la resolución necesaria para la bomba de rociado. Cuando sea instalado en ella uno de mayor resolución, podrá realizarse en esta bomba una linealización y sintonización, puesto que su PID sí que ha sido implementado en el software de control.

7.6 Controladores PID

En este apartado se exponen las bases del desarrollo de los PIDs (tiempo de muestreo, tiempo de respuesta...), la discretización llevada a cabo para controlar la resistencia de calentamiento mediante un PID y la sintonización de los parámetros.

7.6.1 Implementación de los PIDs

Se ha basado en una librería de Arduino (Beauregard, 2017). Para comprobar que esta librería cumple los requisitos del trabajo fin de grado, se ha llevado a cabo un análisis del código de la misma. Éste puede encontrarse en el anexo “Análisis de la implementación de la librería PID de Bred Beauregard”.

Los resultados del análisis plantean las siguientes dos cuestiones:

- Desbordamiento de la función *millis()*. Si la Arduino no es reseteada se produce una ejecución indebida del PID. Es un error que apenas tiene efecto porque sólo se produciría una vez cada 50 días (en el caso de que la Arduino no sea reseteada ninguna vez en esos 50 días). Aun así se ha considerado oportuno comentarlo en este análisis.
- Periodicidad en la llamada al PID. Debido a que se realiza mediante la comprobación del tiempo en el que se hizo la última llamada (con la función *millis()* mencionada en el párrafo anterior), este tiempo puede no ser totalmente preciso.

Lo ideal para solucionar los aspectos anteriores sería que la llamada a los PIDs se realizara de forma totalmente exacta mediante temporizaciones hardware. Analizando la carga de cómputo de *Arduino de Control* y el periodo de muestreo de los PIDs, se determina que esta imprecisión no va a suponer un problema, puesto que sólo se desviará unas decenas o centenas de microsegundos, por lo que el error de desviación es mínimo (en comparación con la ejecución de cada PID, que se ejecutará cada tres segundos).

La explicación de por qué los PID se ejecutan cada tanto tiempo es debido a que cada tres segundos es cuando se reciben nuevos datos del proceso (velocidad del ventilador, Reynolds de agua y acercamiento), y no tiene sentido ejecutar el PID varias veces con datos desactualizados, pues haría que la señal de control aumentase, de forma indebida, más de lo necesario. Un tiempo de muestreo tan grande tampoco es un problema debido a la lenta dinámica de respuesta de la planta. Ésta viene causada por los filtros que se aplican a las lecturas de los sensores. El filtrado es necesario debido a la gran variación de las lecturas, teniéndose que aplicar unos coeficientes de filtrado de gran valor. Por ejemplo, en el caso del ventilador, como el aire impulsado por él crea un régimen turbulento, para una velocidad filtrada de 2m/s, se obtienen variaciones de lectura instantáneas de hasta $\pm 0,5$ m/s.

7.6.2 Discretización de PID

Para controlar la resistencia de calentamiento es necesario realizar una discretización de la salida del PID, puesto que ésta varía en un rango de valores y la resistencia sólo admite valores discretos (encendida o apagada, todo o nada). La discretización se ha realizado siguiendo un ejemplo de la librería PID (PID Relay Output Example, 2017). Éste se basa en establecer una ventana temporal, y fijar un rango de variación de la salida del PID en un intervalo que va desde 0 hasta el tiempo máximo de esa ventana. Por otro lado, en el *loop* se evalúa el tiempo actual dentro de la ventana temporal, y si la salida del PID es superior a éste se activa el pin digital asociado a la resistencia. Es decir, si la ventana temporal es de X milisegundos, y la salida es Y (siendo Y siempre igual o inferior a X), la salida permanecerá activa durante Y milisegundos y desactiva durante X-Y milisegundos. Se explica mejor con un ejemplo:

- Se establece como ventana temporal 5000 milisegundos.
- Se limita la salida del PID entre 0 y 5000.
- Para una supuesta salida, que varía desde 2 a 9, se tiene una medida actual de 3.
- Se establece al PID una consigna de 7.
- El PID empieza a actuar, estableciendo en su salida, por ejemplo, 4000.
- Durante la ejecución del *loop* se monitoriza constantemente la ventana temporal de 5000 milisegundos. De estos 5000, la salida discreta permanecerá activa durante 4000 milisegundos, y desactiva durante 1000 milisegundos.
- Conforme se realizan nuevas mediciones, el PID varía la salida para conseguir alcanzar la referencia y, una vez alcanzada, mantenerse estable en ella. Estas variaciones de salida se traducen en un mayor o menor tiempo activo dentro de la ventana temporal.

7.6.3 Sintonización de los PIDs

Los parámetros se fijan mediante ensayo prueba-error, variando primero la constante proporcional hasta que se obtiene una respuesta cercana a la deseada, y después la constante integral para que el error de posición sea nulo. En todo momento la constante derivativa permanece a 0, para anular su acción (el PID implementado es de tipo paralelo, véase anexo de librería PID), pues la respuesta de la planta obedece a una dinámica muy lenta (como se explicó en el apartado 7.6.1).

Ventilador

Debido al régimen turbulento que produce al impulsar el aire, el filtrado aplicado a esta medición es el más agresivo. Como su respuesta es lenta, el PID implementado va a ser muy conservativo. Parámetros elegidos:

$$K_p: 10 \quad K_i: 2$$

Bomba de agua

En este caso, los parámetros obtenidos son muy pequeños. Esto es debido a que la variable medida es número de Reynolds, y éste es muy grande (en torno a 5000). Parámetros:

$$K_p: 0.01 \quad K_i: 0.005$$

Resistencia de calentamiento

Al contrario que en apartados anteriores, su valor es muy grande. Se debe a que la consigna *approach* toma valores pequeños (entre cero y diez), pero la actuación se realiza sobre la ventana temporal (de 5000 milisegundos) comentada en el apartado anterior. Parámetros:

$$K_p: 1500 \quad K_i: 5$$

8 Estudio de la comunicación

Como objetivo secundario, y una vez que el principal estuviese desarrollado e implementado, se establece el estudio de la comunicación del sistema, así como un análisis del número de Arduinos involucradas en la torre, para ver si puede ser reducido.

8.1 Problema de comunicación

Arduino Sensores se quedaba, en ocasiones, bloqueadas. Tras un análisis de sus funcionalidades, se determina que el problema viene dado por la comunicación I2C. Si se produce algún error en la recepción de alguno de los datos que requiere, queda bloqueada esperándolo.

Para evitarlo se ha implementado en *Arduino Caudalímetros* una monitorización de la comunicación I2C con *Arduino Sensores*. Si ésta no establece comunicación con ella tras un periodo de 30 segundos, se manda una señal de reseteo (mediante una conexión al pin de *RESET* en *Arduino Sensores*). Los criterios seguidos para implementarlo en *Arduino Caudalímetros* han sido:

- Proximidad a *Arduino Sensores*. Están situadas una junto a la otra, por lo que llevar la señal de reseteo desde uno de los pines digitales de *Caudalímetros*, al pin de *RESET* de *Sensores*, puede realizarse sin ningún inconveniente.
- La implementación de un *watchdog* para la comunicación I2C dentro de la funcionalidad de *Caudalímetros* es sencilla de realizar.

8.2 Unificación de Arduinos

Se va a analizar la funcionalidad de las distintas placas Arduino que integran el sistema de control y adquisición, y si sería conveniente reducir su número (unificando funciones en una misma placa). Para ello se va a hacer referencia al Esquema 1, en la introducción de la memoria descriptiva, donde se encuentran representadas las diferentes placas Arduino y su interrelación.

Ninguno de los códigos de las Arduinos se incluye como anexo, debido a que no han sido elaborados por el alumno. Éste simplemente los ha analizado para evaluar si podrían ser unificados.

8.2.1 Arduino para caudalímetros

Controla dos caudalímetros (y es el máximo número que puede controlar, pues Arduino Nano sólo dispone de dos interrupciones externas), situados en las tuberías correspondientes a las bombas de rociado y fluido a refrigerar. La evaluación del caudal se lleva a cabo mediante la medición del número de interrupciones provocadas (que se producen por cada rotación del

caudalímetro) y posteriormente un cálculo matemático. Debido a que éstas tienen lugar cada muy poco tiempo, esta Arduino ha de ser dedicada exclusivamente a la evaluación de caudal.

8.2.2 Arduino para temperaturas

Esta Arduino se encarga de controlar el integrado LTC2983. Contiene muchas líneas de código, pero la mayoría están destinadas a configuración. Su integración en otra Arduino sería posible.

8.2.3 Arduino Sensores

En ella se encuentra el código de Blynk (Blynk, 2017) que implementa la comunicación con el exterior. Es la maestra del bus I2C y se encarga de la lectura de los sensores de presión y humedad. Por la cantidad de código que ya tiene programado, no es recomendable emplearla para nada más.

8.2.4 Propuesta de unificación

La funcionalidad que implementa la Arduino de temperaturas podría implementarse dentro del código de otra Arduino. La solución más idónea sería en la Arduino para caudalímetros. A pesar de que anteriormente se ha indicado que ésta no debería dedicarse a otras tareas (a parte de la medición de caudal), sí que se podría añadir la lectura de los sensores de temperaturas, pues es una tarea que no consume mucho tiempo de cómputo del microcontrolador.

En la Arduino de caudalímetros se realiza, durante un segundo, la medición de las rotaciones de los caudalímetros mediante interrupciones, pero tras ello desactiva las interrupciones y realiza los cálculos necesarios para conocer los caudales que han circulado. Tras la realización de estos cálculos, y antes de comenzar una nueva monitorización de las rotaciones del caudalímetro, sería el lugar más oportuno para incorporar la medición de temperaturas. Además, supondría otra ventaja desde el punto de vista de la comunicación I2C, porque cuando se requieran datos a *Arduino Caudalímetros*, también se enviarían los de temperaturas, ahorrando a la Arduino maestra tenerse que comunicar con otra Arduino (y perdiendo tiempo en establecer la comunicación I2C y esperar la respuesta).

El código de *Temperaturas* también podría ser introducido en *Arduino Control*, pero esto no es recomendable, ya que *Arduino Control* está situada en una caja aparte, dedicada exclusivamente al control, por lo que habría que re-conexionar los sensores para esta caja. Además, ya no sería una caja de control, sino de control y adquisición, perdiendo modularidad el diseño.

9 Revisión de los objetivos

En este apartado se pretende hacer, a modo de resumen, una revisión de los objetivos planteados en el anteproyecto y los apartados en los que se han alcanzado. Muchos de ellos son referenciados repetidas veces a lo largo de la memoria, por lo que se van a indicar los más representativos de la consecución del objetivo.

Objetivos definidos en el anteproyecto:

- Diseño e implementación del algoritmo de control para la torre de refrigeración. Se pretende que este algoritmo:
 - Sea capaz de modificar la potencia transmitida a cada uno de los motores y de accionar la resistencia de calentamiento. (**véase 2. Detector de paso por cero y 3. Actuador sobre motor**)
 - Admita la introducción de valores manuales (porcentaje de funcionamiento de cada actuador). (**véase 4.3.1 Interfaz de usuario del Puerto Serie y 4.3.4 Comunicación I2C**)
 - Implemente método de control para seguimiento de referencias. (**véase 7.6 Controladores PID**)
 - Sea totalmente compatible con el sistema actual (ha de realizarse su integración en él). (**véase 7.2 Comunicación I2C**)
- Identificación de los problemas de comunicación actuales:
 - Propuesta (e implementación si procede) de un método de comunicación más eficaz para evitar que el control de la torre quede bloqueado. (**véase 8.1 Problema de comunicación**)
 - Propuesta (e implementación si procede) de mejora en la estrategia del sistema de control. (**véase 8.2.4 Propuesta de unificación**)

10 Conclusiones

Si bien hay partes que no se han enfocado de manera genérica (por ejemplo que las potencias de los diferentes motores y los tiempos de disparo absolutos se hayan guardado en variables globales en lugar de vectores), el resto de cálculos se han enfocado mediante bucles que permiten una fácil escalabilidad de la solución adoptada (y que en lugar de para 3 motores pueda utilizarse para más).

El fin último de esto es realizar una librería para Arduino, que permita el control de la potencia entregada a varias cargas de forma totalmente transparente al usuario, sin necesidad de incorporar en su código, de forma explícita, el cálculo de tiempos de disparo y las interrupciones. Debido a que ésta queda fuera del ámbito de realización del trabajo de fin de grado (y que requiere de conocimientos de programación orientada a objetos de los que el autor no dispone actualmente), queda como proyecto propio del autor para un futuro.

Referencias

- Aavid Thermalloy. (s.f.). *Heat Sink Catalogue. Referencia del disipador: 533902B02554G.*
- AC Phase Control.* (1 de Agosto de 2017). Obtenido de
<http://playground.arduino.cc/Main/ACPhaseControl>
- AC Zero Cross PWM Library.* (1 de Agosto de 2017). Obtenido de
<https://github.com/wmacevoy/ACZCPWM>
- Arduino. (1 de Agosto de 2017). *Sitio web oficial de Arduino.* Obtenido de
<https://www.arduino.cc>
- Arduino Playground.* (1 de Agosto de 2017). Obtenido de <http://playground.arduino.cc/>
- Atmel. (2016). *ATmega328/P: Hoja de características completa.*
- Beauregard, B. (1 de Agosto de 2017). *Arduino PID Library.* Obtenido de
<https://playground.arduino.cc/Code/PIDLibrary>
- Blynk. (1 de Agosto de 2017). *Blynk.* Obtenido de <http://www.blynk.cc/>
- Fairchild Semiconductor. (2006). *Application Note AN-3003. Phase Crossing TRIAC Drivers.*
- J.Escudero, A. R. (s.f.). *Electrónica de Potencia.* Córdoba: Ediciones Litopress.
- Kotsopodis, P. (1 de Agosto de 2017). *Alfadex.* Obtenido de
<http://www.alfadex.com/2014/02/dimming-230v-ac-with-arduino-2/>
- PID Relay Output Example. (1 de Agosto de 2017). *Arduino Playground.* Obtenido de
<https://playground.arduino.cc/Code/PIDLibraryRelayOutputExample>
- Sparkfun. (1 de Agosto de 2017). *Logic Levels.* Obtenido de
<https://learn.sparkfun.com/tutorials/logic-levels>
- STMicroelectronics. (2008). *Application Note AN2703. Parameter list for TRIACs.*
- STMicroelectronics. (2010). *Hoja de características del TRIAC BTA16.*
- Texas Instruments. (1998). *Hoja de características del Circuito Integrado MOC3021.*
- Trujillo, F. D., Pozo, A., & Triviño, A. (2011). *Cálculo de disipador de calor.* Málaga: OCW de la Universidad de Málaga.
- Variador de potencia comercial.* (1 de Agosto de 2017). Obtenido de
<https://www.aliexpress.com/item/2000W-High-power-electronic-voltage-regulator-for-dimmer-speed-temperature-adjustment-Integrated-Circuits/32350637578.html?spm=2114.search0302.4.70.zVDyeK>
- Vishay Semiconductors. (2010). *Hoja de características del Circuito Integrado 4N25.*
- Vishay Semiconductors. (2011). *Hoja de características del Circuito Integrado H11AA1.*

ANEXOS

REFERENCIAS DE LA TABLA DE PRESUPUESTO

Referencias de la tabla de presupuestos. Enlaces y precios de consulta válidos a 17 de Agosto de 2017:

- [1] Disipadores: <http://es.rs-online.com/web/p/disipadores/0403162/>
- [2] BTA16: <http://es.rs-online.com/web/p/products/7140506/>
- [3] MOC3021: <http://es.rs-online.com/web/p/products/6912265/>
- [4] H11AA1: <http://es.rs-online.com/web/p/products/6711399/>
- [5] Placas matriz para prototipo: https://www.amazon.es/Mudder-Universal-Prototipo-Multiple-Tama%C3%B1os/dp/B01ER06KXE/ref=sr_1_3?ie=UTF8&qid=1502981057&sr=8-3&keywords=placa+prototipo
- [6] Arduino Nano: https://www.amazon.es/Arduino-Elegoo-ATmega328P-Compatible-Paquete/dp/B0716T2L77/ref=sr_1_1?s=electronics&ie=UTF8&qid=1502981123&sr=1-1-spons&keywords=placa+arduino+nano&psc=1
- [7] Condensadores 0,1 μ F: <http://es.rs-online.com/web/p/products/8741412/>
- [8] Conjunto de todos los valores necesarios de resistencias, 10 unidades de cada uno.
 - 56k Ω : <http://es.rs-online.com/web/p/products/2142184/>
 - 10k Ω : <http://es.rs-online.com/web/p/products/0132731/>
 - 1k Ω : <http://es.rs-online.com/web/p/products/2141951/>
 - 220 Ω : <http://es.rs-online.com/web/p/products/0135819/>
 - 100 Ω : <http://es.rs-online.com/web/p/products/0132258/>
- [9] Interruptor balancín iluminado: <http://es.rs-online.com/web/p/interruptores-de-balancin/3779759/>
- [10] Caja eléctrica impermeable: https://www.amazon.es/Plastico-Conexiones-Impermeable-ELEGANT-Electr%C3%B3nica/dp/B01N5FB7I0/ref=sr_1_2?s=electronics&ie=UTF8&qid=1502982928&sr=1-2&keywords=caja%2Belectrica&th=1
- [11] Conectores impermeables: <https://www.aliexpress.com/item/SP13-Docking-Waterproof-Aviation-connector-IP68-LED-power-Cable-Connector-In-line-cable-connector-auto-electrical/32757350880.html?spm=a2g0s.13010208.9999999.262.o76lFK>
- [12] Terminal PCB 3 vías: <http://es.rs-online.com/web/p/products/2204276/>
- [13] Terminal PCB 2 vías: <http://es.rs-online.com/web/p/products/1930564/>

ANÁLISIS DE LA
IMPLEMENTACIÓN DE
LA LIBRERÍA PID DE
BRED BEAUREGARD

Para la implementación de los PIDs se ha escogido una **librería para Arduino desarrollada por Bred Beauregard**. Se adjunta el código (tanto de ejemplo de utilización como del interior de la librería) y se comenta, en el margen, qué está realizando el autor en cada caso en relación. El objetivo es comprobar que el código puede ser integrado de forma efectiva en la torre de refrigeración. Como algunos comentarios son más extensos de lo que permite el espacio al margen del código, se referencia con un número que corresponde a una de las explicaciones desarrolladas a continuación:

Referencia 1: Se produce una ejecución indebida del PID si la placa Arduino no es reseteada en 50 días. Esto es debido a la forma de calcular el tiempo (para ver si se ha superado el periodo T) mediante la función *millis*, que devuelve el número de milisegundos desde que el programa comenzó su ejecución (en forma de *unsigned long*). Al producirse el desbordamiento de este *unsigned long* se comienza desde cero por lo que la condición *timeChange>=SampleTime* se cumpliría de forma inmediata y se ejecutaría el cómputo incluso aunque no se haya superado el *SampleTime*. Debido a que esto ocurre sólo una vez cada cincuenta días es un problema que pasa fácilmente desapercibido, pero se ha considerado oportuno comentarlo en este análisis.

Referencia 2: El autor declara esta variable (y la usa a la hora de calcular el PID) para poder efectuar cambios en los parámetros del PID durante su funcionamiento y que no se den cambios bruscos en la salida. El razonamiento que sigue se basa en que, para cambiar los parámetros actuales, el sistema ha de estar estable en un punto de trabajo (así la acción proporcional y derivativa en la señal de control será mínima), y juntando el término *Ki* y el error acumulado en la misma variable, al realizarse un cambio en *Ki* no se estaría produciendo un cambio en el error acumulado. Si en el cálculo de la salida se hiciese

```
Output = kp * error + ki * errSum - kd * dInput; donde  
error = Setpoint - Input;  
errSum += error;
```

En lugar de

```
Output = kp * error + ITerm- kd * dInput;
```

Se tendría que un cambio en los parámetros del PID, tanto *kp* como *kd* no influirían apenas en la salida (ya que el error y la diferencia entre la salida medida actual y anterior serían muy pequeñas) pero una variación de *ki* (multiplicada por la suma de todos los errores anteriores) tendría un gran efecto sobre la salida (por ejemplo, al reducir *ki* a la mitad ¡se reduciría la salida a la mitad!). Con la variable *ITerm* este problema queda solucionado.

Referencia 3: Eliminación de picos producidos por la acción derivativa. En la asignatura de *Laboratorio de Control de Procesos* se ha estudiado que lo habitual es aplicar un filtro en la acción derivativa. Sin embargo, el autor de este PID ha optado por un enfoque diferente: En lugar de implementar la acción derivativa sobre la derivada del error (que provoca picos cada vez que la referencia es cambiada, ya que el error es calculado como la diferencia de la referencia y la salida de la planta), declara dos variables (*dInput* y *lastInput*) en las que se apoya para realizar la derivada de la salida medida de la planta (utilizando el instante actual e inmediatamente anterior).

```

1  ****
2  * PID Basic Example
3  * Reading analog input 0 to control analog PWM output 3
4  ****
5
6 #include <PID_v1.h>
7
8 #define PIN_INPUT 0
9 #define PIN_OUTPUT 3
10
11 //Define Variables we'll be connecting to
12 double Setpoint, Input, Output;
13
14 //Specify the links and initial tuning parameters
15 double Kp=2, Ki=5, Kd=1;
16 PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
17
18 void setup()
19 {
20     //initialize the variables we're linked to
21     Input = analogRead(PIN_INPUT);
22     Setpoint = 100;
23
24     //turn the PID on
25     myPID.SetMode(AUTOMATIC);
26 }
27
28 void loop()
29 {
30     Input = analogRead(PIN_INPUT);
31     myPID.Compute();
32     analogWrite(PIN_OUTPUT, Output);
33 }
34
35
36

```

Declara las variables y los parámetros utilizados por el PID y crea uno llamado "myPID"

Inicializa la entrada, establece la referencia y conecta el PID en modo automático

Efectúa la lectura de la magnitud que controla, la guarda en la variable "Input" y ejecuta los cálculos del PID, escribiendo finalmente la salida del PID ("output") sobre el actuador.

En este punto salta a la vista que la realización de la llamada al PID no es la más adecuada desde el punto de vista de la periodicidad. Además, de esta forma se estaría empleando la placa Arduino sólo para la ejecución del control, sin poder "paralelizar" ninguna otra tarea. Lo adecuado sería llamar a "Compute" mediante temporizadores hardware (haciendo uso de uno de los timers de los que cuenta el microcontrolador ATmega168, núcleo de Arduino). Para ello habría que modificar el código que se muestra en la siguiente página (y donde se realiza la medición de tiempo mediante la función millis) para que no se ejecute antes del periodo "T").

```

1  ****
2  * Arduino PID Library - Version 1.1.1
3  * by Brett Beauregard <br3ttb@gmail.com> brettbeauregard.com
4  *
5  * This Library is licensed under a GPLv3 License
6  ****
7
8 #if ARDUINO >= 100
9   #include "Arduino.h"
10
11 #else
12   #include "WProgram.h"
13
14 #endif
15
16 #include <PID_v1.h>
17
18 /*Constructor (...)*
19  *      The parameters specified here are those for which we can't set up
20  *      reliable defaults, so we need to have the user set them.
21  */
22
23 PID::PID(double* Input, double* Output, double* Setpoint,
24           double Kp, double Ki, double Kd, int ControllerDirection)
25 {
26
27     myOutput = Output;
28     myInput = Input;
29     mySetpoint = Setpoint;
30     inAuto = false;
31
32     PID::SetOutputLimits(0, 255);           //default output limit corresponds to
33                                         //the arduino pwm limits
34
35     SampleTime = 100;                     //default Controller Sample Time is 0.1 seconds
36
37     PID::SetControllerDirection(ControllerDirection);
38     PID::SetTunings(Kp, Ki, Kd);
39
40     lastTime = millis() - SampleTime;
41 }
42
43 /* Compute() ****
44  *      This, as they say, is where the magic happens.  this function should be called
45  *      every time "void loop()" executes.  the function will decide for itself whether a new
46  *      pid Output needs to be computed.  returns true when the output is computed,
47  *      false when nothing has been done.
48  */
49
50 bool PID::Compute()
51 {
52     if(!inAuto) return false;
53     unsigned long now = millis();
54     unsigned long timeChange = (now - lastTime);

```

Parte correspondiente a la
declaración del PID "myPID".

Medición de tiempo. Esta parte y la comprobación siguiente
se eliminarían en caso de ser llamada mediante interrupción.

```

52 if(timeChange>=SampleTime)
53 {
54     /*Compute all the working error variables*/
55     double input = *myInput;
56     double error = *mySetpoint - input;
57     ITerm+= (ki * error);
58     if(ITerm > outMax) ITerm= outMax;
59     else if(ITerm < outMin) ITerm= outMin;
60     double dInput = (input - lastInput);
61
62     /*Compute PID Output*/
63     double output = kp * error + ITerm- kd * dInput;
64
65     if(output > outMax) output = outMax;
66     else if(output < outMin) output = outMin;
67     *myOutput = output;
68
69     /*Remember some variables for next time*/
70     lastInput = input;
71     lastTime = now;
72     return true;
73 }
74 else return false;
75 }

76
77
78 /* SetTunings(...)*****
79 * This function allows the controller's dynamic performance to be adjusted.
80 * it's called automatically from the constructor, but tunings can also
81 * be adjusted on the fly during normal operation
82 ****/
83 void PID::SetTunings(double Kp, double Ki, double Kd)
84 {
85     if (Kp<0 || Ki<0 || Kd<0) return;
86
87     dispKp = Kp; dispKi = Ki; dispKd = Kd;
88
89     double SampleTimeInSec = ((double)SampleTime)/1000;
90     kp = Kp;
91     ki = Ki * SampleTimeInSec;
92     kd = Kd / SampleTimeInSec;
93
94     if(controllerDirection ==REVERSE)
95     {
96         kp = (0 - kp);
97         ki = (0 - ki);
98         kd = (0 - kd);
99     }
100
101
102 /* SetSampleTime(...) *****

```

Si el tiempo transcurrido es mayor al periodo de muestreo, entonces es ejecutado el cálculo de los valores del PID. Observación al respecto en referencia 1.

ITerm: Referencia 2.

Implementación mecanismo antiwindup limitando la acción integral.

Eliminación picos provocados por parte derivativa: Ref. 3.

La implementación realizada corresponde a un esquema de control de un PID paralelo.

Limitación del valor máximo y mínimo de la salida para proteger al actuador.

En caso de que la acción de control sea inversa (por ejemplo en refrigeración, al aplicar mayor señal de control aumenta la potencia del actuador (equipo de refrigeración) y la temperatura disminuye).

```

103 * sets the period, in Milliseconds, at which the calculation is performed
104 ****
105 void PID::SetSampleTime(int NewSampleTime)
106 {
107     if (NewSampleTime > 0)
108     {
109         double ratio = (double)NewSampleTime
110             / (double)SampleTime;
111         ki *= ratio;
112         kd /= ratio;
113         SampleTime = (unsigned long)NewSampleTime;
114     }
115 }
116
117 /* SetOutputLimits(...)*****
118 *      This function will be used far more often than SetInputLimits. while
119 *      the input to the controller will generally be in the 0-1023 range (which is
120 *      the default already,) the output will be a little different. maybe they'll
121 *      be doing a time window and will need 0-8000 or something. or maybe they'll
122 *      want to clamp it from 0-125. who knows. at any rate, that can all be done
123 *      here.
124 ****
125 void PID::SetOutputLimits(double Min, double Max)
126 {
127     if(Min >= Max) return;
128     outMin = Min;
129     outMax = Max;
130
131     if(inAuto)
132     {
133         if(*myOutput > outMax) *myOutput = outMax;
134         else if(*myOutput < outMin) *myOutput = outMin;
135
136         if(ITerm > outMax) ITerm= outMax;
137         else if(ITerm < outMin) ITerm= outMin;
138     }
139 }
140
141 /* SetMode(...)*****
142 * Allows the controller Mode to be set to manual (0) or Automatic (non-zero)
143 * when the transition from manual to auto occurs, the controller is
144 * automatically initialized
145 ****
146 void PID::SetMode(int Mode)
147 {
148     bool newAuto = (Mode == AUTOMATIC);
149     if(newAuto && !inAuto)
150     { /*we just went from manual to auto*/
151         PID::Initialize();
152     }
153     inAuto = newAuto;

```

Establecimiento de los límites del actuador.

Implementación de mecanismo de transferencia bumpless (paso del modo manual a automático). Para ello se llama a la función "inicializar".

```

154 }
155
156 /* Initialize()*****
157 * does all the things that need to happen to ensure a bumpless transfer
158 * from manual to automatic mode.
159 *****/
160 void PID::Initialize()
161 {
162     ITerm = *myOutput;
163     lastInput = *myInput;
164     if(ITerm > outMax) ITerm = outMax;
165     else if(ITerm < outMin) ITerm = outMin;
166 }
167
168 /* SetControllerDirection(...)*****
169 * The PID will either be connected to a DIRECT acting process (+Output leads
170 * to +Input) or a REVERSE acting process(+Output leads to -Input.) we need to
171 * know which one, because otherwise we may increase the output when we should
172 * be decreasing. This is called from the constructor.
173 *****/
174 void PID::SetControllerDirection(int Direction)
175 {
176     if(inAuto && Direction != controllerDirection)
177     {
178         kp = (0 - kp);
179         ki = (0 - ki);
180         kd = (0 - kd);
181     }
182     controllerDirection = Direction;
183 }
184
185 /* Status Funcions*****
186 * Just because you set the Kp=-1 doesn't mean it actually happened. these
187 * functions query the internal state of the PID. they're here for display
188 * purposes. this are the functions the PID Front-end uses for example
189 *****/
190 double PID::GetKp(){ return dispKp; }
191 double PID::GetKi(){ return dispKi; }
192 double PID::GetKd(){ return dispKd; }
193 int PID::GetMode(){ return inAuto ? AUTOMATIC : MANUAL; }
194 int PID::GetDirection(){ return controllerDirection; }
195
196

```

La función "inicializar" actualiza los valores del término integral a los de la salida actual: es un mecanismo bumpless para paso de modo manual a automático.

En caso de que la acción de control sea inversa (por ejemplo en refrigeración, al aplicar mayor señal de control aumenta la potencia del actuador (equipo de refrigeración) y la temperatura disminuye).

Funciones para obtener los valores de los parámetros del PID.

SOFTWARE DESARROLLADO PARA *ARDUINO CONTROL*

Arduino_control.ino

```

1  /*
2   * Este fichero forma parte del "SOFTWARE PARA PLACA CONTROLADORA DE MOTORES",
3   * desarrollado por Francisco Luque Luque para el trabajo fin de grado "DISEÑO
4   * E IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL PARA UNA TORRE DE REFRIGERACIÓN",
5   * del Grado en Ingeniería Electrónica Industrial de la Universidad de Córdoba (2017)
6   *
7   * Este es el fichero principal del software de control.
8   */
9  ****
10  *
11  *          SOFTWARE PARA PLACA CONTROLADORA DE MOTORES
12  *
13  * Autor: Francisco Luque Luque
14  * Contacto: luqueluquefrancisco@gmail.com
15  * Versión 1.0, 30 de Julio de 2017.
16  ****
17  *
18  /*
19   * Mediante este programa se controlan los tres variadores de potencia
20   * que actúan sobre los tres motores de la torre (ventilador, bomba de rociado
21   * y bomba de fluido a refrigerar), así como de la resistencia de calentamiento
22   * del tanque de fluido a refrigerar.
23  */
24
25
26 #include "configuracion_software.h" // En este fichero se encuentran todas las definiciones
27 // y declaraciones de variables.
28
29 void setup() {
30     configuracion_e_inicializacion_de_pines();
31     Setup_Fan_PID();
32     Setup_Spray_PID();
33     Setup_Water_PID();
34     Setup_Heating();
35     inicializar_sensores();
36     inicializar_comunicacion_I2C();
37     #if defined(MODO_DEPURACION)
38         inicializar_comunicacion_Puerto_Serie();
39     #endif
40     reset_controles();
41 }
42
43 void loop() {
44     lectura_y_proteccion_por_sensores();
45
46     if (recepcion_I2C_disponible == 1) {
47         recepcion_I2C_disponible=0;
48         tratamiento_recepcion_I2C(cabecera_trama, v_data_trama);
49         #if defined(MODO_DEPURACION)
50             hay_valores_para_mostrar=1;

```

```
51     #endif
52 }
53
54 if(PID_Fan.Compute() == 1)    calculo_valores_actuacion('f',Fan_powerPercentage);
55 if(PID_Spray.Compute() == 1)   calculo_valores_actuacion('s',Spray_powerPercentage);
56 if(PID_Water.Compute() == 1)   calculo_valores_actuacion('w',Water_powerPercentage);
57 PID_Heating.Compute();
58 Heating_action(HeatingWindow, output_Heating, HeatingWindow_inicial);
59
60 if (watchdog_comunicacion_I2C(TIEMPO_WATCHDOG_COMUNICACION_I2C) == 1) {
61     reset_controles();
62     #if defined(MODO_DEPURACION)
63         Serial.println("Se ha realizado un reset software por el watchdog de comunicación I2C.");
64     #endif
65 }
66
67 #if defined(MODO_DEPURACION)
68 if(hay_valores_para_mostrar == 1) {
69     hay_valores_para_mostrar = 0;
70     mostrar_valores();
71 }
72 #endif
73 }
74
75
76
```

configuracion_software.h


```

51 // VARIABLES GLOBALES
52
53 volatile boolean hay_valores_esperando_interrupcion=0;      // Para actualizar los valores de
54                                         // control tras el paso por cero
55
56 // Para actuación
57
58 volatile byte contador_activaciones=0; // Variables para la activación y desactivación de
59 volatile byte contador_desactivaciones; // los pines de los actuadores.
60 boolean desactivacion_de_motor[3];
61 boolean desactivacion_de_motor Esperando_interrupcion[3];
62 volatile boolean hay_QUE_desactivar=0;
63
64 byte pines_ordenados[3] = {PIN_F, PIN_W, PIN_S}; // Vector que contiene los pines ordenados
65                                         // por orden de disparo
66 byte pines_ordenados Esperando_interrupcion[3] = {PIN_F, PIN_W, PIN_S}; // Para el volcado sobre los
67                                         // que realmente actúan
68 boolean v_estado_pines_durante_activacion[3] = {1, 1, 1}; // Valor de cada pin (un 0 ó
69                                         // un 1) para la primera fase
70                                         // de la temporización (que
71                                         // corresponde con las activaciones
72                                         // de los motores).
73 boolean v_estado_pines_durante_activacion Esperando_interrupcion[3] = {1, 1, 1};
74
75 boolean v_estado_pines_durante_desactivacion[3] = {1, 1, 1}; // Para las desactivaciones
76 boolean v_estado_pines_durante_desactivacion Esperando_interrupcion[3] = {1, 1, 1};
77
78 unsigned int v_times[3] = {1, 1, 1}; // Vector de tiempos relativos para actuación
79 unsigned int v_times Esperando_interrupcion[3] = {1, 1, 1};
80
81 unsigned int tiempo_disparo_Fan = 10000; // Tiempo absoluto en us segundos, correspondiente a cada
82 unsigned int tiempo_disparo_Water = 10000; // motor, para su disparo.
83 unsigned int tiempo_disparo_Spray = 10000;
84
85 const float diez_entre_pi = 3.1831; // Usada para el cálculo del tiempo de disparo
86
87 double Fan_powerPercentage = 0; // Potencias de cada motor, en valores de 0 a 100%
88 double Spray_powerPercentage = 0;
89 double Heating_powerPercentage = 0;
90 double Water_powerPercentage = 0;
91
92 double Spray_previousPowerPercentage; // Potencias "anteriores". Sirve para guardar los valores actuales
93 double Water_previousPowerPercentage; // de potencias antes de ponerlos al 0% debido a lectura de sensor
94 boolean Spray_powerPercentage_blocked = 0; // Si están a cero significa que no estaban bloqueadas.
95 boolean Water_powerPercentage_blocked = 0;
96
97 // Para comunicación
98 volatile boolean recepcion_I2C_disponible = 0;
99 volatile byte cabecera_trama; // Cabecera de la trama de comunicación
100 volatile float v_data_trama[4]; // En orden: datos de Fan, Spray, Heating y Water.

```

```

101
102 volatile unsigned long tiempo_inicial_watchdog_recepcion_I2C; // Variable global que utiliza watchdog de recepción I2C
103
104 volatile byte contador_de_envios_I2C=0;
105
106 #if defined(MODO_DEPURACION) // Para mostrar los valores por el puerto serie
107     boolean hay_valores_para_mostrar=0;
108 #endif
109
110 // Variables de sensores de nivel
111 boolean water_level_sensor; // Estado del sensor de agua
112 boolean water_level_sensor_ant; // Estado anterior del sensor de agua
113
114 boolean spray_level_bottom_sensor; // Sensores de deposito de rociado inferior
115 boolean spray_level_bottom_sensor_ant;
116
117
118 boolean histeresis_water_level_activa; // Para protección de las bombas contra
119 unsigned long tiempo_histeresis_water_level_se_activa; // activación y desactivación constante
120 boolean histeresis_spray_level_bottom_activa;
121 unsigned long tiempo_histeresis_spray_level_bottom_se_activa;
122
123 // Variables de control PID
124
125 // FAN
126     double Fan_speed; // Variable de proceso, medida
127     double setpoint_Fan; // Consigna a la que se quiere tener la variable de proceso
128     double Kp_Fan = 10, Ki_Fan = 2, Kd_Fan = 0; // Parámetros del PID
129     PID PID_Fan(&Fan_speed, &Fan_powerPercentage, &setpoint_Fan, Kp_Fan, Ki_Fan, Kd_Fan, P_ON_E, DIRECT);
130
131 // WATER PUMP
132     double Reynolds_W; // Variable de proceso, medida
133     double setpoint_Water;
134     double Kp_Water = 0.01, Ki_Water = 0.005, Kd_Water = 0;
135     PID PID_Water(&Reynolds_W, &Water_powerPercentage, &setpoint_Water, Kp_Water, Ki_Water, Kd_Water, P_ON_E, DIRECT);
136 // Valores sintonizados manualmente: Kp: 0.01, Ki:0.005, Kd=0.
137
138 // SPRAY PUMP
139     double Reynolds_S; // Variable de proceso, medida
140     double setpoint_Spray;
141     double Kp_Spray = 10, Ki_Spray = 0, Kd_Spray = 0;
142     PID PID_Spray(&Reynolds_S, &Spray_powerPercentage, &setpoint_Spray, Kp_Spray, Ki_Spray, Kd_Spray, P_ON_E, DIRECT);
143
144 // HEATING
145     double alpha_EMA = 0.1; // Parámetro para el filtrado
146     double ApproachHR = 0; // Variable de proceso, medida
147     double ApproachHR_EMA; // Variable de proceso, con filtro
148     double setpoint_Heating; // Consigna, mismas unidades que approach
149     double output_Heating; // Señal de control
150     int HeatingWindow = 5000; // Discretización del PID

```

```

151 unsigned long HeatingWindow_inicial;
152 double Kp_Heating = 1500, Ki_Heating = 5, Kd_Heating = 0; // Parámetros del PID
153 PID PID_Heating(&ApproachHR_EMA, &output_Heating, &setpoint_Heating, Kp_Heating, Ki_Heating, Kd_Heating, P_ON_E, DIRECT);
154
155
156
157 // -----
158 // MENSAJES ALMACENADOS EN MEMORIA ROM
159 // -----
160
161 #if defined(MODO_DEPURACION)
162     const PROGMEM char mensaje_menu_de_seleccion[]={"Commands:\n"
163                                         "First command: Mode selection\n"
164                                         "\t'S': Show the actual values (no second command needed)\n"
165                                         "\t'H' Approach (no second command needed)\n"
166                                         "\t'A' Mode: automatic\n"
167                                         "\t'M' Mode: manual\n"
168                                         "\t'R' Stop all controls (no second command needed)\n"
169                                         "Second command: Motor selection\n"
170                                         "\t'F' Fan power\n"
171                                         "\t'W' Waterpump power\n"
172                                         "\t'S' Spraypump power\n"
173                                         "\t'H' Heating control\n\n"
174                                         "Waiting for user input: \n"};
175
176 const PROGMEM char mensaje_de_error[]="\n\nERROR\nLo introducido no esta definido en el menu\n\n";
177
178 const PROGMEM char example_message[]="\nEXAMPLE: \tFan in manual mode with a reference of 50%: 'MF50' \n\t"
179 "Instead of the input showed below, is recommended the next: 'MF' '50' \n\t"
180 "(what means, write MF and send it. Then do the same with the value 50).\n\n";
181
182 // Mensajes usados para depuración:
183 const PROGMEM char mensaje_calculo1[]{"Tiempos de disparo absolutos(F,W,S): "};
184 const PROGMEM char mensaje_calculo2[]{"Tiempos de disparo relativos: "};
185 const PROGMEM char mensaje_calculo3[]{"Pines ordenados por tiempo de disparo: "};
186 const PROGMEM char mensaje_calculo4[]{"Estado del pin durante activacion y estado del pin durante desactivacion:\n"};
187 const PROGMEM char mensaje_calculo5[]{"Vector de desactivaciones: "};
188
189 #endif
190

```

configuraciones.ino

```
1  /*
2   * Este fichero forma parte del "SOFTWARE PARA PLACA CONTROLADORA DE MOTORES",
3   * desarrollado por Francisco Luque Luque para el trabajo fin de grado "DISEÑO
4   * E IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL PARA UNA TORRE DE REFRIGERACIÓN",
5   * del Grado en Ingeniería Electrónica Industrial de la Universidad de Córdoba (2017)
6   *
7   * En este fichero se encuentran las funciones de inicialización de pines y de
8   * restablecimiento de variables.
9   */
10
11 // -----
12 //           CONFIGURACIÓN E INICIALIZACIÓN DE PINES
13 // -----
14
15 void configuracion_e_inicializacion_de_pines(void) {
16
17     noInterrupts();
18
19     // ENTRADAS
20     // Detección de paso por cero
21     pinMode(PIN_DETETC_PASO_0, INPUT);
22
23     // Sensores
24     pinMode(PIN_WATER_LEVEL,      INPUT);
25     pinMode(PIN_SPRAY_LEVEL_BOTTOM, INPUT);
26 //    pinMode(PIN_SPRAY_LEVEL_TOP,    INPUT);
27
28     // SALIDAS
29     pinMode(PIN_F, OUTPUT);
30     pinMode(PIN_S, OUTPUT);
31     pinMode(PIN_W, OUTPUT);
32     pinMode(PIN_H, OUTPUT);
33     pinMode(PIN_HEATING_STATUS, OUTPUT);
34
35     // Inicializaciones
36     digitalWrite(PIN_F, 1); // Desactivación de motores
37     digitalWrite(PIN_S, 1);
38     digitalWrite(PIN_W, 1);
39     digitalWrite(PIN_H, 0); // Desactivación de heating e indicador del mismo
40     digitalWrite(PIN_HEATING_STATUS, 0);
41
42     // Configuración interrupciones
43     attachInterrupt(digitalPinToInterrupt(PIN_DETETC_PASO_0), servicio_int_ext, FALLING);
44
45     interrupts();
46 }
47
48 // -----
49 //           RESET DE CONTROLES
50 // -----
```

```

51 void reset_controles (void) {
52     /* Esta función se encarga de inicializar todos los controles de la torre,
53      * apagando todos los motores y reseteando las variables a valor 0.
54      */
55     byte i;
56     byte pines_ordenados_tmp[3]={PIN_F, PIN_W, PIN_S}; // Orden de los pines
57
58     // DESACTIVACIÓN DE PIDS
59     PID_Fan.SetMode(MANUAL);
60     PID_Spray.SetMode(MANUAL);
61     PID_Heating.SetMode(MANUAL);
62     PID_Water.SetMode(MANUAL);
63
64     // Actuación
65     Fan_powerPercentage=0;           Water_powerPercentage=0;           Spray_powerPercentage=0;
66     Heating_powerPercentage=0;       Water_previousPowerPercentage=0;   Spray_previousPowerPercentage=0;
67     tiempo_disparo_Fan=10000;       tiempo_disparo_Water=10000;       tiempo_disparo_Spray=10000;
68
69     digitalWrite(PIN_H, LOW);        // Desactivación de heating
70     digitalWrite(PIN_HEATING_STATUS, LOW);
71
72     // Comunicación
73     recepcion_I2C_disponible=0;
74     watchdog_comunicacion_I2C(0);
75
76     // Control
77     setpoint_Fan=0;               setpoint_Water=0;               setpoint_Spray=0;               setpoint_Heating=0;
78
79     // ACTUALIZACIÓN VALORES de actuación
80     noInterrupts();                // Volcado de las variables utilizadas
81     for (i=0;i<3;i++){           // para cálculos sobre las que se
82         pines_ordenados[i]=pines_ordenados_tmp[i];          // emplean para actuar sobre la placa
83         v_estado_pines_durante_activacion[i]=1;            // Se dejan desactivados
84         v_estado_pines_durante_desactivacion[i]=1;          // Se dejan desactivados
85         v_times[i]=0;
86     }
87     interrupts();
88
89 #if defined(MODO_DEPURACION)
90     mostrar_mensaje_almacenado_en_ROM(mensaje_menu_de_seleccion);
91 #endif
92 }
93

```

calculo_valores.ino

```

1  /*
2   * Este fichero forma parte del "SOFTWARE PARA PLACA CONTROLADORA DE MOTORES",
3   * desarrollado por Francisco Luque Luque para el trabajo fin de grado "DISEÑO
4   * E IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL PARA UNA TORRE DE REFRIGERACIÓN",
5   * del Grado en Ingeniería Electrónica Industrial de la Universidad de Córdoba (2017)
6   *
7   * En este fichero se encuentran las funciones relacionadas con los cálculos
8   * de los valores de disparo.
9   */
10
11
12 // -----
13 //          CÁLCULO DEL TIEMPO DE DISPARO
14 // -----
15 unsigned int correccion_paso_cero_t_disparo(unsigned int t_disparo){
16 /*
17     * Realiza la corrección de t_disparo en función de las mediciones de osciloscopio
18     * tomadas sobre el pulso de paso por cero.
19     */
20 if(t_disparo<=TIEMPO_CORRECCION_PASO_POR_CERO) t_disparo=0;
21 else if((t_disparo+TIEMPO_CORRECCION_PASO_POR_CERO)>=10000) t_disparo=10000;
22 else t_disparo=t_disparo-TIEMPO_CORRECCION_PASO_POR_CERO;
23 return t_disparo;
24 }
25
26
27 // -----
28 //          CÁLCULO VALORES DE ACTUACIÓN
29 // -----
30 void calculo_valores_actuacion(byte motor_a_recalcular, double nueva_potencia_deseada) {
31 /*
32     * Ha de ser llamada cada vez que se modifica alguna de las potencias.
33     *
34     * Respecto a las señales de activación de los motores, se aplica lógica inversa
35     * 0: Motor activo
36     * 1: Motor desactivo
37     */
38
39     boolean flag;
40     byte i;
41
42     byte pines_ordenados_tmp[3] = {PIN_F, PIN_W, PIN_S};
43     byte v_estado_pines_durante_activacion_tmp[3] = {0, 0, 0};           // Vectores para la activación y desactivación
44     byte v_estado_pines_durante_desactivacion_tmp[3] = {1, 1, 1};        // 0-0: funcionando al 100%
45                                         // 0-1: potencia está siendo controlada
46                                         // 1-1: está parado (0%)
47     unsigned int tmp;
48
49     unsigned int v_times_rel[3];      // Vector de tiempos relativos al tiempo anterior
50     unsigned int v_times_tmp[3];      // Vector de tiempos para las operaciones

```

```

51
52     boolean desactivacion_de_motor_tmp[3];
53
54     // Comprobación de límites de potencia
55     if (nueva_potencia_deseada > 100.0) nueva_potencia_deseada = 100.0;
56     if (nueva_potencia_deseada < 0.0)    nueva_potencia_deseada = 0.0;
57
58     //
59     // Identificación de motor y recálculo de su tiempo de disparo
60
61     switch ((int) motor_a_recalcular) {    // Se calcula el retraso en el disparo
62         case 'f':;                         // de la variable que se ha cambiado
63         case 'F':;
64             Fan_powerPercentage = nueva_potencia_deseada;
65             tiempo_disparo_Fan=(100*(100-nueva_potencia_deseada));
66             tiempo_disparo_Fan=correccion_paso_cero_t_disparo(tiempo_disparo_Fan);
67             break;
68
69         case 'w':;
70         case 'W':;
71             if (Water_powerPercentage_blocked == 1) Water_powerPercentage = 0;
72             else Water_powerPercentage = nueva_potencia_deseada;
73             tiempo_disparo_Water=(100*(100-nueva_potencia_deseada));
74             tiempo_disparo_Water=correccion_paso_cero_t_disparo(tiempo_disparo_Water);
75             break;
76
77         case 's':;
78         case 'S':;
79             if (Spray_powerPercentage_blocked == 1) Spray_powerPercentage = 0;
80             else Spray_powerPercentage = nueva_potencia_deseada;
81             if(nueva_potencia_deseada!=0) {        // A la espera de incorporar un
82                 nueva_potencia_deseada=100;        // caudalímetro con mayor
83                 tiempo_disparo_Spray=0;          // resolución, se implementa
84             }                                    // función de encendido/apagado.
85             else {
86                 nueva_potencia_deseada=0;
87                 tiempo_disparo_Spray=10000;
88             }
89             break;
90
91     default:
92         // No se ha introducido un parámetro de motor correcto en la llamada.
93         // No se calcula nada y se retorna al punto de llamada.
94         return;
95     }
96
97     //
98     // Ordenación de los pines según su tiempo de disparo
99     // (mayor potencia == menor retraso == disparo antes)
100

```

```

101 // Inicialización, se colocan en el mismo orden que pines_ordenados_tmp:
102 v_times_tmp[0] = tiempo_disparo_Fan;           v_times_tmp[1] = tiempo_disparo_Water;
103 v_times_tmp[2] = tiempo_disparo_Spray;         flag = 1;
104
105 while (flag) {                                // Ordenación
106     flag = 0;
107     for (i = 0; i < 2; i++) {
108         if (v_times_tmp[i] > v_times_tmp[i + 1]) {          // A menor v_times_tmp antes
109             flag = 1;                                         // ha de ser disparado.
110             tmp =(unsigned int) pines_ordenados_tmp[i];        // Se ordenan los pines
111             pines_ordenados_tmp[i] = pines_ordenados_tmp[i+1]; // sobre los que se actuará
112             pines_ordenados_tmp[i+1] = (byte) tmp;              // Y se ordenan los tiempos
113             tmp = v_times_tmp[i];                            // de disparo
114             v_times_tmp[i]=v_times_tmp[i+1];
115             v_times_tmp[i+1]=tmp;
116         }
117     }
118 }
119
120 // -----
121 // Activación/desactivación permanente de la señal de control
122
123 for (i = 0; i < 3; i++) {
124     if (v_times_tmp[i] == 0) {                         // Activación permanente
125         v_estado_pines_durante_activacion_tmp[i] = 0;    // del motor
126         v_estado_pines_durante_desactivacion_tmp[i] = 0;
127         v_times_tmp[i] = 0;
128     }
129     if (v_times_tmp[i]+TIEMPO_CORRECCION_PASO POR_CERO >= 9050) { // Desactivación permanente
130         v_estado_pines_durante_activacion_tmp[i] = 1;       // del motor
131         v_estado_pines_durante_desactivacion_tmp[i] = 1;
132         v_times_tmp[i] = (10000-TIEMPO_CORRECCION_PASO POR_CERO);
133     }
134 }
135
136 // -----
137 // Cálculos para las temporizaciones relativas
138
139 v_times_rel[0] = v_times_tmp[0];
140 for(i=1;i<3;i++) v_times_rel[i]=v_times_tmp[i]-v_times_tmp[i-1];
141
142 for (i=0; i<3; i++) {
143     if ( (v_times_rel[i] == 0) || // 
144         (v_times_tmp[i] == 0) || //
145         (v_times_tmp[i] >= (10000-TIEMPO_CORRECCION_PASO POR_CERO)))
146         v_times_rel[i] = 5; // Para evitar que sea 0 la temporización
147 }
148
149 // -----
150 // Cálculos para las desactivaciones

```

```

151 /*
152 * Si la temporización relativa es menor al tiempo que han de estar los pines activos,
153 * entonces no se desactiva el pin hasta que el siguiente ha de desactivarse.
154 * Por el contrario, si es mayor, se temporiza el tiempo de desactivación (y se desactiva
155 * el pin) y éste es restado de la temporización relativa.
156 */
157 for (i=0; i<2; i++) desactivacion_de_motor_tmp[i]=0; // Inicialización
158
159 for (i=1; i<3; i++) {
160     if(v_times_rel[i]>TIEMPO_MANTENER_ACTIVO_PINES){
161         desactivacion_de_motor_tmp[i-1]=1;
162         for (byte j=i; j<3; j++) {
163             if(v_times_rel[j]>TIEMPO_MANTENER_ACTIVO_PINES)
164                 v_times_rel[j]=(v_times_rel[j]-TIEMPO_MANTENER_ACTIVO_PINES);
165             else
166                 v_times_rel[j]=v_times_rel[j];
167         }
168     }
169     else desactivacion_de_motor_tmp[i-1]=0;
170 }
171
172 desactivacion_de_motor_tmp[2]=1;
173
174 // _____
175 // Actualización de los valores de actuación
176
177 noInterrupts();
178 for (i = 0; i < 3; i++) {
179     pines_ordenados Esperando_interrupcion[i] = pines_ordenados_tmp[i];
180     v_estado_pines_durante_activacion Esperando_interrupcion[i] = v_estado_pines_durante_activacion_tmp[i];
181     v_estado_pines_durante_desactivacion Esperando_interrupcion[i] = v_estado_pines_durante_desactivacion_tmp[i];
182     v_times Esperando_interrupcion[i] = v_times_rel[i];
183     desactivacion_de_motor Esperando_interrupcion[i]=desactivacion_de_motor_tmp[i];
184 }
185 hay_valores Esperando_interrupcion=1;
186 interrupts();
187
188 #if defined(MODO_DEPURACION)
189     hay_valores_para_mostrar=1;
190 #endif
191
192 }
193

```

comunicacion_I2C.ino

```
1  /*
2   * Este fichero forma parte del "SOFTWARE PARA PLACA CONTROLADORA DE MOTORES",
3   * desarrollado por Francisco Luque Luque para el trabajo fin de grado "DISEÑO
4   * E IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL PARA UNA TORRE DE REFRIGERACIÓN",
5   * del Grado en Ingeniería Electrónica Industrial de la Universidad de Córdoba (2017)
6   *
7   * En este fichero se encuentran las funciones relacionadas la comunicación I2C.
8   */
9
10
11 /*
12  * Funciones relacionadas con la comunicación I2C.
13  *
14  * Si se requiere cambiar parámetros automáticos y manuales, es altamente
15  * recomendable que primero se envíe la trama de control para el cambio de
16  * parámetros automáticos, y después otra trama de control con los cambios
17  * manuales.
18 */
19
20 // -----
21 //          INICIALIZACIÓN DE LA COMUNICACIÓN I2C
22 // -----
23
24 void inicializar_comunicacion_I2C(void){
25     Wire.begin(SLAVE_CONTROL_ADDRESS);      // Se introduce como exclavo en el bus I2C
26     Wire.onReceive(recepcion_I2C);
27     Wire.onRequest(peticion_envio_I2C);
28
29     watchdog_comunicacion_I2C(0);         // Inicialización watchdog de comunicación I2C
30 }
31
32 // -----
33 //          I2C_ANYTHING
34 // -----
35 /* Funciones que gestionan la recepción I2C de datos que no son de tipo byte. */
36 // Written by Nick Gammon, May 2012
37 // Comienzo del código escrito por Nick Gammon:
38 #include <Arduino.h>
39
40 template <typename T> unsigned int I2C_writeAnything (const T& value)
41 {
42     const byte * p = (const byte*) &value;
43     unsigned int i;
44     for (i = 0; i < sizeof value; i++)
45         Wire.write(*p++);
46     return i;
47 } // end of I2C_writeAnything
48
49 template <typename T> unsigned int I2C_readAnything(T& value)
50 {
```

```

51     byte * p = (byte*) &value;
52     unsigned int i;
53     for (i = 0; i < sizeof value; i++)
54         *p++ = Wire.read();
55     return i;
56 } // end of I2C_readAnything
57
58 // Final del código escrito por Nick Gammon.
59
60 // -----
61 //           ENVÍO I2C
62 // -----
63
64 void peticion_envio_I2C (void) {
65 /*
66  * Interrupción asociada a la petición I2C de envío de datos.
67 */
68
69 contador_de_envios_I2C++;
70
71 // Preparación del envío
72 byte control_data=0;
73 bitWrite(control_data,7,PID_Water.GetMode());
74 bitWrite(control_data,6,PID_Spray.GetMode());
75 bitWrite(control_data,5,PID_Heating.GetMode());
76 bitWrite(control_data,4,PID_Fan.GetMode());
77 bitWrite(control_data,3,water_level_sensor);
78 bitWrite(control_data,2,spray_level_bottom_sensor);
79
80 if(contador_de_envios_I2C<5) {      // Se envían potencias
81     bitWrite(control_data,0,0);
82
83     // Escritura en el bus
84     Wire.write(control_data);
85     I2C_writeAnything((float) Heating_powerPercentage);
86     I2C_writeAnything((float) Fan_powerPercentage);
87     I2C_writeAnything((float) Spray_powerPercentage);
88     I2C_writeAnything((float) Water_powerPercentage);
89 }
90 else {                                // Se envían consignas de los PIDs
91     contador_de_envios_I2C=0;
92     bitWrite(control_data,0,1);
93
94     // Escritura en el bus
95     Wire.write(control_data);
96     I2C_writeAnything((float) setpoint_Heating);
97     I2C_writeAnything((float) setpoint_Fan);
98     I2C_writeAnything((float) setpoint_Spray);
99     I2C_writeAnything((float) setpoint_Water);
100}

```

```

101 }
102
103     watchdog_comunicacion_I2C(0);
104 }
105
106 // -----
107 // RECEPCIÓN I2C
108 // -----
109
110 void recepcion_I2C (int numero_bytes_recibidos) {
111 /*
112     * Interrupción que gestiona la recepción I2C. No se realiza ninguna
113     * comprobación de errores, sólo que la trama recibida es de 17 bytes. Si
114     * no es de 17 bytes, se ignora la recepción.
115 */
116 byte i;
117 if (((unsigned int) numero_bytes_recibidos) >= (
118             sizeof cabecera_trama +
119             sizeof Fan_speed +
120             sizeof Reynolds_S +
121             sizeof ApproachHR +
122             sizeof Reynolds_W
123         )) {
124     cabecera_trama=Wire.read();
125     for (i=0;i<4;i++) I2C_readAnything(v_data_trama[i]);
126
127     recepcion_I2C_disponible=1;
128
129     watchdog_comunicacion_I2C(0);
130 }
131
132 // -----
133 // RECEPCIÓN I2C
134 // -----
135
136 void tratamiento_recepcion_I2C(byte cabecera, volatile float v_datos[4]){
137 /*
138     * Función llamada cuando se tienen datos disponibles. Se encarga de analizar si es
139     * trama de datos o de control y ejecutar las acciones oportunas.
140 */
141
142 // -----
143 // TRAMA DE DATOS
144
145     if (bitRead(cabecera, 7) == 1) {
146         Fan_speed = (double) v_datos[0];
147         Reynolds_S = (double) v_datos[1];
148         ApproachHR = (double) v_datos[2];
149         Reynolds_W = (double) v_datos[3];
150
151         ApproachHR_EMA = (1 - alpha_EMA) * ApproachHR_EMA + alpha_EMA * ApproachHR; // filtrado de approach

```

```

151     }
152
153 // -----
154 //          TRAMA DE CONTROL
155
156     else {
157
158         // CONTROLES AUTOMÁTICOS
159         if(bitRead(cabecera,6)==1) {
160             if (bitRead(cabecera,3)==1)           // [1000]: Fan
161                 setpoint_Fan= (double) v_datos[0];
162                 PID_Fan.SetMode(AUTOMATIC);
163             }
164             else if (bitRead(cabecera,2)==1)           // [0100]: Spray
165                 setpoint_Spray= (double) v_datos[1];
166                 if (Spray_powerPercentage_blocked==1) Spray_previousPowerPercentage=255;
167                 else PID_Spray.SetMode(AUTOMATIC);
168             }
169             else if (bitRead(cabecera,1)==1)           // [0010]: Heating
170                 setpoint_Heating= (double) v_datos[2];
171                 PID_Heating.SetMode(AUTOMATIC);
172             }
173             else if (bitRead(cabecera,0)==1)           // [0001]: Water
174                 setpoint_Water= (double) v_datos[3];
175                 if (Water_powerPercentage_blocked==1) Water_previousPowerPercentage=255;
176                 else PID_Water.SetMode(AUTOMATIC);
177             }
178         }
179
180         // CONTROLES MANUALES
181     else {
182         if (bitRead(cabecera,3)==1)           // [1000]: Fan
183             PID_Fan.SetMode(MANUAL);
184             calculo_valores_actuacion('f', (double) v_datos[0]);
185         }
186         else if (bitRead(cabecera,2)==1)           // [0100]: Spray
187             if (Spray_powerPercentage_blocked==1) Spray_previousPowerPercentage=(double)v_datos[1];
188             else {
189                 PID_Spray.SetMode(MANUAL);
190                 calculo_valores_actuacion('s', (double) v_datos[1]);
191             }
192         }
193         else if (bitRead(cabecera,1)==1)           // [0010]: Heating
194             potencia_Heating_manual(v_datos[2]);
195         }
196         else if (bitRead(cabecera,0)==1)           // [0001]: Water
197             if (Water_powerPercentage_blocked==1) Water_previousPowerPercentage=(double)v_datos[3];
198             else {
199                 PID_Water.SetMode(MANUAL);

```

```

201         calculo_valores_actuacion('w', (double) v_datos[3]);
202     }
203 }
204 }
205 }
206 }
207
208 // -----
209 //           WATCHDOG PARA COMUNICACIÓN I2C
210 // -----
211 boolean watchdog_comunicacion_I2C(unsigned long tiempo_watchdog) {
212 /*
213 * Si transcurrido el tiempo establecido por el parámetro de entrada
214 * "tiempo_watchdog" no se ha recibido comunicación I2C, devuele 1 (que
215 * indica situación anómala).
216 * Si devuelve un 0 es que aún no ha transcurrido el tiempo de watchdog (que
217 * indica situación normal).
218 *
219 * A esta función hay que llamarla en loop con el parámetro de entrada que
220 * indique el tiempo de watchdog.
221 *
222 * Si se introduce un cero como parámetro de entrada, lo que se hace es
223 * resetear el watchdog. Esto ha de hacerse cada vez que se haya una
224 * comunicación por I2C.
225 */
226 unsigned long tiempo_actual;
227
228 if (tiempo_watchdog==0){
229     tiempo_inicial_watchdog_repcion_I2C=millis();
230     return 0;
231 }
232 tiempo_actual=millis();
233 if (tiempo_actual<tiempo_inicial_watchdog_repcion_I2C){ // Seguridad, en caso de
234     tiempo_inicial_watchdog_repcion_I2C=tiempo_actual;    // overflow de millis.
235     return 0;
236 }
237 if((tiempo_actual-tiempo_inicial_watchdog_repcion_I2C)>tiempo_watchdog){
238     return 1;
239 }
240 else return 0;
241 }
242
243

```

control_PID.ino

```
1  /*
2   * Este fichero forma parte del "SOFTWARE PARA PLACA CONTROLADORA DE MOTORES",
3   * desarrollado por Francisco Luque Luque para el trabajo fin de grado "DISEÑO
4   * E IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL PARA UNA TORRE DE REFRIGERACIÓN",
5   * del Grado en Ingeniería Electrónica Industrial de la Universidad de Córdoba (2017)
6   *
7   * En este fichero se encuentran las funciones relacionadas con el control PID.
8   */
9
10 // -----
11 //          FAN
12 // -----
13 void Setup_Fan_PID(void) {
14     PID_Fan.SetSampleTime(TIEMPO_MUESTRO_PIDS);
15     PID_Fan.SetOutputLimits(20, 70);    // Intervalo de comportamiento lineal
16     PID_Fan.SetMode(MANUAL);
17 }
18 // -----
19 //          WATER PUMP
20 // -----
21 void Setup_Water_PID(void) {
22     PID_Water.SetSampleTime(TIEMPO_MUESTRO_PIDS);
23     PID_Water.SetOutputLimits(30, 65); // Intervalo de comportamiento lineal
24     PID_Water.SetMode(MANUAL);
25 }
26 // -----
27 //          HEATING
28 // -----
29 /*
30  * Discretización de la salida del PID para calentamiento.
31 */
32 void Setup_Heating() {
33     HeatingWindow_inicial = millis();
34     PID_Fan.SetSampleTime(TIEMPO_MUESTRO_PIDS);
35     PID_Heating.SetOutputLimits(0, HeatingWindow);
36     PID_Heating.SetMode(MANUAL);
37 }
38
39 void Heating_action(int WindowSize, double Output, unsigned long &windowStartTime) {
40 /*
41  * Enciende y apaga una salida digital en función de la salida del PID
42 */
43     if (millis() - windowStartTime >= (unsigned long)WindowSize) { // Desplazar la ventana
44         windowStartTime += WindowSize;                                // de activación
45     }
46     if (Output < millis() - windowStartTime) {
47         digitalWrite(PIN_H, LOW);
48         digitalWrite(PIN_HEATING_STATUS, LOW);
49     }
50 else {
```

```

51     if ((Reynolds_W>100.0) && (Reynolds_S>0.0) &&           // Medidas de seguridad para activar
52         (Fan_speed>0.0) && (ApproachHR_EMA>0.0)) {           // el calentamiento.
53         digitalWrite(PIN_H, HIGH);
54         digitalWrite(PIN_HEATING_STATUS, HIGH);
55     }
56     else {
57         digitalWrite(PIN_H, LOW);
58         digitalWrite(PIN_HEATING_STATUS, LOW);
59     }
60 }
61 Heating_powerPercentage=output_Heating*100/HeatingWindow;
62 }
63
64 // -----
65 //          POTENCIA DE HEATING MANUAL
66 // -----
67 void potencia_Heating_manual(double nueva_potencia_deseada) {
68
69     // Comprobación de límites de potencia
70     if (nueva_potencia_deseada>100.0) nueva_potencia_deseada=100.0;
71     if (nueva_potencia_deseada<0.0)   nueva_potencia_deseada=0.0;
72
73     PID_Heating.SetMode(MANUAL);
74
75     // Recálculo de la salida de heating
76     output_Heating=nueva_potencia_deseada*HeatingWindow/100;
77     Heating_powerPercentage=output_Heating*100/HeatingWindow;
78 }
79
80 // -----
81 //          SPRAY PUMP
82 // -----
83 void Setup_Spray_PID(void) {
84     PID_Spray.SetSampleTime(TIEMPO_MUESTRO_PIDS);
85     PID_Spray.SetOutputLimits(0, 100);
86     PID_Spray.SetMode(MANUAL);
87 }
88
89

```

interrupciones.ino

```

1  /*
2   * Este fichero forma parte del "SOFTWARE PARA PLACA CONTROLADORA DE MOTORES",
3   * desarrollado por Francisco Luque Luque para el trabajo fin de grado "DISEÑO
4   * E IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL PARA UNA TORRE DE REFRIGERACIÓN",
5   * del Grado en Ingeniería Electrónica Industrial de la Universidad de Córdoba (2017)
6   *
7   * En este fichero se encuentran las rutinas de atención a interrupciones.
8   */
9  // -----
10 //           Interrupción externa de PASO POR CERO
11 // -----
12 void servicio_int_ext() {
13
14     if (hay_valores Esperando_interrupcion==1) {
15         temporizar_microsegundos(v_times Esperando_interrupcion[0]); // para minimizar el impacto de volcado de nuevos valores
16         hay_valores Esperando_interrupcion=0;
17         for (byte i = 0; i < 3; i++) {
18             pines_ordenados[i] = pines_ordenados Esperando_interrupcion[i];
19             v_estado_pines_durante_activacion[i] = v_estado_pines_durante_activacion Esperando_interrupcion[i];
20             v_estado_pines_durante_desactivacion[i] = v_estado_pines_durante_desactivacion Esperando_interrupcion[i];
21             v_times[i] = v_times Esperando_interrupcion[i];
22             desactivacion_de_motor[i]=desactivacion_de_motor Esperando_interrupcion[i];
23         }
24     }
25     else temporizar_microsegundos(v_times[0]);
26
27     contador_activaciones=0;
28     contador_desactivaciones=0;
29 }
30
31 // -----
32 //           Interrupción temporizador 1
33 // -----
34 ISR (TIMER1_OVF_vect) {
35
36     // Activación de los pines de control
37     if (contador_activaciones<3)
38         digitalWrite(pines_ordenados[contador_activaciones], v_estado_pines_durante_activacion[contador_activaciones]);
39
40     // Desactivación de los pines de control
41     if ((desactivacion_de_motor[contador_activaciones]==1) && (hay_que_desactivar==0)){
42         hay_que_desactivar=1;
43         temporizar_microsegundos(TIEMPO_MANTENER_ACTIVO_PINES);
44     }
45
46     else if (hay_que_desactivar==1) {
47         while (contador_desactivaciones<=contador_activaciones){
48             digitalWrite(pines_ordenados[contador_desactivaciones],
49             v_estado_pines_durante_desactivacion[contador_desactivaciones]);

```

```
50     contador_desactivaciones++;
51 }
52 if(contador_activaciones<2) temporizar_microsegundos(v_times[++contador_activaciones]);
53 else parar_temporizador();
54 }
55
56 else if(contador_activaciones<2) temporizar_microsegundos(v_times[++contador_activaciones]);
57 else parar_temporizador();
58
59 }
60
61 }
```

puerto_serie.ino

```

1  /*
2   * Este fichero forma parte del "SOFTWARE PARA PLACA CONTROLADORA DE MOTORES",
3   * desarrollado por Francisco Luque Luque para el trabajo fin de grado "DISEÑO
4   * E IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL PARA UNA TORRE DE REFRIGERACIÓN",
5   * del Grado en Ingeniería Electrónica Industrial de la Universidad de Córdoba (2017)
6   *
7   * En este fichero se encuentran las funciones relacionadas con el Puerto Serie.
8   */
9
10 #if defined(MODO_DEPURACION)
11
12 // -----
13 // MOSTRAR MENSAJE ALMACENADO EN ROM
14 // -----
15 void mostrar_mensaje_almacenado_en_ROM(const& mensaje){
16     int i;
17     int longitud_cadena;
18     char caracter;
19
20     longitud_cadena=strlen_P(mensaje);
21     for(i=0;i<longitud_cadena;i++) {
22         caracter=pgm_read_byte_near(mensaje + i);
23         Serial.print(caracter);
24     }
25 }
26
27 // -----
28 // INICIALIZAR COMUNICACIÓN POR PUERTO SERIE
29 // -----
30 void inicializar_comunicacion_Puerto_Serie(void){
31     Serial.begin(VEL_COM);
32
33     mostrar_valores(); // Se muestran los valores actuales de los parámetros.
34     mostrar_mensaje_almacenado_en_ROM(mensaje_menu_de_seleccion); // Se muestra un menú de selección para cambiarlos.
35     Serial.println(""); // Se muestra un ejemplo de introducción de parámetros.
36     mostrar_mensaje_almacenado_en_ROM(example_message);
37 }
38
39 // -----
40 // TRATAMIENTO RECEPCIÓN POR PUERTO SERIE
41 // -----
42 void serialEvent(){
43 /*
44     * Interpreta la recepción obtenida por el puerto serie. Esta recepción es de
45     * longitud variable.
46 */
47
48     byte c_in;
49     byte automatic_mode;
50

```

```

51 // -----
52 // RECEPCIÓN DEL MODO AUTOMÁTICO, MANUAL O RESET
53
54 c_in=Serial.read();           // c_in contiene el modo de trabajo: automático/manual O RESET
55 switch (c_in) {
56     case 's': case 'S':          // Mostrar los valores actuales
57         mostrar_valores();
58         return;
59
60     case 'a':;                  // Modo automático
61     case 'A':
62         automatic_mode=1;
63         break;
64
65     case 'm':;                  // Modo manual
66     case 'M':;
67         automatic_mode=0;
68         break;
69
70     case 'r':;                  // RESET de todo el control.
71     case 'R':
72         Serial.println("\t Control variable: R");
73         while(Serial.available()>0) Serial.read(); // se vacia el puerto serie
74         reset_controles();
75         Serial.println("Se ha realizado un reset software por recepcion de R en Puerto Serie.");
76         return;
77
78     case 'k':;      // Función oculta del menú (una vez sintonizados no se volverán a cambiar).
79     case 'K':      // Sirve para la sintonización de PIDs desde Puerto Serie.
80
81     if(!espera_recep_PSerie(VEL_COM, TIEMPO_WATCHDOG_PS)) return;
82     else {           // Recibido dato
83         c_in=Serial.read();
84         switch(c_in){
85             case 'f': case 'F':
86                 Serial.println("");
87                 Serial.println("PID de Fan:");
88                 cambio_parametros_PID(Kp_Fan,Ki_Fan,Kd_Fan);
89                 PID_Fan.SetTunings(Kp_Fan,Ki_Fan,Kd_Fan);
90                 mostrar_parametros_PID(Kp_Fan,Ki_Fan,Kd_Fan);
91                 break;
92             case 'w': case 'W':
93                 Serial.println("");
94                 Serial.println("PID de Water:");
95                 cambio_parametros_PID(Kp_Water,Ki_Water,Kd_Water);
96                 PID_Water.SetTunings(Kp_Water,Ki_Water,Kd_Water);
97                 mostrar_parametros_PID(Kp_Water,Ki_Water,Kd_Water);
98                 break;
99             case 's': case 'S':
100                Serial.println("");

```

```

101
102     Serial.println("PID de Spray:");
103     cambio_parametros_PID(Kp_Spray,Ki_Spray,Kd_Spray);
104     PID_Spray.SetTunings(Kp_Spray,Ki_Spray,Kd_Spray);
105     mostrar_parametros_PID(Kp_Spray,Ki_Spray,Kd_Spray);
106     break;
107 }
108 return;
109
110 default: // Si el valor introducido no es válido
111     mostrar_mensaje_almacenado_en_ROM(mensaje_de_error);
112     while(Serial.available()>0) Serial.read();
113     mostrar_mensaje_almacenado_en_ROM(mensaje_menu_de_seleccion);
114     return;
115 }
116
117 //----- RECEPCIÓN DEL MOTOR OBJETIVO Y DE SU PARÁMETRO
118
119 if(espera_recep_P Serie(VEL_COM, TIEMPO_WATCHDOG_PS)==0) return;
120 c_in=Serial.read(); // c_in contiene el parámetro objetivo (F, W, S o H)
121 // de la variable que se ha cambiado
122
123 if(espera_recep_P Serie(VEL_COM, TIEMPO_WATCHDOG_PS)==1) {
124     switch (c_in) {
125         case 'f':;
126         case 'F':;
127             Serial.print("\t Control variable: Fan. Value: ");
128             if (automatic_mode==1){
129                 setpoint_Fan=Serial.parseFloat();
130                 Serial.println(setpoint_Fan);
131                 PID_Fan.SetMode(AUTOMATIC);
132             }
133             else {
134                 c_in=recepcion_3_octetos_y_reconstruccion_byte(VEL_COM,0);
135                 Serial.print(c_in); Serial.println(" %");
136                 PID_Fan.SetMode(MANUAL);
137                 calculo_valores_actuacion('f',(double) c_in);
138             }
139         break;
140
141         case 'w':;
142         case 'W':;
143             Serial.print("\t Control variable: Water. Value: ");
144             if (automatic_mode==1){
145                 espera_recep_P Serie(VEL_COM, TIEMPO_WATCHDOG_PS);
146                 setpoint_Water=Serial.parseFloat();
147                 Serial.println(setpoint_Water);
148                 if (Water_powerPercentage_blocked==1) Water_previousPowerPercentage=255;
149                 else PID_Water.SetMode(AUTOMATIC);
150             }

```

```

151 }
152 else {
153     c_in=recepcion_3_octetos_y_reconstruccion_byte(VEL_COM,0);
154     Serial.print(c_in); Serial.println(" %");
155     if (Water_powerPercentage_blocked==1) Water_previousPowerPercentage=(double)c_in;
156     else {
157         PID_Water.SetMode(MANUAL);
158         calculo_valores_actuacion('w', (double) c_in);
159     }
160 }
161 break;

163 case 's':;
164 case 'S':;
165     Serial.print("\t Control variable: Spray. Value: ");
166     if (automatic_mode==1){
167         espera_recep_P Serie(VEL_COM, TIEMPO_WATCHDOG_PS);
168         setpoint_Spray=Serial.parseFloat();
169         Serial.println(setpoint_Spray);
170         if (Spray_powerPercentage_blocked==1) Spray_previousPowerPercentage=255;
171         else PID_Spray.SetMode(AUTOMATIC);
172     }
173     else {
174         c_in=recepcion_3_octetos_y_reconstruccion_byte(VEL_COM,0);
175         Serial.print(c_in); Serial.println(" %");
176         if (Spray_powerPercentage_blocked==1) Spray_previousPowerPercentage=(double)c_in;
177         else {
178             PID_Spray.SetMode(MANUAL);
179             calculo_valores_actuacion('s', (double) c_in);
180         }
181     }
182 break;

184 case 'h':;
185 case 'H':;
186     Serial.print("\t Control variable: Heating. Value: ");
187     if (automatic_mode==1){
188         espera_recep_P Serie(VEL_COM, TIEMPO_WATCHDOG_PS);
189         setpoint_Heating=Serial.parseFloat();
190         Serial.println(setpoint_Heating);
191         PID_Heating.SetMode(AUTOMATIC);
192     }
193     else {
194         c_in=recepcion_3_octetos_y_reconstruccion_byte(VEL_COM,0);
195         Serial.print(c_in); Serial.println(" %");
196         potencia_Heating_manual((double)c_in);
197     }
198 break;

199 default: // Si el valor introducido no es válido

```

```

201     mostrar_mensaje_almacenado_en_ROM(mensaje_de_error);
202     while(Serial.available()>0) Serial.read();
203     mostrar_mensaje_almacenado_en_ROM(mensaje_menu_de_seleccion);
204     return;
205   }
206   hay_valores_para_mostrar=1;
207 }
208
209
210 // -----
211 // FORMATEADO DE VALORES PARA LA INTERFAZ DE USUARIO
212 // -----
213
214 void formateado_valores(byte modo, double potencia, double consigna, double medicion) {
215 /*
216  * Función complementaria a mostrar_valores. Se encarga de formatear los valores
217  * de cada motor de la misma forma.
218 */
219 if (modo==1) { Serial.print(" Automatic"); Serial.print("\t Setpoint: ");
220   Serial.println(consigna); }
221 else if (modo==2) { Serial.print(" BLOCKED "); Serial.print("\t Setpoint: ");
222   Serial.println(consigna); }
223 else Serial.println("\t Manual");
224 Serial.print("\t\t Pot: "); Serial.print(potencia); Serial.print("%");
225 Serial.print("\t Measure: "); Serial.println(medicion);
226 }
227 // -----
228 // MOSTRAR VALORES POR LA INTERFAZ DE USUARIO
229 // -----
230 void mostrar_valores(void) {
231 /*
232  * Se encarga de mostrar, por el puerto serie, los valores actuales de los
233  * diferentes parámetros de control.
234 */
235
236 byte modo;
237
238 Serial.println("Current values:");
239 Serial.print("\tFan: \t");
240 formateado_valores(PID_Fan.GetMode(), Fan_powerPercentage, setpoint_Fan, Fan_speed);
241
242 Serial.print("\tSpraypump: ");
243 if (Spray_powerPercentage_blocked==1) modo=2;
244 else modo=PID_Spray.GetMode();
245 formateado_valores(modo, Spray_powerPercentage, setpoint_Spray, Reynolds_S);
246
247 Serial.print("\tHeating: ");
248 formateado_valores(PID_Heating.GetMode(), Heating_powerPercentage, setpoint_Heating, ApproachHR_EMA);
249
250 Serial.print("\tWaterpump: ");

```

```

251 if (Water_powerPercentage_blocked==1) modo=2;
252 else modo=PID_Water.GetMode();
253 formateado_valores(modo,Water_powerPercentage, setpoint_Water, Reynolds_W);
254
255 Serial.println("\tLevel sensors:");
256 Serial.print("\t Bottom level Spray sensor: "); Serial.print(spray_level_bottom_sensor);
257 Serial.print("\t\t Bottom level Water sensor: "); Serial.println(water_level_sensor);
258 }
259
260
261 // -----
262 // ESPERA RECEPCIÓN POR PUERTO SERIE CON FUNCIONALIDAD DE WATCHDOG
263 // -----
264 boolean espera_recep_P Serie(unsigned long velocidad_comunicacion, unsigned long tiempo_watchdog) {
265 /* Parámetros de entrada:
266 * - unsigned long: Velocidad de comunicación del Puerto Serie.
267 * - unsigned long: Tiempo para el watchdog de recepción.
268 * Excepción: Si vale 0 la funcionalidad de watchdog queda desactivada, sólo se realiza
269 * una espera en función de la velocidad de comunicación.
270 *
271 * Parámetros de salida:
272 * - boolean:
273 *   . 1: Se ha recibido dato en el tiempo establecido por el watchdog.
274 *   . 0: No se ha recibido dato dentro del plazo límite.
275 *
276 * Detalle de la función:
277 * Esta función realiza una espera que varía (dependiendo de la velocidad de
278 * comunicación), hasta que se ha recibido el siguiente octeto. El cálculo que se
279 * realiza viene dado por:
280 *   bits / (velocidad_comunicacion (bits/segundo)) = número de bits / segundo
281 *   (10 bits * 1000000 useg/seg)/(VEL_COM bits/seg) == tiempo que tarda en llegar
282 *   cada octeto, expresado en microsegundos. Son diez bits en lugar de 8 porque hay
283 *   que sumarle el bit de comienzo y parada a cada trama.
284 *
285 * Es conveniente su llamada si se requiere un watchdog en la recepción (establecer tiempo
286 * máximo de espera).
287 */
288 unsigned int tmp;
289 unsigned long tiempo_inicial;
290 unsigned long tiempo_actual;
291
292 if (Serial.available()>0) return 1;
293
294 tmp=(unsigned int)((10000000)/velocidad_comunicacion);
295 delayMicroseconds(tmp+100);
296
297 if (Serial.available()>0) return 1;
298
299 if (tiempo_watchdog>0){ // Se activa el watchdog del puerto serie.
300   tiempo_inicial=millis();

```

```

301     while(Serial.available()==0){
302         tiempo_actual=millis();
303         if (tiempo_actual<tiempo_inicial) tiempo_inicial=tiempo_actual; // Seguridad, en caso de
304                                         // overflow de millis.
305         else if ((tiempo_actual-tiempo_inicial)>tiempo_watchdog) {
306             Serial.println("ERROR");
307             Serial.println("\t Tiempo de espera agotado.");
308             return 0;
309         }
310     }
311     return 1;
312 }
313 else return 1; // No hay watchdog activo, sólo se temporiza
314                 // el tiempo de recepción por velocidad de comunicación
315 }
316
317
318 // -----
319 // RECEPCIÓN 3 OCTETOS QUE CONFORMAN UN NÚMERO DE TIPO BYTE
320 // -----
321 byte recepcion_3_octetos_y_reconstruccion_byte(unsigned long velocidad_comunicacion,boolean iniciar_comunicacion){
322 /*
323 * PARÁMETROS DE ENTRADA
324 *   velocidad_comunicacion: Velocidad de comunicación del puerto serie.
325 *   iniciar_comunicacion: Para iniciar y finalizar la comunicación con la
326 *                           velocidad especificada (valor=1) o se parte de una
327 *                           comunicación ya iniciada (valor=0).
328 *
329 * PARÁMETROS DE SALIDA
330 *   numero (byte): Número reconstruido.
331 *
332 * Esta función lee del puerto serie un máximo de 3 octetos, a partir de
333 * los que reconstruye un número natural de 0 a 255 (devuelve un octeto).
334 *
335 * Esta función es más rápida que ParseInt de Arduino.
336 */
337 byte datos_recibidos[3];
338 byte numero=0;
339 byte i=0;
340
341 // Inicializaciones
342 for(i=0;i<3;i++) datos_recibidos[i]=0;
343
344 if (iniciar_comunicacion==1) Serial.begin(velocidad_comunicacion);
345
346 // Espera con watchdog de Puerto Serie. Pasado el tiempo de watchdog, se devuelve el número 0.
347 if(espera_recep_Pserie(VEL_COM,TIEMPO_WATCHDOG_PS)==0) return 0;
348
349 // Recepción y almacenamiento
350 i=0;

```

```
351     while((Serial.available()>0) && (i<3) ) { // Se guarda dato a dato en el vector datos_recibidos
352         datos_recibidos[i]=Serial.read()-48;
353         i++;
354         espera_recep_P Serie(velocidad_comunicacion,0); // Sin watchdog, se supone que el usuario ha introducido
355                                         // el número entero antes de pulsar la tecla 'enter'
356     }
357     // Reconstrucción y devolución de valor
358     if (i==1) numero=datos_recibidos[0];
359     else if (i==2) numero=datos_recibidos[0]*10+datos_recibidos[1];
360     else if (i==3) numero=datos_recibidos[0]*100+datos_recibidos[1]*10+datos_recibidos[2];
361     else numero=0; // medida de seguridad
362     if (iniciar_comunicacion==1) Serial.end();
363     return numero;
364 }
365
366 #endif
367
368
369
370
```

sensores.ino

```
1  /*
2   * Este fichero forma parte del "SOFTWARE PARA PLACA CONTROLADORA DE MOTORES",
3   * desarrollado por Francisco Luque Luque para el trabajo fin de grado "DISEÑO
4   * E IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL PARA UNA TORRE DE REFRIGERACIÓN",
5   * del Grado en Ingeniería Electrónica Industrial de la Universidad de Córdoba (2017)
6   *
7   * En este fichero se encuentran las funciones relacionadas con la lectura de
8   * los sensores de nivel y las actuaciones a realizar en función de las mismas.
9   */
10
11
12 // -----
13 //          INICIALIZACIÓN
14 // -----
15 void inicializar_sensores(void) {
16     // AGUA
17     water_level_sensor=digitalRead(PIN_WATER_LEVEL);
18     water_level_sensor_ant=water_level_sensor;
19     histeresis_water_level_activa=0;
20     tiempo_histeresis_water_level_se_activa=0;
21
22     // ROCIADO
23     spray_level_bottom_sensor=digitalRead(PIN_SPRAY_LEVEL_BOTTOM);
24     spray_level_bottom_sensor_ant=spray_level_bottom_sensor;
25     histeresis_spray_level_bottom_activa=0;
26     tiempo_histeresis_spray_level_bottom_se_activa=0;
27 }
28
29
30
31 void lectura_y_proteccion_por_sensores(void) {
32 /*
33 * En esta función se leen los sensores, aplicando la protección mediante
34 * histéresis temporal a las bombas.
35 */
36
37     unsigned long tiempo_actual=millis();
38
39 // -----
40 //          LECTURA NIVEL DE AGUA Y PROTECCIÓN
41 //          DE LA BOMBA DE AGUA MEDIANTE HISTÉRESIS TEMPORAL
42
43     // Lectura bloqueada con anterioridad
44     if (histeresis_water_level_activa==1) {
45         if (tiempo_actual<tiempo_histeresis_water_level_se_activa){ // Protección contra desbordamiento de millis
46             tiempo_histeresis_water_level_se_activa=tiempo_actual;}
47
48         else if ((tiempo_actual-tiempo_histeresis_water_level_se_activa)>HISTERESIS_PROTECCION_BOMBAS){
49             histeresis_water_level_activa=0;} // Se desbloquea la lectura al cumplirse la histéresis
50     }
```

```

51
52 // Lectura no bloqueada, se efectúa lectura y se comprueban condiciones de bloqueo
53 else {
54     water_level_sensor=digitalRead(PIN_WATER_LEVEL);
55     if (water_level_sensor_ant!=water_level_sensor) {
56         histeresis_water_level_activa=1;
57         tiempo_histeresis_water_level_se_activa=tiempo_actual;
58         water_level_sensor_ant=water_level_sensor;
59     }
60     else water_level_sensor_ant=water_level_sensor;
61 }
62
63 //
64 // _____
65 //          BLOQUEO DE BOMBA DE AGUA POR NIVEL BAJO
66 //
67 // Water_previousPowerPercentage: Variable que guarda el estado.
68 //           Si se está en modo manual guarda el valor de Water_powerPercentage.
69 //           Si se está en modo automático se pone a 255.
70
71 if (water_level_sensor==1) {                                // Lectura correcta
72     if(Water_powerPercentage_blocked==1) {
73         Water_powerPercentage_blocked=0;
74         if (Water_previousPowerPercentage==255) PID_Water.SetMode(AUTOMATIC); // Si el PID estaba en
75         else {                                         // modo automático,
76             Water_powerPercentage=Water_previousPowerPercentage;           // toma valor 255
77             calculo_valores_actuacion('w', Water_powerPercentage);
78         }
79     }
80 }
81 else {                                                 // ATENCIÓN, nivel bajo en la lectura
82     if(Water_powerPercentage_blocked==0) {
83         Water_powerPercentage_blocked=1;
84         if(PID_Water.GetMode()==1) Water_previousPowerPercentage=255; // Si el PID está en
85         else Water_previousPowerPercentage=Water_powerPercentage;   // modo automático,
86         PID_Water.SetMode(MANUAL);                                     // toma valor 255
87         Water_powerPercentage=0;
88         calculo_valores_actuacion('w', Water_powerPercentage);
89     }
90 }
91
92 //
93 // _____
94 //          LECTURA NIVEL DE SPRAY_BOTTOM Y PROTECCIÓN
95 //          DE LA BOMBA DE SPRAY MEDIANTE HISTÉRESIS TEMPORAL
96
97 // Lectura bloqueada con anterioridad
98 if (histeresis_spray_level_bottom_activa==1) {
99     if (tiempo_actual<tiempo_histeresis_spray_level_bottom_se_activa) // Protección contra desbordamiento de millis

```

```

100     tiempo_histeresis_spray_level_bottom_se_activa=tiempo_actual;
101
102     else if ((tiempo_actual-tiempo_histeresis_spray_level_bottom_se_activa)>HISTERESIS_PROTECCION_BOMBAS)
103         histeresis_spray_level_bottom_activa=0; // Se desbloquea la lectura al cumplirse la histéresis
104     }
105
106 // Lectura no bloqueada, se efectúa lectura y se comprueban condiciones de bloqueo
107 else {
108     spray_level_bottom_sensor=digitalRead(PIN_SPRAY_LEVEL_BOTTOM);
109     if (spray_level_bottom_sensor_ant!=spray_level_bottom_sensor) {
110         histeresis_spray_level_bottom_activa=1;
111         tiempo_histeresis_spray_level_bottom_se_activa=tiempo_actual;
112         spray_level_bottom_sensor_ant=spray_level_bottom_sensor;
113     }
114     else spray_level_bottom_sensor_ant=spray_level_bottom_sensor;
115 }
116

117 //
118 // _____
119 // BLOQUEO DE BOMBA DE ROCIADO POR NIVEL BAJO
120 //
121 // Spray_previousPowerPercentage: Variable que guarda el estado.
122 // Si se está en modo manual guarda el valor de Spray_powerPercentage.
123 // Si se está en modo automático se pone a 255.
124
125 if (spray_level_bottom_sensor==1) { // Lectura correcta
126     if(Spray_powerPercentage_blocked==1) {
127         Spray_powerPercentage_blocked=0;
128         if (Spray_previousPowerPercentage==255) PID_Spray.SetMode(AUTOMATIC); // Si el PID estaba en
129         else { // modo automático,
130             Spray_powerPercentage=Spray_previousPowerPercentage; // toma valor 255
131             calculo_valores_actuacion('s', Spray_powerPercentage);
132         }
133     }
134 }
135
136 else { // ATENCIÓN, nivel bajo en la lectura
137     if(Spray_powerPercentage_blocked==0) {
138         Spray_powerPercentage_blocked=1;
139         if(PID_Spray.GetMode()==1) Spray_previousPowerPercentage=255; // Si el PID está en
140         else Spray_previousPowerPercentage=Spray_powerPercentage; // modo automático,
141         PID_Spray.SetMode(MANUAL); // toma valor 255
142         Spray_powerPercentage=0;
143         calculo_valores_actuacion('S', Spray_powerPercentage);
144     }
145 }
146

```

temporizador.ino

```

1  /*
2   * Este fichero forma parte del "SOFTWARE PARA PLACA CONTROLADORA DE MOTORES",
3   * desarrollado por Francisco Luque Luque para el trabajo fin de grado "DISEÑO
4   * E IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL PARA UNA TORRE DE REFRIGERACIÓN",
5   * del Grado en Ingeniería Electrónica Industrial de la Universidad de Córdoba (2017)
6   *
7   * En este fichero se encuentran las funciones relacionadas con el temporizador 1.
8   */
9
10 // -----
11 //          LECTURA TEMPORIZADOR 1
12 // -----
13 unsigned int leer_timer1(void){
14 /* Lectura atómica del registro de 16 bits del timer 1 */
15     unsigned char sreg;
16     unsigned int i;
17     sreg=SREG;           // Guarda la bandera de interrupción global
18     cli();              // Desabilita interrupciones
19     i = TCNT1;           // Lee el valor del timer 1
20     SREG = sreg;         // Restaura la bandera de interrupción global
21     return i;            // Se devuelve el valor del timer 1
22 }
23
24 // -----
25 //          ESCRITURA TEMPORIZADOR 1
26 // -----
27 void escribir_timer1(unsigned int i){
28 /* Escritura atómica del registro de 16 bits del timer 1 */
29     unsigned char sreg;
30     sreg=SREG;
31     cli();
32     TCNT1 = i;
33     SREG = sreg;
34 }
35
36 // -----
37 //          CONFIGURAR TEMPORIZADOR 1 CON PREESCALADOR
38 // -----
39 void configurar_modo_normal_t1_con_preescalador_8(void) {
40 /*
41 * Configura la cuenta ascendente normal y habilita interrupción por
42 * desbordamiento (cuando se alcanza el valor máximo de cuenta)
43 */
44     TCCR1B &= 0xF8; // Se para el temporizador 1
45     TCCR1A &= 0xFC; // (WGM11 && WGM10 = 0 &&
46     TCCR1B &= 0xE2; // WGM13 && WGM12 = 0 )-> Normal mode
47     TIMSK1 = 0x01; // Se escribe bit TOIE: Overflow Interrupt Enable
48 }
49
50 // -----

```

```
51 // ARRANQUE Y PARADA DEL TEMPORIZADOR 1
52 // -----
53 void temporizar_con_preeescalador_8(void) { TCCR1B |= 0x02; } // Arranca la cuenta
54
55 void parar_temporizador(void){ TCCR1B&=0xE0; } // Para la cuenta
56
57
58 // -----
59 // REALIZAR TEMPORIZACIÓN DE 'X' MICROSEGUNDOS
60 // -----
61 void temporizar_microsegundos(unsigned int numero_microsegundos){
62     /*Temporiza el número de microsegundos introducidos como parámetro. */
63     unsigned int numero_ciclos;
64     unsigned int numero_a_recargar;
65
66     configurar_modo_normal_t1_con_preeescalador_8();
67
68     if (numero_microsegundos<5) numero_microsegundos=5; // 16MHz -> 1 ciclo=0.0625us;
69     numero_ciclos=numero_microsegundos*2; // preeescalador clk/8-> 1ciclo=0.5us
70     numero_a_recargar=65536-numero_ciclos;
71     escribir_timer1(numero_a_recargar);
72     temporizar_con_preeescalador_8();
73 }
74
75
76
77
```