

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



TRABAJO FIN DE GRADO

**GRADO EN INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN (GITST)**

**Diseño e implementación de un
actuador para el control de la
refrigeración de una sala de servidores**

JUAN CARLOS CALVO SANSEGUNDO

2017

TRABAJO FIN DE GRADO

Título: DISEÑO E IMPLEMENTACIÓN DE UN ACTUADOR
PARA EL CONTROL DE LA REFRIGERACIÓN DE UNA
SALA DE SERVIDORES

Autor: JUAN CARLOS CALVO SANSEGUNDO

Tutor: D. JOSE MANUEL MOYA FERNÁNDEZ

Ponente:

Departamento: Departamento de Ingeniería Electrónica (DIE)

TRIBUNAL:

Presidente:

Vocal:

Secretario:

Suplente:

Fecha de lectura:

Calificación:

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



TRABAJO FIN DE GRADO

**GRADO EN INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN (GITST)**

**Diseño e implementación de un
actuador para el control de la
refrigeración de una sala de servidores**

JUAN CARLOS CALVO SANSEGUNDO

2017

*Dedicado a mis padres,
mi hermano
y mi abuela.*

Agradecimientos

En primer lugar, quiero dar las gracias a todos los compañeros de *GreenLSI* por haber compartido estos meses de trabajo con vosotros y por vuestra ayuda. En especial a mi tutor Jose Manuel, por haberme guiado en estos meses para sacar adelante este trabajo y que pudiera ver la luz.

También me gustaría dar las gracias a todos los compañeros que he conocido en todo este tiempo pero sobretodo a Yaiza y Felipe. Gracias por acompañarme hasta el final y por haberme dado muy buenos momentos. Me llevo un grato recuerdo de vosotros.

En este agradecimiento no podrían faltar mis padres y mi hermano, sin los cuales yo no estaría escribiendo estas palabras. Me habeís apoyado y animado desde el inicio y a pesar de las dificultades, habeís seguido apoyándome y confiando en mi, dándome todo lo necesario para que pudiera alcanzar mi objetivo.

Tampoco podían faltar mis amigas Rebeca, Estela, Mari, Emma, Alba, Ana y Marta. Gracias por estar ahí siempre y por tener tantas ganas de verme terminar la carrera.

No podría terminar este agradecimiento sin darte las gracias a ti, abuela. Este trabajo va dedicado a ti. Por todo el tiempo que esta carrera me ha hecho pasar en tu casa, aguantarme durante los días malos y haberme hecho más ameno esas largas tardes de estudio. Siento que no pueda estar escribiendo estas palabras junto a ti, pero sé que me estarás viendo y estarás muy contenta de que a pesar de todas las dificultades, he conseguido llegar a mi objetivo.

Resumen

Este trabajo fin de grado (TFG) presenta el diseño e implementación de un sistema de actuación con el que se pueda controlar la refrigeración de una sala de servidores o de un centro de datos (CPD).

En este trabajo se pretende dar soporte a un sistema de monitorización y optimización energética de centros de datos y poder aplicar una de sus optimizaciones que está basada en el sistema de refrigeración. El sistema de optimización, mediante un algoritmo, predice la temperatura a la que debe estar una sala de servidores para que el consumo energético de dicha sala sea mínimo. Para lograr este objetivo, es necesario actuar sobre el sistema de refrigeración y regular su temperatura de funcionamiento de forma dinámica. De este modo, se consigue que la sala alcance el valor de temperatura deseado.

Teniendo en cuenta esta necesidad, se ha diseñado e implementado un sistema de actuación basado en un sistema de control en lazo cerrado. Se ha conseguido que el sistema sea capaz de regular la temperatura de la sala, siguiendo el valor de temperatura óptima proporcionada por el algoritmo. Este sistema obtiene los datos de temperatura desde una plataforma web, en tiempo real y sin necesidad de ningún tipo de acción humana. También se ha conseguido que el sistema sea dinámico y capaz de responder a las distintas variaciones que puedan producirse en la temperatura óptima, de una forma estable y con la mayor rapidez posible. El sistema de control utilizado se basa en un controlador PID, que posee una implementación sencilla y proporciona un amplio rango de operación, además de que es ampliamente usado en la industria para el control de procesos, incluyendo la temperatura.

Este sistema de actuación ha sido probado en la sala de servidores B039 del departamento de Ingeniería Electrónica y posteriormente será integrado en el sistema de monitorización y optimización energética desarrollado por el grupo *GreenLSI*, perteneciente a este departamento.

Palabras clave

Centro de procesamiento de datos, sistema de control, optimización energética, control de la temperatura, controlador PID, actuador, sistema ciber-físico, sistema de monitorización.

Abstract

This final project work shows the design and implementation of an actuator to control the cooling system of a server room or data center.

The objective of this work is to apply one type of energy optimization based on the cooling system. An energy monitoring and optimization system predicts what is the optimal temperature of the server room so that the energy consumption is minimum. To apply this optimization, it is necessary to modify dynamically the setpoint of the cooling system to achieve that the temperature of the room reaches the optimal temperature.

For this reason, an actuator based on a closed loop control system has been designed and implemented. This actuator is capable to setting the optimal temperature in the server room or data center. The actuator obtains the temperature data connecting to a web platform, in an automatic way and in real time. Also, the actuator is capable of responding to any type of optimum temperature in a dynamic, stable and fast way. The control system used by the actuator is based on a PID controller, which has a simple implementation and provides a wide range of operations. Besides, this type of controller is commonly used in the industry to process control, including the temperature.

This actuator has been tested in a server room of the Electronic Engineering Department of Universidad Politécnica de Madrid. Later, this actuator will be integrated in the energy monitoring and optimization system developed by this group.

Keywords

Data centers, control system, actuator, energy optimization, temperature control, PID controller, Cyber-Physical System, monitoring system.

Lista de acrónimos

BJT *Bipolar Junction Transistor*

CE *Chip Enable*

CPD Centro de Procesamiento de Datos

CPU Unidad de Procesamiento de Datos - *Central Processor Unit*

DCIM *Data Center Infrastructure Management*

DIE Departamento de Ingeniería Electrónica

DPM *Dynamic Power Management*

DVFS *Dynamic Voltage Frecuency Scalling*

ETSIT Escuela Técnica Superior de Ingenieros de Telecomunicación

GPIO *General Purpose Input/Output*

HTTP Protocolo de Transferencia de Hipertexto - *HyperText Transfer Protocol*

IRLED *Infrared Light-Emitting Diode*

IT *Information Technology*

JSON *JavaScript Object Notation*

KHz KiloHercio - *KiloHertz*

LSI Laboratorio de Sistemas Integrados

MISO *Master Input Slave Output*

X

MOSI *Master Output Slave Input*

NRMSE *Normalized Root Mean Square Error*

PD Proporcional Derivativo - *Proportional-Derivative*

PI Proporcional Integral - *Proportional-Integral*

PID Proporcional Integral Derivativo - *Proportional-Integral-Derivative*

PUE *Power Usage Effectiveness*

SCADA *Supervisory Control and Data Acquisition*

SCLK *Serial Clock*

SPI *Serial Peripheral Interface*

TIC Tecnologías de la Información y Comunicación

TWh Teravatio-hora - *Terawatt hour*

UPM Universidad Politécnica de Madrid

URL Localizador de Recursos Uniforme - *Uniform Resource Locator*

VM Máquina Virtual - *Virtual Machine*

VOVO *Vary-On Vary-Off*

ZB Zettabyte

Índice general

Agradecimientos	III
Resumen	v
Abstract	vii
Lista de acrónimos	ix
Índice de figuras	xv
1. Introducción	1
1.1. Eficiencia energética en los centros de datos	2
1.1.1. Técnicas de eficiencia en los equipos IT	2
1.1.2. Técnicas de eficiencia en la refrigeración	3
1.2. Enfoque ciber-físico	4
1.3. Objetivos y fases del trabajo	5
2. Estado del arte	7
2.1. Sistema de control	7
2.2. Tipos de controladores	8
2.2.1. Controlador de 2 posiciones	9

2.2.2. Controlador proporcional (P)	10
2.2.3. Controlador Integral (I)	11
2.2.4. Controlador Proporcional Integral (PI)	12
2.2.5. Controlador Derivativo (D)	12
2.2.6. Controlador Proporcional Derivativo (PD)	13
2.2.7. Controlador PID	13
2.3. Control de temperatura	15
3. Banco de pruebas	17
4. Diseño	19
4.1. Descripción del sistema completo	19
4.2. Bloque de adquisición de datos	21
4.3. Bloque de comparación	23
4.4. Bloque de control	24
4.4.1. Caracterización de la planta	24
4.4.2. Diseño del controlador PID	26
4.4.3. Discretización del controlador PID	28
4.5. Bloque de generación de comandos	29
4.5.1. Subbloque de selección del comando	30
4.5.2. Subbloque de modulación del comando	31
4.6. Bloque de transmisión	32
4.6.1. Conversor de bits a señal	32
4.6.2. Circuito emisor de infrarrojos	33

4.7. Bloque de envío de datos	34
5. Implementación	35
5.1. Bloque de adquisición	36
5.2. Bloque de comparación	39
5.3. Bloque de control	39
5.4. Bloque de generación de comandos	42
5.4.1. Subbloque de selección del comando	43
5.4.2. Subbloque de modulación del comando	44
5.5. Bloque de transmisión	46
5.5.1. Conversor de bits a señal	46
5.5.2. Circuito emisor de infrarrojos	47
5.6. Bloque de envío de datos	48
5.7. Ciclo de ejecución	48
6. Test y resultados	51
6.1. Resultados de las pruebas	52
6.2. Análisis de los resultados	54
7. Conclusiones y Líneas Futuras	57
7.1. Conclusiones	57
7.2. Lineas Futuras	58
8. Anexos	59
8.1. Anexo 1. Estimación de la función de transferencia	59
8.1.1. Proceso de estimación de la función de transferencia	59

8.1.2. Scripts de matlab para hacer la estimación	65
8.2. Anexo 2: Otros experimentos	66
8.3. Anexo 3: Script del controlador PID	71

Índice de figuras

1.1.	Distribución del consumo en un centro de datos	2
1.2.	Esquema del sistema de optimización. Fuente: [17]	4
2.1.	Sistema de control en lazo abierto. Fuente: <i>Benjamin C. Kuo</i> [19] . . .	8
2.2.	Sistema de control en lazo cerrado. Fuente: <i>Katsuhiko Ogata</i> [18] . . .	8
2.3.	Esquema de funcionamiento del controlador <i>On-Off</i>	9
2.4.	Respuesta de un sistema con un controlador <i>On-Off</i> . Fuente: Omron [22]	10
2.5.	Respuesta de un sistema con un controlador P. Fuente: Omron [22] . .	11
2.6.	Respuesta de un sistema con una acción PI. Fuente: <i>Benjamin C. Kuo</i> [19]	12
2.7.	Respuesta de un sistema con una acción PD. Fuente: <i>Benjamin C. Kuo</i> [19]	13
2.8.	Respuesta de un sistema usando un PID. Fuente: <i>Benjamin C. Kuo</i> [19]	14
2.9.	Esquemas de representación de un PID. Fuente: <i>Katsuhiko Ogata</i> [20]	14
2.10.	Representación del efecto windup. Fuente: <i>Katsuhiko Ogata</i> [20]	15
3.1.	Fotografía de la sala B039	18
4.1.	Diagrama del sistema completo	20
4.2.	Esquema del bloque de adquisición de datos	21

4.3. Interfaz gráfica de <i>Graphite</i>	22
4.4. Diagrama del bloque de comparación	23
4.5. Esquema del bloque de control	24
4.6. Respuesta del sistema a una entrada escalón usando el PID diseñado .	28
4.7. Respuesta del sistema a una entrada escalón con el PID discretizado .	29
4.8. Esquema del bloque de generación de comandos	30
4.9. Protocolo de almacenamiento del comando	31
4.10. Ejemplo de modulación de una secuencia de bits 0101	32
4.11. Diagrama del bloque de transmisión	32
4.12. Esquema del circuito emisor de infrarrojos	33
4.13. Esquema del bloque de envío de datos	34
5.1. Raspberry PI 2 model B	35
5.2. Imagen del circuito de infrarrojos	47
6.1. Gráfica con los resultados del experimento 1	52
6.2. Gráfica con los resultados del experimento 2	53
6.3. Gráfica con los resultados del experimento 3	53
6.4. Gráfica con los resultados del experimento 4	54
8.1. Diagrama de pareto coef ajuste - error medio experimentos 6 y 13 . .	62
8.2. Diagrama de pareto coef de ajuste - error máximo experimentos 6 y 13	63
8.3. Gráfica con los resultados del experimento 5	67
8.4. Gráfica con los resultados del experimento 6	67
8.5. Gráfica con los resultados del experimento 7	68

ÍNDICE DE FIGURAS

XVII

8.6. Gráfica con los resultados del experimento 8	68
8.7. Gráfica con los resultados del experimento 9	69
8.8. Gráfica con los resultados del experimento 10	69
8.9. Gráfica con los resultados del experimento 11	70
8.10. Gráfica con los resultados del experimento 12	70

Capítulo 1

Introducción

Los centros de procesamiento de datos (CPD) son uno de los elementos más importantes de Internet ya que almacenan, gestionan y procesan la información que se mueve a través de la red. Durante los últimos años se ha producido un notable incremento en el uso de las Tecnologías de la Información y Comunicación (TIC), lo que ha originado un aumento del número, rendimiento y capacidad de almacenamiento de los CPDs. En 2014, las empresas EMC e IDC elaboraron un informe [1] que sostiene que el tamaño del universo digital en 2013 fue de 4,4 ZB de información y estima que dicha cifra se incrementará cada año, alcanzando en 2020 los 44ZB de información, un tamaño 10 veces superior al del 2013.

Esta tendencia implica que los CPDs cada vez tendrán que procesar, almacenar y gestionar más información en el futuro debido al despliegue de nuevo servicios (*cloud computing, IOT, smartcities, e-health...*). Al tener que manejar un mayor volumen de información, su infraestructura será cada vez más grande y tendrá un mayor número de equipos (servidores, routers, switches, equipos de refrigeración e iluminación...), lo que hará que su consumo se incremente. En 2014, un informe elaborado por la empresa Yole Développement [2] muestra que el consumo de energía mundial de todos los CPDs en ese año fue de 352.4 TWh, que equivale al 1.6 % del consumo mundial de energía de ese año, y concluye que si se mantiene esta tendencia, el consumo en 2020 será aproximadamente de 507.9 TWh, lo que supone un incremento del 25 %.

Todo esto supone un gran impacto económico y medioambiental, por lo que es necesario aplicar medidas que permitan que los centros de datos operen de una forma más eficiente y reduzcan su consumo eléctrico. Para contextualizar correctamente este problema, primero es necesario conocer cómo se distribuye el consumo en un CPD y cómo se mide su eficiencia. Una vez se conoce la distribución del consumo, se van a mencionar las técnicas que se utilizan para reducirlo. Por último, se exponen las soluciones que existen actualmente para tratar este problema y se enfoca el papel de este TFG en dicha solución.

1.1. Eficiencia energética en los centros de datos

En la figura 1.1 se muestran 2 gráficos que representan la distribución del consumo de un CPD típico, realizados por Rychard L. Sawyer [3] y la empresa EYP Mission Critical Facilities [4], respectivamente. Ambos gráficos difieren en el porcentaje y en cuál es el tipo de infraestructura que más consume. Sin embargo, ambos coinciden en que los equipos IT y la refrigeración son los 2 tipos de infraestructura que más potencia consumen dentro del CPD, llegando a usar más del 75-80 % de la potencia total.

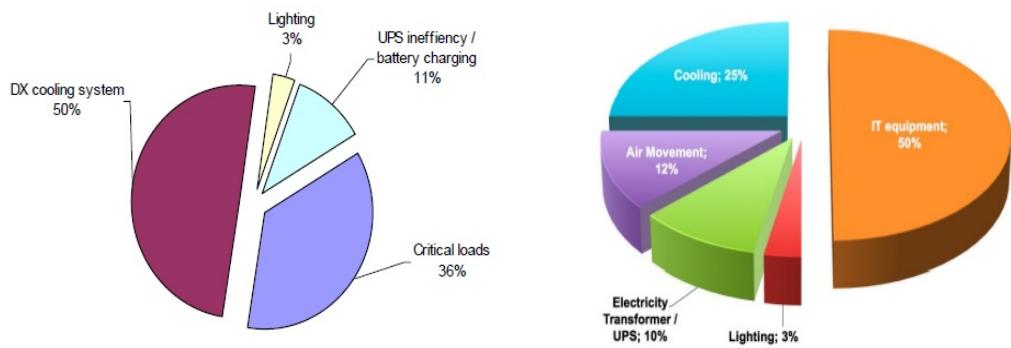


Figura 1.1: Distribución del consumo en un centro de datos

Para conocer la eficiencia de un CPD, se han desarrollado diferentes métricas con el objetivo de medir tanto la eficiencia global del CPD como la eficiencia de cada tipo de consumo. Una de las métricas ampliamente usada por la industria para medir la eficiencia global del CPD es el PUE (*Power Usage Effectiveness*) [5]. Este parámetro relaciona la potencia total consumida en el centro de datos con la potencia consumida por los equipos IT. Dicha relación se muestra en la ecuación 1.1.

$$PUE = \frac{IT + Cooling + PowerLosses + Building}{IT} \quad (1.1)$$

Por lo tanto, el objetivo es conseguir reducir este factor lo más próximo a la unidad. Para ello, se usan principalmente técnicas de eficiencia energética destinadas a reducir el consumo de los equipos IT o el consumo de refrigeración, ya que son los 2 tipos de consumo que más contribuyen al consumo total y su impacto es más significativo. A continuación se mencionan brevemente algunas de estas técnicas.

1.1.1. Técnicas de eficiencia en los equipos IT

Se consideran equipos IT a los servidores, routers, switches y demás elementos que procesan, gestionan y almacenan la información. Se han desarrollado técnicas tanto a nivel **hardware** como a nivel **software** [6] para reducir su consumo.

Desde el punto de vista **hardware**, se usan técnicas como DVFS (*Dynamic Voltage Frequency Scaling*) [7] y DPM (*Dynamic Power Management*) [8]. DVFS es una técnica que permite reducir el consumo de los procesadores mediante la reducción de la tensión de alimentación y la frecuencia de la CPU, según la carga de trabajo del procesador. Esta reducción se basa en la relación cuadrática que existe entre la potencia consumida y la tensión de alimentación de los circuitos ($p \propto C * V^2 * f$). Por otro lado, DPM se basa en la reducción del consumo de potencia ajustando dinámicamente las prestaciones del sistema. Esta técnica puede ser implementada tanto a nivel de circuito como de sistema.

Desde el punto de vista **software**, una de las técnicas más utilizadas es la virtualización. La virtualización permite que un servidor físico albergue múltiples máquinas virtuales (VMs) independientes y haciendo que sea transparente el movimiento de cargas de trabajo entre un servidor y otro. Para ello, se desarrollan algoritmos [9] basados en recolocar de forma dinámica las máquinas virtuales entre los diferentes nodos físicos, teniendo en cuenta criterios de consumo. De este modo, se pretende aumentar el uso de los recursos del sistema y minimizar el número de nodos físicos que están activos, teniendo en cuenta la carga de trabajo que hay en cada momento y buscando el ahorro energético.

1.1.2. Técnicas de eficiencia en la refrigeración

La refrigeración está formada por la unidad de refrigeración, los ventiladores, las canalizaciones y demás elementos que permiten que fluya el aire por el CPD. Para reducir el consumo en esta infraestructura, existe un especial interés en técnicas como el *free cooling* o abordar el problema del sobre enfriamiento u *overcooling*.

El *free cooling* [10] consiste en utilizar el aire del exterior para refrigerar la sala, en lugar de usar el sistema de refrigeración. Cuando la temperatura del exterior es inferior a la temperatura del CPD, el aire caliente fluye hacia el exterior de forma natural, por lo que se puede evitar el uso del compresor y se produce un importante ahorro de energía.

El *overcooling* [11] es un problema que se basa en el sobre enfriamiento de los equipos. La temperatura del aire acondicionado se fija a partir de la potencia térmica máxima que tiene que disipar el sistema de refrigeración cuando el CPD encuentra en el momento de máximo funcionamiento. Este enfoque supone un desaprovechamiento de la energía, ya que el CPD no está siempre a pleno funcionamiento. Si se calcula la temperatura a la que debe funcionar el aire acondicionado en función de la carga del CPD, se puede aumentar en ocasiones la temperatura de salida del aire acondicionado y reducir su consumo. Sin embargo, esto puede provocar un aumento en la temperatura de la sala y de los componentes, lo que provoca un aumento de las corrientes de fugas [12], que no son despreciables, y el aumento del consumo de los equipos.

1.2. Enfoque ciber-físico

Analizando las técnicas anteriormente mencionadas, se observa que dicha técnicas no pueden ser utilizadas de manera aislada, ya que pueden provocar efectos no deseados sobre otros tipos de consumo. Por tanto, es necesario utilizar dichas técnicas de una forma conjunta y coordinada para conseguir que el CPD funcione de manera óptima, es decir, consumiendo la menor cantidad de energía posible en cada momento.

El presente TFG forma parte de una línea de investigación desarrollada por el equipo de optimización energética en centros de datos del Laboratorio de Sistemas Integrados *GreenLSI* [13], perteneciente al Departamento de Ingeniería Electrónica (DIE) de la ETSIT-UPM.

En ella se expone el problema energético que hay en los CPDs y afirma que hay que tratar ese problema mediante la aplicación conjunta de técnicas de eficiencia energética que permitan reducir el consumo de los equipos y de refrigeración [14]. Además, considera que es necesario la monitorización del CPD [15] para que dichas técnicas puedan adaptarse de forma dinámica a la carga computacional del CPD y a las condiciones del entorno, pudiendo aplicar en todo momento la configuración más eficiente. En la figura 1.2 se muestra el esquema del modelo elaborado por *GreenLSI*.

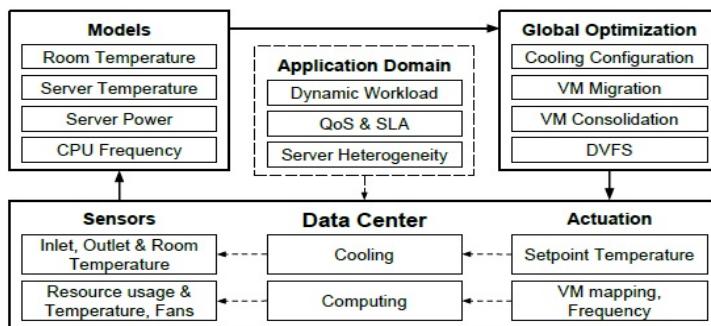


Figura 1.2: Esquema del sistema de optimización. Fuente: [17]

La solución expuesta por *GreenLSI* se basa en un sistema de monitorización y actuación del CPD. Este sistema consta de una red de sensores distribuidos por el CPD que recoge datos de variables relacionadas con el consumo del CPD, tanto del entorno (temperatura, humedad, presión...) como de los equipos (frecuencia de la CPU, temperatura del equipo...). Por otro lado, se han elaborado una serie de modelos [16] que permiten estimar el valor de ciertos parámetros relacionados con el consumo (temperatura de la sala, temperatura de los servidores, potencia de los servidores...).

Con la información proporcionada por los sensores y utilizando los modelos de estimación, se realizan una serie de predicciones sobre el estado del CPD. Despues, con las predicciones obtenidas, se realizan distintas optimizaciones tanto a nivel hardware

como software, con el objetivo de que el CPD tenga un consumo óptimo en ese estado. Por último, se aplican esas optimizaciones realizando diferentes acciones sobre los elementos que conforman el CPD, ya sean los equipos IT o el sistema de refrigeración.

En la figura 1.2 puede verse que una de las optimizaciones realizadas es la configuración de la refrigeración o *cooling configuration*. Esta optimización consiste en ajustar la temperatura de la sala en base al estado del CPD y evitar el problema de sobreenfriamiento de los equipos, con el objetivo de lograr un ahorro energético en el consumo global. Para ello, se irá modificando la temperatura de funcionamiento del sistema de refrigeración o *setpoint*. Este trabajo se centrará en aplicar dicha optimización.

1.3. Objetivos y fases del trabajo

El propósito de este trabajo es diseñar un sistema de actuación que permita controlar la temperatura de la sala y fijarla en todo momento al valor proporcionado por el sistema de optimización energética. El sistema irá modificando el *setpoint* del sistema refrigeración para lograr que la sala alcance la temperatura óptima. Para ello, el sistema de actuación debe tener las siguientes características:

1. **Autonomía:** el actuador tiene que llevar a cabo las acciones necesarias para fijar la temperatura de la sala sin intervención humana.
2. **Dinámico:** el actuador debe responder a los cambios que se produzcan en la temperatura de optimización. Dicha respuesta debe hacerse a la mayor brevedad posible y teniendo en cuenta las limitaciones físicas que pudiera haber.
3. **Estabilidad:** el actuador debe funcionar de una forma correcta, segura y sin comportamientos que puedan afectar al correcto funcionamiento del CPD.
4. **No invasivo:** el funcionamiento del actuador no debe influir en el comportamiento normal de cada uno de los elementos del CPD.
5. **Adaptable:** el actuador debe poder configurarse para que pueda ser usado en diferentes centros de datos. Dicha configuración tiene que ser sencilla y que implique el menor número de cambios posibles.

En cuanto a las fases del trabajo, primero se exponen en el capítulo 2 los principales sistemas de control usados en la industria para controlar la temperatura. Después, en el capítulo 3 se caracteriza el banco de pruebas usado para probar el actuador. A continuación, en los capítulos 4 y 5 se detallan todas las cuestiones relacionadas con el diseño y la implementación del actuador, respectivamente. Luego, en el capítulo 6 se muestran los experimentos realizados y se analizan los resultados obtenidos. Por último, en el capítulo 7, se exponen las principales conclusiones de este trabajo y cuáles son sus principales líneas futuras.

Capítulo 2

Estado del arte

El objetivo de este capítulo es dar una visión general de los sistemas o acciones de control que actualmente son utilizados en la industria para el control de magnitudes físicas, concretamente para controlar la temperatura. En este TFG el sistema de control es una pieza fundamental, ya que es el encargado de generar la señal de control que se aplica en todo momento al sistema de refrigeración para que la sala alcance la temperatura deseada.

Primero se va a explicar el concepto de sistema de control y sus configuraciones más importantes. Después, se van a exponer las principales acciones de control usadas en la industria para realizar el control de la temperatura. Por último, se expone la situación actual en cuanto al control automático de la temperatura en los centros de datos.

2.1. Sistema de control

Un sistema de control puede definirse como un conjunto de dispositivos encargados de administrar, dirigir o regular el comportamiento de otro sistema, y así obtener los resultados deseados. Si enfocamos la definición a este trabajo, la variable a controlar es la temperatura de la sala y esto se consigue controlando el comportamiento del sistema de refrigeración, es decir, su temperatura de funcionamiento o *setpoint*.

Abstrayéndose de la aplicación concreta, un sistema de control sencillo básicamente posee una entrada de referencia y una salida o variable controlada. La entrada de referencia representa el valor que se desea que tenga la variable controlada y la función del sistema de control es conseguir que la variable controlada alcance ese valor. Dependiendo de si el sistema de control usa o no la entrada de referencia, podemos distinguir 2 tipos de configuración [18]:

Configuración en lazo abierto: la variable controlada no es comparada con la entrada de referencia. Por lo tanto, cada valor de la entrada de referencia corresponde a una condición de operación fija. Esta configuración se usa principalmente en sistemas donde no hay perturbaciones y se conoce la relación entre la entrada y la salida. El motivo es que estos sistemas son fáciles de construir y mantener pero son más sensibles a las perturbaciones externas y la precisión de la salida depende de la calibración de los componentes, por lo que es complicado lograr que sean exactos. En la figura 2.1 puede verse un esquema de esta configuración.

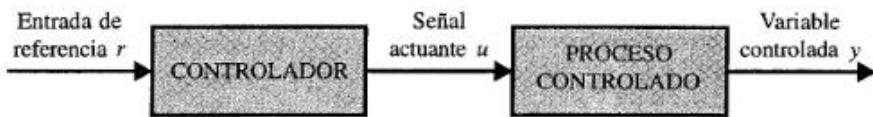


Figura 2.1: Sistema de control en lazo abierto. Fuente: *Benjamin C. Kuo* [19]

Configuración en lazo cerrado: la variable controlada es medida y comparada con la entrada de referencia, generando una señal de error. Dicha señal es utilizada por el controlador para generar la señal de control y lograr que la variable controlada alcance el valor de la entrada. Los sistemas que usan esta configuración son más inmunes frente a las perturbaciones externas y más económicos en el sentido de que puede utilizar componentes más baratos y menos precisos para conseguir el ajuste deseado. Sin embargo, tienen mayores problemas de estabilidad porque el lazo de realimentación puede convertir un sistema estable en uno inestable. En la figura 2.2 puede verse un esquema de esta configuración.

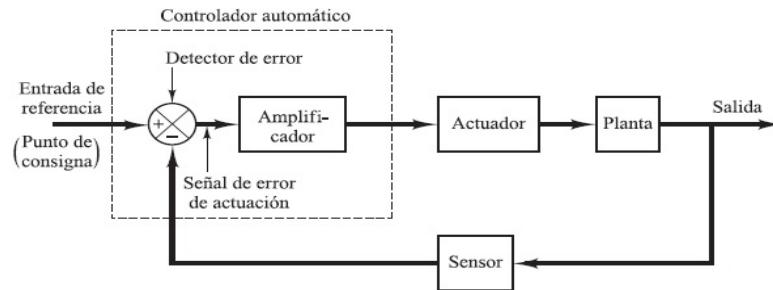


Figura 2.2: Sistema de control en lazo cerrado. Fuente: *Katsuhiko Ogata* [18]

Hay que recalcar que los sistemas de control pueden tener múltiples entradas y múltiples salidas. Sin embargo, estas cuestiones no son tratadas en este TFG debido a que el control se realiza sobre una única variable.

2.2. Tipos de controladores

En este apartado se explican los principales controladores utilizados por la industria para el control de la temperatura. Consultando algunos fabricantes de contro-

ladores como Omron [22], Omega [23], Coulton [24] e Imopc [25], se puede deducir que los controladores más utilizados son:

- Controlador de 2 posiciones
- Controlador Proporcional (P)
- Controlador Proporcional-Integral-Derivativo (PID)

Para entender mejor su funcionamiento, también se van a explicar otros tipos de controladores definidos en la teoría de control [18] [19] [20] y que también son usados en la industria. Estos controladores son:

- Controlador integral (I)
- Controlador Derivativo (D)
- Controlador Proporcional-Integral (PI)
- Controlador Proporcional-Derivativo (PD)

2.2.1. Controlador de 2 posiciones

También es conocido como controlador *On-Off* o *todo-nada*. Es un controlador sencillo y barato, por lo que su uso está extendido tanto en sistemas de control industriales como domésticos.

La señal de control $m(t)$ comuta entre 2 valores posibles, en función de la señal de error $e(t)$. En la figura 2.3 se muestra el esquema de funcionamiento.

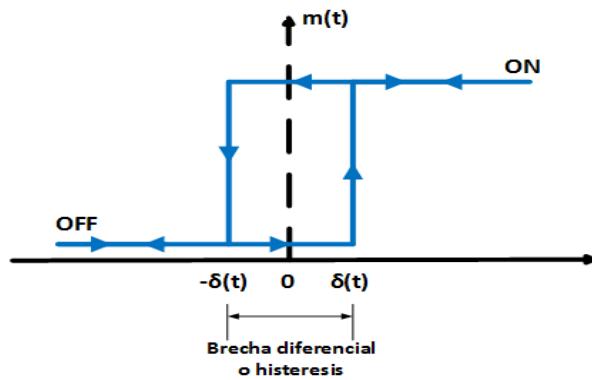


Figura 2.3: Esquema de funcionamiento del controlador *On-Off*

En dicha figura puede verse que la comutación no se realiza en 0 sino en los extremos de un intervalo denominado *brecha diferencial* o *histéresis*. Este mecanismo

permite evitar daños en los componentes de la planta cuando la commutación entre los 2 estados se realiza de manera muy rápida. La señal de control no commuta hasta que la señal de error supera una cierta cantidad $\delta(t)$, tanto si el error es positivo ($e(t) > \delta(t)$) como si es negativo ($e(t) < -\delta(t)$). En ocasiones puede fijarse ese intervalo a 0, siempre y cuando la commutación no sea muy rápida y se garantice el correcto funcionamiento de la planta.

El funcionamiento del controlador es el siguiente: supongamos que la señal de control está en OFF y el error es negativo. Cuando $e(t) > \delta(t)$, la señal de control commuta al estado ON. Este estado habitualmente se traduce en una acción que hace que la planta a controlar se encienda y funcione (encender un motor, abrir una válvula...).

Supongamos ahora que la señal de control está en ON y el error es positivo. Si se cumple que $e(t) < -\delta(t)$, la señal de control commuta al estado OFF. Este estado habitualmente se traduce en una acción que hace que la planta a controlar se apague o deja de funcionar (apagar un motor, cerrar una válvula...).

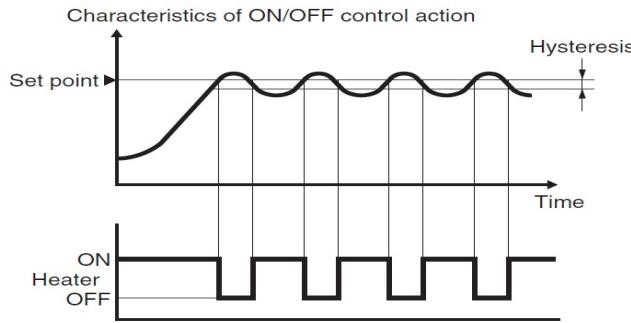


Figura 2.4: Respuesta de un sistema con un controlador *On-Off*. Fuente: Omron [22]

El funcionamiento del controlador es sencillo aunque presenta limitaciones en cuanto a que sólo puede usarse para aquellos sistemas que puedan ser controlados mediante 2 estados de funcionamiento.

2.2.2. Controlador proporcional (P)

Su acción se basa en multiplicar la señal de error por una constante denominada *constante proporcional* K_p . Su expresión matemática es la siguiente:

$$m_P(t) = K_p e(t) \quad (2.1)$$

El controlador amplifica la señal de error para conseguir que la señal medida siga a la señal de referencia. Sin embargo, analizando la ecuación, se ve que no se consigue eliminar el error estacionario ya que si el error fuese nulo, la señal de control también lo sería.

Este controlador puede ajustarse a través de la constante K_p o mediante el concepto de la **banda proporcional**, que se define como la cantidad que tiene que cambiar la variable controlada para lograr un cambio del 100 % en la acción de control. Su expresión matemática es $BP\% = \frac{100}{K_p}$ y es un concepto similar a la ganancia proporcional. En la figura 2.5 se muestra una gráfica que representa el comportamiento de esta acción.

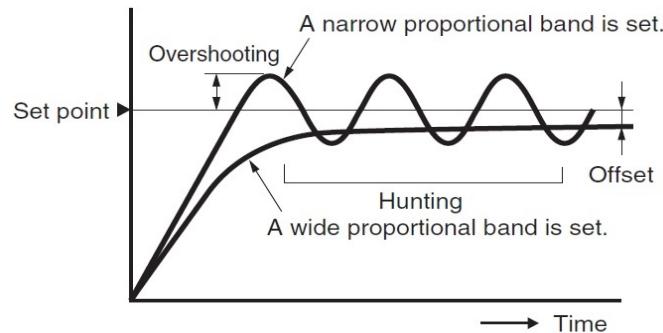


Figura 2.5: Respuesta de un sistema con un controlador P. Fuente: Omron [22]

En dicha gráfica puede verse que si la banda proporcional es amplia (equivale a una K_p pequeña), la respuesta de la planta se approxima al valor de referencia pero tiene un ligero offset, mientras que si la banda es reducida (equivale a una K_p grande), la respuesta presenta oscilaciones entorno al valor de referencia.

2.2.3. Controlador Integral (I)

Su acción se basa en integrar la señal de error y multiplicarla por una constante denominada *constante integral* K_i . Su expresión matemática es la siguiente:

$$m_I(t) = K_i \int_0^t e(t) dt \quad (2.2)$$

El controlador genera una señal que es función de la "historia" de la señal de error, lo que permite conseguir una señal de control no nula aunque la señal de error sí lo sea. Este controlador consigue eliminar el error estacionario aunque empeora la estabilidad del sistema, aumentando el sobreimpulso de la respuesta transitoria e incluso puede hacer que el sistema se vuelva inestable. No suele utilizarse sólo sino que se combina con otras acciones de control, por ejemplo, con un controlador proporcional.

2.2.4. Controlador Proporcional Integral (PI)

Este controlador se basa en una combinación de la acción proporcional y la acción integral. Su expresión matemática es la siguiente:

$$m_{PI}(t) = K_p e(t) + K_i \int_0^t e(t) dt = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(t) dt \right) \quad (2.3)$$

Ajustando los parámetros K_p y K_i se consigue ajustar la respuesta a los requisitos deseados. El ajuste de la acción integral puede realizarse mediante K_i o usando un parámetro denominado *tiempo integral* T_i , que se define como $T_i = \frac{K_p}{K_i}$.

En la figura 2.6 se muestra la respuesta de una planta con este controlador. En dicha figura se observa como la acción integral consigue eliminar el error estacionario y que la respuesta se aproxime al valor deseado. Es necesario ajustar correctamente los valores K_p y K_i , ya que puede ocurrir que la señal tenga un mayor sobreimpulso o que sea demasiado lenta y tarde más tiempo en alcanzar el estado estacionario.

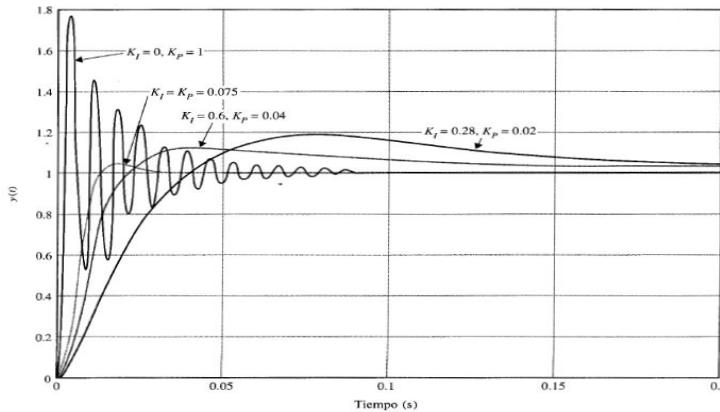


Figura 2.6: Respuesta de un sistema con una acción PI. Fuente: *Benjamin C. Kuo* [19]

2.2.5. Controlador Derivativo (D)

Su acción se basa en derivar la señal de error y multiplicarla por una constante denominada *constante derivativa* K_d . Su expresión matemática es la siguiente:

$$m_D(t) = K_d \frac{de(t)}{dt} \quad (2.4)$$

Esta acción añade sensibilidad al sistema y permite corregir el error antes de que se vuelva excesivo. Provoca un aumento en la estabilidad relativa que se traduce en un menor sobreimpulso y una respuesta con un menor tiempo de subida y de establecimiento. Sin embargo, no puede utilizarse sólo porque no es capaz de eliminar el error estacionario para una señal constante. Suele combinarse con otras acciones de control, por ejemplo, con un controlador proporcional.

2.2.6. Controlador Proporcional Derivativo (PD)

Este controlador se basa en la combinación de la acción proporcional y la acción derivativa. Su expresión matemática es la siguiente:

$$m_{PD}(t) = K_p e(t) + K_d \frac{de(t)}{dt} = K_p \left(e(t) + T_d \frac{de(t)}{dt} \right) \quad (2.5)$$

Ajustando los parámetros K_p y K_d se consigue ajustar la respuesta a los requisitos deseados. El ajuste de la acción derivativa puede hacerse mediante K_d o con un parámetro denominado *tiempo derivativo* T_d , que se define como $T_d = \frac{K_d}{K_p}$.

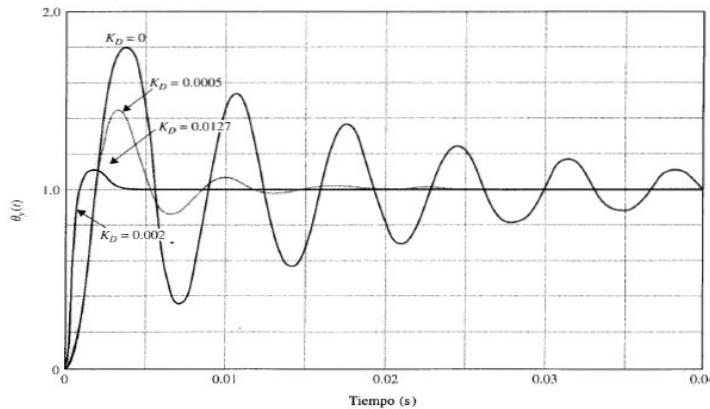


Figura 2.7: Respuesta de un sistema con una acción PD. Fuente: Benjamin C. Kuo [19]

En la figura 2.7 se muestra la respuesta de una planta con este controlador. En dicha figura se observa que a medida que aumenta la acción derivativa, la respuesta tiene un menor sobreimpulso, es más rápida y tiene un menor tiempo de establecimiento.

2.2.7. Controlador PID

Combina las ventajas de las 3 acciones . Su expresión matemática es:

$$m(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt} = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(t) dt + T_d \frac{de(t)}{dt} \right) \quad (2.6)$$

En la figura 2.8 se compara la respuesta de una planta usando un controlador PID frente a un controlador PI y un controlador PD. Ajustando los parámetros adecuadamente, se consiguen los resultados deseados. Este controlador es ampliamente usado en la industria debido a que puede adaptarse a un amplio rango de valores de funcionamiento y a que posee un planteamiento sencillo.

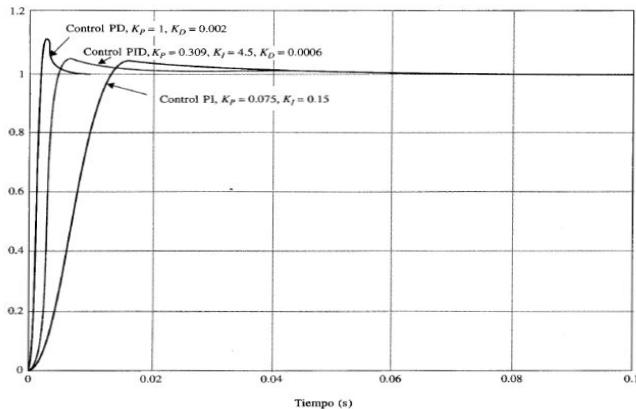


Figura 2.8: Respuesta de un sistema usando un PID. Fuente: *Benjamin C. Kuo* [19]

Este controladores posee diferentes representaciones, dependiendo de las necesidades. En la figura 2.9 se muestran algunas de ellas. El 1º esquema es la representación paralela o ideal y se caracteriza en que cada una de las acciones no se influyen mutuamente. Su expresión matemática es la primera parte de la ecuación 2.6.

El esquema central es la representación estándar o no interactiva y se basa en que la acción proporcional influye en el resto de acciones pero la acción integral y derivativa no se influyen entre sí. Su expresión matemática es la segunda parte de la ecuación 2.6. Por último, la figura de la derecha es la representación serie o interactuante y se caracteriza porque la acción integral influye en la derivativa o viceversa.

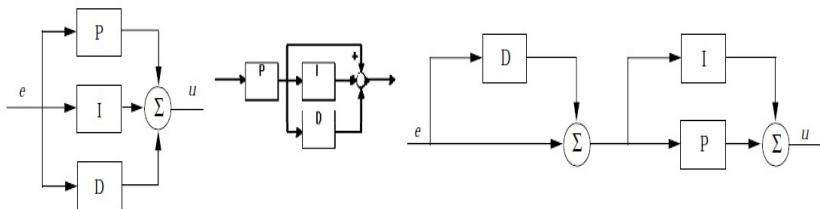


Figura 2.9: Esquemas de representación de un PID. Fuente: *Katsuhiko Ogata* [20]

Este controlador tiene un funcionamiento lineal pero en muchas ocasiones pueden aparecer efectos no lineales. Una no linealidad típica es el efecto windup. El actuador posee un rango de funcionamiento limitado y si el sistema de control tiene un amplio rango de operación, puede ocurrir que la señal de control alcance el límite del actuador.

Cuando esto sucede, el lazo de realimentación se rompe porque el actuador se satura y deja de seguir a la señal de control. Sin embargo, el error se sigue acumulando y la señal de control sigue aumentando. Cuando la señal de error cambia de signo, la señal de control empieza a reducirse pero requiere de un gran tiempo para volver a entrar en la región lineal. Esto provoca grandes transitorios en la respuesta de la planta. En la figura 2.10 se representa este efecto.

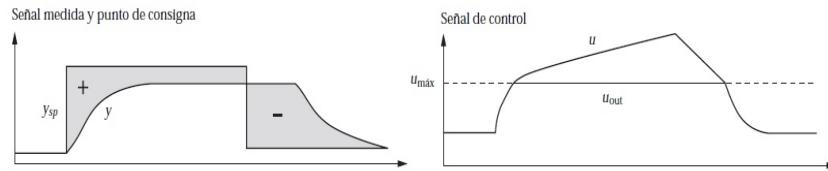


Figura 2.10: Representación del efecto windup. Fuente: *Katsuhiko Ogata* [20]

Para solucionarlo, existen varios métodos como limitar el valor de referencia para que la señal de control no alcance los límites del actuador, la integración condicional que se basa en habilitar la acción integral si se cumplen determinadas condiciones o la técnica del recálculo y seguimiento que recalcula la integral cuando el actuador se satura y consigue que la señal de control se aproxime al límite del actuador [20].

2.3. Control de temperatura

En este apartado se pretende exponer cómo se realiza el control de la temperatura en un centro de datos, es decir, como se ajusta el valor de referencia según la situación del CPD.

En la actualidad, existen diferentes herramientas de control y gestión de los centros de datos, como Data Center Infrastructure Management (DCIM) o Supervisory Control And Data Acquisition (SCADA).

El sistema SCADA supervisa y controla las operaciones a través de sensores que están colocados en diferentes lugares y que son monitorizados desde una unidad centralizada. Las funciones incluyen gestión de alarmas, diagnóstico, mantenimiento, interfaces gráficas que muestran la situación actual del CPD, entre otros.

El sistema DCIM está formado por un sistema software, hardware y sensores que permiten gestionar, supervisar y planificar la capacidad de la infraestructura crítica de un CPD. El sistema maneja los datos proporcionados por los sensores y con ellos puede gestionar tanto los servidores como el resto de equipos. Dispone de una plataforma de gestión y monitorización para visualizar el estado del CPD en tiempo real.

Sin embargo, estos sistemas son complejos de manejar y requieren de algún tipo de intervención humana, incluyendo el caso de la temperatura. Estos sistemas monitorizan y controlan la temperatura pero se necesita un operario para realizar los cambios.

Por tanto, el sistema de actuación diseñado en este trabajo debe ser capaz de proporcionar al controlador el dato óptimo de temperatura de una forma automática y sin ningún tipo de acción humana.

Capítulo 3

Banco de pruebas

En este capítulo se describe el banco de pruebas en el que se va a verificar el funcionamiento del actuador. También se detallan algunas características del banco que son importantes para la etapa de diseño.

El banco de pruebas elegido es la sala B039 del Departamento de Ingeniería Electrónica. En esta sala se guardan gran parte de los servidores que almacenan la información del departamento, por lo que se trata de un entorno real y adecuado para probar el actuador. Además, se encuentra monitorizado y está implementado una parte del sistema de monitorización explicado en el apartado 1.2. En la figura 3.1 se muestra una imagen de dicha sala.

En esta sala existen 2 sistemas de refrigeración. Sus características son las siguientes:

- **Sistema de refrigeración del edificio:** su principal objetivo es mantener una temperatura de confort en cada una de las salas del edificio. Sin embargo, este sistema no es suficiente para evacuar el calor generado por los equipos, debido a que sólo funciona durante los meses de verano. Además, no existe ningún mecanismo que permita controlar el sistema para que se pueda fijar la temperatura de esta sala según sus necesidades y de manera independiente al resto de salas del edificio. Por tanto, este sistema es descartado para el control de la temperatura.
- **Unidad de refrigeración comercial:** su objetivo es la evacuación del calor generado por los equipos de la sala. Este sistema funciona durante todo el año y permite configurar el *setpoint*. Por tanto, el control de la temperatura se va a realizar utilizando este sistema.

El control del *setpoint* se realiza de forma manual y a través de un mando a distancia. Este mando dispone de varias teclas que permiten configurar el modo de

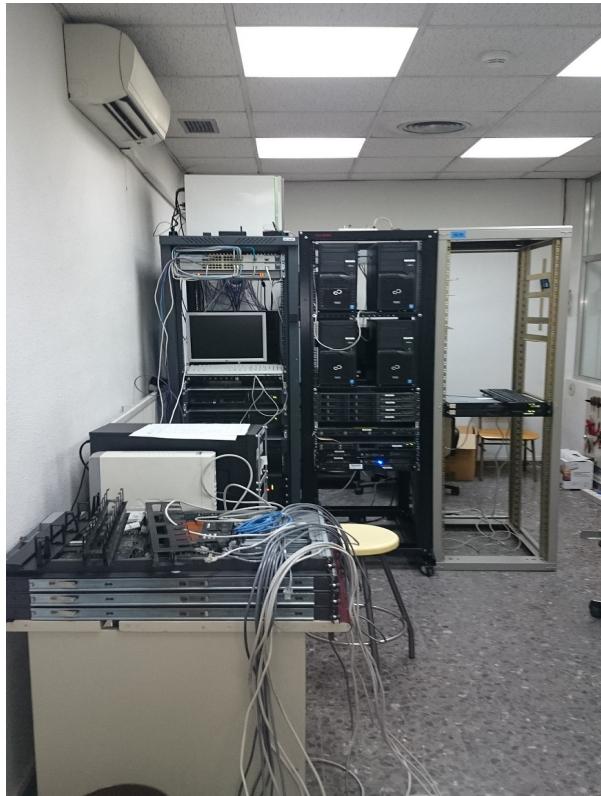


Figura 3.1: Fotografía de la sala B039

funcionamiento. Al pulsar alguna de estas teclas se envía una orden al sistema de refrigeración para que modifique su estado de funcionamiento.

Cada una de las órdenes o modos de funcionamiento es representada a través de un comando. El comando está formado por una secuencia de bits con un determinado protocolo o formato que es interpretado por el receptor situado en la unidad de refrigeración. El receptor decodifica la secuencia recibida y en base a ello, modifica el funcionamiento del sistema de refrigeración.

Cada comando es enviado al receptor mediante una señal de infrarrojos. Dicha señal está modulada a una frecuencia de 38 KHz, que es la frecuencia de portadora fijada por el fabricante para la comunicación a través del mando a distancia. El sistema posee un rango de funcionamiento que va de 18°C a 32°C, con saltos de 1°C entre 2 temperaturas consecutivas.

Por tanto, la comunicación del sistema de actuación con la unidad de refrigeración deberá realizarse mediante un sistema de infrarrojos que emule el comportamiento del mando a distancia y que lo haga de manera automática. Además, deberá tener en cuenta las cuestiones mencionadas sobre el rango de funcionamiento y la modulación.

Capítulo 4

Diseño

En este capítulo se expone de manera detallada el diseño del sistema de actuación y se analiza cada uno de los subsistemas que lo componen. El diseño se ha hecho teniendo en cuenta los requisitos indicados en el apartado 1.3 del capítulo 1.

4.1. Descripción del sistema completo

La arquitectura elegida se basa en un sistema de control en lazo cerrado, ya que hace al sistema más inmune frente a las perturbaciones y logra unos mejores ajustes. No se ha escogido la configuración en lazo abierto porque requiere de unos ajustes más precisos y es más sensible a las perturbaciones. Además, la temperatura de la sala depende de factores como el aislamiento de la sala, la temperatura exterior, el calor generado por los equipos, etc, cuyo impacto en la respuesta global es difícil de caracterizar con exactitud. En la figura 4.1 se muestra un esquema de la arquitectura.

El sistema de actuación tiene 2 entradas: la temperatura a la que debe estar la sala o *temperatura óptima*, proporcionada por el sistema de optimización y la temperatura a la que se encuentra la sala o *temperatura medida*, proporcionada por el sensor. El sistema tiene una única salida que es el *setpoint* del aire acondicionado. El sistema está formado por los siguientes bloques:

- Bloque de adquisición de datos.
- Bloque de comparación.
- Bloque de control.
- Bloque de generación del comando.
- Bloque de transmisión.

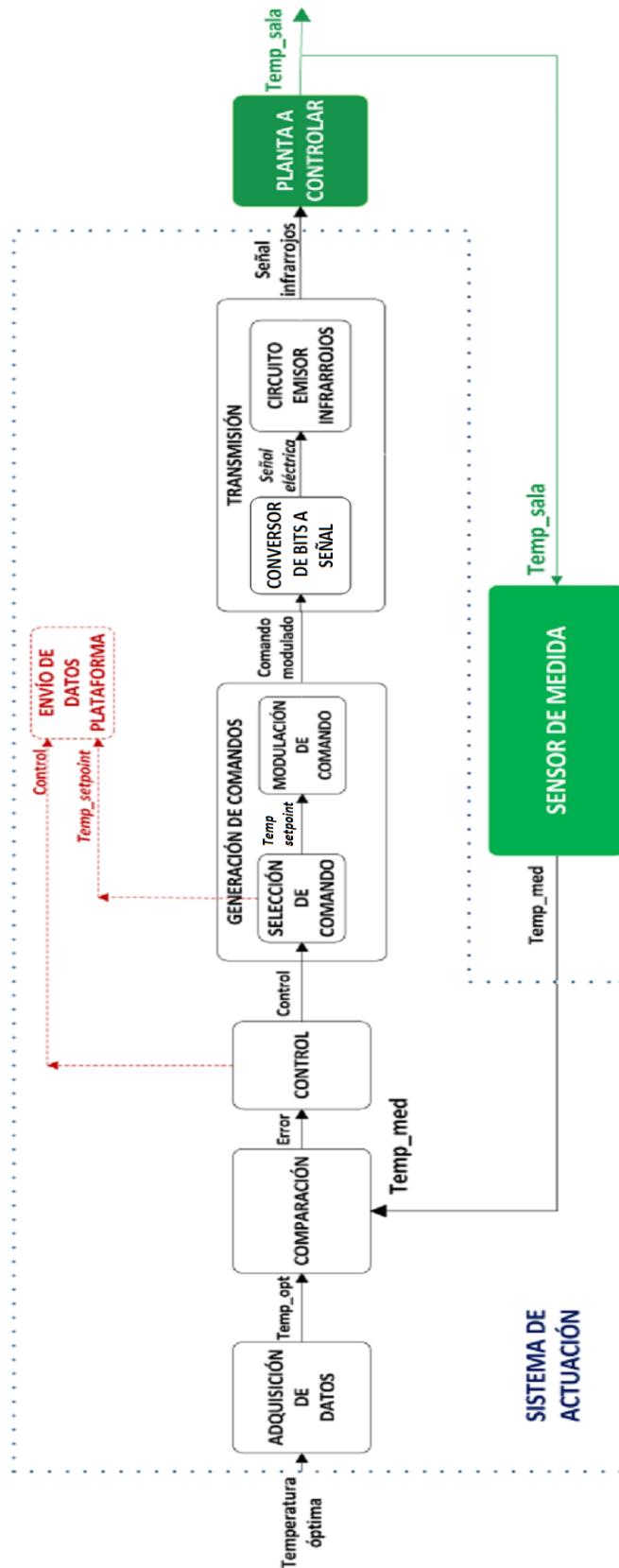


Figura 4.1: Diagrama del sistema completo

En la figura 4.1 también se incluyen la planta a controlar y el sensor de medida. Estos componentes son fundamentales para explicar el funcionamiento del actuador pero no forman parte de él, por lo que no son considerados en la etapa de diseño.

También existe un bloque de envío de datos que se utiliza para mandar datos a la plataforma de monitorización y poder visualizar y evaluar el correcto funcionamiento del actuador. Este bloque se incluye en el diseño, aunque no es fundamental para el funcionamiento del actuador.

En los siguientes apartados se hace una descripción detallada de cada uno de los bloques del actuador.

4.2. Bloque de adquisición de datos

Es la primera etapa del sistema de actuación. El dato de *temperatura óptima* y el dato de *temperatura medida* se encuentran almacenados en una plataforma y con un determinado formato. Hay que tener en cuenta el tipo de plataforma y el formato utilizado, ya que ambos varían según el CPD e incluso puede darse el caso de que ambos datos estén almacenados en plataformas diferentes y con distintos formatos. También hay que considerar que el dato puede estar almacenado con un formato que no permite al actuador utilizarlo directamente y necesita ser procesado.

El actuador va a disponer de 2 bloques de adquisición, uno para cada dato. Cada bloque tendrá como entrada el dato de temperatura almacenado en la plataforma y como salida ese mismo dato procesado para que el actuador pueda utilizarlo. En la figura 4.2 se muestra un esquema de cada uno de estos bloques.



Figura 4.2: Esquema del bloque de adquisición de datos

Para este trabajo se va a usar *Graphite* como plataforma de almacenamiento de datos debido a que los sensores de la sala B039 envían los datos a un servidor que utiliza dicha plataforma. Según su documentación [26], *Graphite* almacena series de datos junto con su marca temporal o *timestamp*. Además, proporciona una aplicación web a través de la cual se pueden visualizar los datos de forma gráfica o se pueden exportar en diferentes formatos. En la figura 4.3 se muestra una imagen de la interfaz gráfica de *Graphite*.

Graphite permite exportar los datos que tiene almacenados, en diferentes formatos. De entre todos ellos, se ha decidido utilizar JSON. El motivo es que tiene un

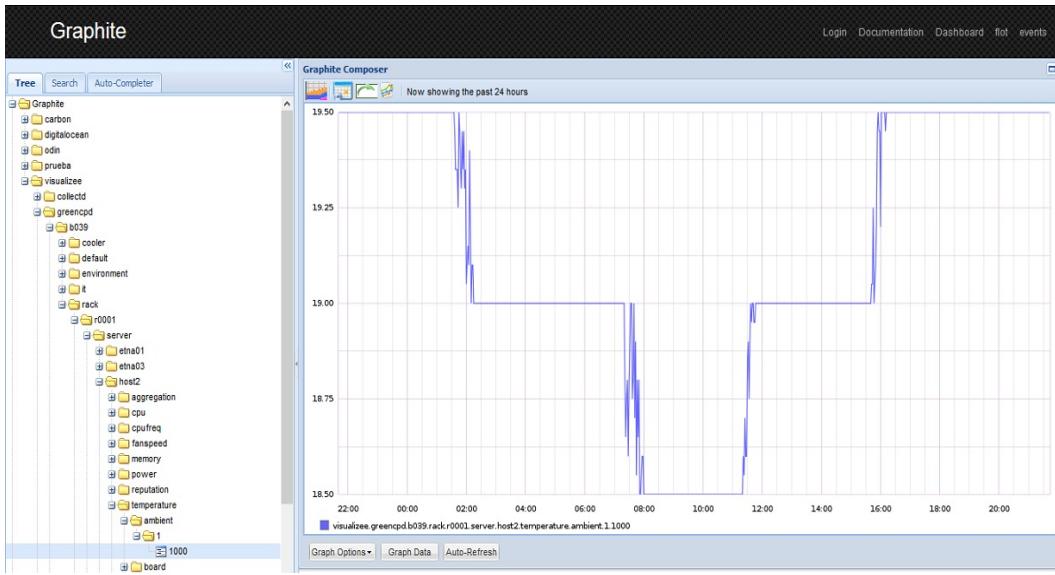


Figura 4.3: Interfaz gráfica de *Graphite*

esquema de representación fácil de entender y el procesamiento del dato es más fácil de hacer en este formato que en el resto de formatos existentes. A continuación se muestra un ejemplo de un dato que ha sido exportado de *Graphite* en formato JSON:

```
[{"target": "visualizee.greencpd.rack.b039.cooler.temperature.supply.setpoint.2", "datapoints": [[18.0, 1496395200], [18.0, 1496395210], [19.0, 1496395220], [21.0, 1496395230], [23.0, 1496395240], [22.0, 1496395250]]}]
```

El objeto JSON posee 2 campos: **target** y **datapoints**. El campo **target** almacena una cadena de caracteres con la ruta en la que se encuentra el dato que se quiere exportar. El campo **datapoints** es un array de N-duplas donde cada dupla contiene el valor de la temperatura junto con el instante de medida o *timestamp*. El valor de la temperatura es un número real con una precisión de centésimas y el *timestamp* es un entero que expresa el tiempo en formato epoch.

Los datos almacenados en *Graphite* se pueden exportar realizando una petición http. A continuación se muestra un ejemplo del formato de una url usada para exportar un dato de la plataforma.

```
http://visualizee.die.upm.es:8000/render?format=json&target=visualizee.greencpd.b039.rack.r0001.server.host2.temperature.ambient.1.1000&from=-1min&until=-4min
```

Según la documentación [26], la url contiene 4 parámetros que son configurados

para seleccionar el dato que se desea exportar, el formato en el que se obtienen los datos y el número de muestras que se van a exportar. Estos parámetros son:

- **format:** contiene el formato en el que se desea extraer los datos.
- **target:** contiene la ruta donde se encuentra el dato a exportar.
- **from:** contiene el instante de inicio de la toma de muestras. Si no se especifica nada, se toma por defecto las últimas 24 horas.
- **until:** contiene el instante final de la toma de muestras. Si no se especifica nada, se toma por defecto el instante actual.

En cada petición es recomendable exportar varias muestras y no sólo la del instante de la iteración, ya que puede darse el caso de que el sistema de actuación haga la petición del dato de temperatura y el sensor todavía no lo haya medido o genere null. De este modo, se soluciona este problema y el error cometido es mínimo porque la temperatura es una variable que evoluciona lentamente y la variación que puede producirse entre muestras muy próximas es muy pequeña. En este trabajo, el sensor de banco de pruebas toma muestras cada 10 segundos, luego se van a exportar las muestras de *temperatura óptima* y *temperatura medida* tomadas en el último minuto para tener un cierto margen de seguridad.

Por último, una vez exportado el dato de la plataforma y extraído del objeto JSON, éste se multiplica por un factor de conversión para convertirlo a un número entero que tenga la precisión necesaria para ser utilizado en las siguientes etapas. En este trabajo, el bloque de comparación y de control van manejar números enteros que representan a números reales con una precisión de milésimas. Por tanto, el factor de conversión es 1000. De este modo, se conserva la precisión del dato de temperatura y ya está preparado para ser usado en próximas etapas.

4.3. Bloque de comparación

Su función es calcular el error que hay entre la *temperatura óptima* y la *temperatura medida*. En la figura 4.4 se muestra un diagrama de este bloque.



Figura 4.4: Diagrama del bloque de comparación

El bloque resta a la *temperatura óptima* el valor de la *temperatura medida*. El resultado de dicha operación es el valor de la señal de error. El error está expresado en las mismas unidades que las temperaturas.

4.4. Bloque de control

Su función es generar la señal de control que se enviará al sistema de refrigeración para que la sala alcance la *temperatura óptima*. Dicha señal es generada a partir de la señal de error y siguiendo un determinado algoritmo o política de control. En la figura 4.5 se muestra un esquema con las entradas y salidas del bloque.



Figura 4.5: Esquema del bloque de control

De todos los controladores explicados en el capítulo 2, se ha escogido el controlador PID, ya que es un controlador ampliamente usado en la industria, incluyendo procesos de control de la temperatura, con un planteamiento sencillo y con diferentes métodos de diseño y ajuste. Se ha decidido usar la representación ideal o paralela.

El ajuste del controlador puede realizarse tanto en el dominio del tiempo como en la frecuencia [21]. Algunos de los métodos utilizados son: Ziegler-Nichols, diagrama de bode, criterio de Routh-Hurwitz, lugar de raíces, método de prueba y error...

Algunos de estos métodos permiten el ajuste de la planta sin necesidad de conocer su función de transferencia. Sin embargo, es recomendable obtener dicha función, para así facilitar el proceso de ajuste de los parámetros. Por otro lado, la expresión anterior está definida para un controlador en tiempo continuo. Sin embargo, el controlador se va a implementar en un sistema digital, luego es necesario discretizarlo.

Por tanto, el proceso de diseño del controlador consta de las siguientes fases: primero, se estima la función de transferencia que modela la planta a controlar. Después, se diseña el controlador PID y se ajustan sus parámetros usando alguno de los métodos anteriormente descritos. Por último, se discretiza el controlador PID diseñado. A continuación, se explican cada una de estas fases en los siguientes apartados.

4.4.1. Caracterización de la planta

En esta fase se va a estimar la función de transferencia que representa el comportamiento dinámico de la planta. La variable a controlar es la temperatura de la

sala y dicho control se va a realizar a través del sistema de refrigeración de la sala. Por tanto, la planta a controlar está formada por el sistema de refrigeración y la sala.

La planta puede ser caracterizada aplicando modelos físicos de los distintos componentes que componen la planta (ciclo de refrigeración + sala). Sin embargo, esto es poco viable porque hay parámetros que son difíciles de caracterizar y medir.

Por este razón, se decide hacer la caracterización de la sala mediante la técnica de identificación de sistemas. Se van a realizar una serie de experimentos basados en excitar la planta con una señal de entrada específica (escalón, rampa...) y se miden los valores generados en la salida. Despues, se aplican métodos estadísticos que permiten estimar una función matemática que representa el comportamiento de la planta.

El experimento realizado en este trabajo se basa en aplicar una señal de entrada de tipo escalón al sistema de refrigeración y medir la temperatura de la sala. El valor inicial y el valor final de la señal escalón son el valor mínimo y máximo de funcionamiento del sistema de refrigeración (18°C y 32°C respectivamente). De este modo, se pretende abarcar todo el rango de funcionamiento del sistema de refrigeración y caracterizar mejor la planta.

Hay que tener en cuenta que aunque la temperatura máxima sea de 32°C , la sala no puede alcanzar ese valor, ya que la temperatura máxima recomendada para el buen funcionamiento de la sala está entre (25°C - 27°C) y si se supera, podrían dañarse los equipos. Por tanto, el experimento se mantendrá hasta que la sala alcance ese valor recomendado.

Los valores de la salida son medidos cada 10 segundos, ya que es el periodo de muestreo del sensor. Podría haberse elegido un periodo de muestreo superior, por ejemplo 1 min, ya que la evolución de la temperatura es un proceso lento. Sin embargo, se opta por un periodo igual al periodo de muestreo del sensor y se evita tener que hacer un procesamiento adicional. Este experimento se repetirá varias veces para obtener un mayor número de muestras.

Una vez realizados todos los experimentos, se estima la función transferencia de la planta, utilizando el programa de cálculo matemático MATLAB. Se va a diseñar un script con el que se obtiene los valores de la señal de entrada y de salida utilizadas en cada experimento. Despues se estima la función de transferencia usando la función de matlab *tfest*. Esta función toma como parámetros los datos de entrada y de salida y el número de polos y ceros del sistema. Para cada conjunto de datos se van a realizar todas las combinaciones posibles de polos y ceros hasta orden 3, ya que este tipo de sistemas suelen ser de 1º o 2º orden y no es recomendable usar sistemas cuyo orden es muy elevado debido al problema de sobreajuste u *overfitting*. El objetivo es estimar una función que tenga un buen ajuste con cualquier conjunto de datos.

Una vez se han obtenido todas las combinaciones, se escoge aquella que tenga un mejor coeficiente de ajuste. Este coeficiente de ajuste es proporcionado por la misma

función *tfest* y es calculado mediante el error cuadrático medio normalizado o NRMSE *Normalized Root-Mean-Square Error*.

La función *tfest* presenta el inconveniente de que sólo puede manejar un conjunto de datos a la vez, por lo que no se puede calcular una estimación basada en los conjuntos de datos de todos los experimentos. Para solucionar este problema, se calcula el coeficiente de ajuste que tiene cada función de transferencia de cada experimento, con respecto a sus datos y a los datos de los otros experimentos y se calcula el valor medio. Con esta información se selecciona la función que presente un mejor ajuste tanto a sus datos como a los datos del resto de experimentos.

En caso de que hubiera varios experimentos con un coeficiente de ajuste medio parecido, se calcula también el error medio y el máximo error absoluto y se representa cada parámetro junto con el coeficiente de ajuste en un diagrama de pareto para decidir mejor cuál de ellas se ajusta mejor.

Este proceso de caracterización descrito, en principio puede ser aplicado a otros bancos de pruebas, siempre y cuando sea adaptado a sus características particulares.

Por claridad, en este apartado sólo se muestra la función escogida. En el anexo 8.1 se detallan todas las funciones de transferencia de todos los experimentos, así como el proceso de selección de la función de transferencia. A continuación se muestra la función de transferencia que caracteriza a la planta.

$$G(s) = \frac{8,4315 \cdot 10^{-5}s + 1,6968 \cdot 10^{-9}}{s^3 + 0,2820s^2 + 2,1553 \cdot 10^{-4}s + 1,0462 \cdot 10^{-14}} \quad (4.1)$$

4.4.2. Diseño del controlador PID

El diseño del controlador consiste en ajustar los parámetros K_p , K_i y K_d para que la respuesta cumpla una serie de especificaciones. El controlador que se va a diseñar debe cumplir las siguientes especificaciones.

- El sistema debe ser estable en todo momento.
- El error estacionario para una señal escalón debe ser próximo a cero con una tolerancia del 5 %.
- El tiempo de establecimiento del sistema debe ser menor de 10 min. Este requisito puede verse incumplido debido a las limitaciones físicas. La temperatura tiene una evolución lenta y no puede fijarse un tiempo de establecimiento pequeño.

Para ajustar los parámetros del controlador se decidió inicialmente usar el método de Ziegler-Nichols [21]. Este método posee 2 versiones que permiten ajustar el

sistema dependiendo de si la planta tiene una respuesta con forma de *es*e o posee una respuesta con oscilaciones para una determinada ganancia K_{cr} .

La función estimada no cumplía los requisitos para aplicar la 2º versión, por lo que se decidió aplicar la 1º versión. Aunque la respuesta de la planta en lazo abierto era estable, no presentaba dicha forma, por lo que resultaba complicado aplicar este método y también fue descartado. También se decidió probar otros métodos como el diagrama de bode pero al tener un margen de ganancia infinito y un margen de fase elevado resultaba complicado establecer un patrón para realizar el ajuste.

Por tanto, se ha decidido escoger el método de ajuste y error. Este método consiste en ir ajustando los parámetros del controlador PID hasta conseguir que se cumplan las especificaciones fijadas. El proceso llevado a cabo es el siguiente:

1. Se fijan los valores de K_i y K_d al valor más pequeño posible y se fija la constante proporcional a 1. Se va incrementando dicha ganancia hasta conseguir oscilaciones estables y próximas al valor deseado.
2. Se incrementa el valor de K_i hasta conseguir eliminar el error estacionario. Si es necesario, se disminuye ligeramente el valor K_p para eliminar las oscilaciones.
3. Se incrementa el valor de K_d hasta conseguir la respuesta deseada pero más rápida. Si es necesario, se aumenta ligeramente el valor de K_p .

El proceso de ajuste de los parámetros del controlador y la simulación de la respuesta del sistema se ha realizado a través de MATLAB. Se ha diseñado un script que simula el sistema de control en lazo cerrado formado por el controlador PID y la función de transferencia de la planta a controlar. Este script genera la respuesta del sistema de control en base a los parámetros del PID y permite evaluar la estabilidad del sistema mediante el diagrama de Nyquist. En el anexo 8.3 puede verse el script elaborado.

A continuación se muestra en la figura 4.6 una imagen con la respuesta obtenida en la simulación. Los valores de K_p , K_i y K_d que cumplen con las especificaciones son:

$$K_p = 28; \quad K_i = 0,037; \quad K_d = 300;$$

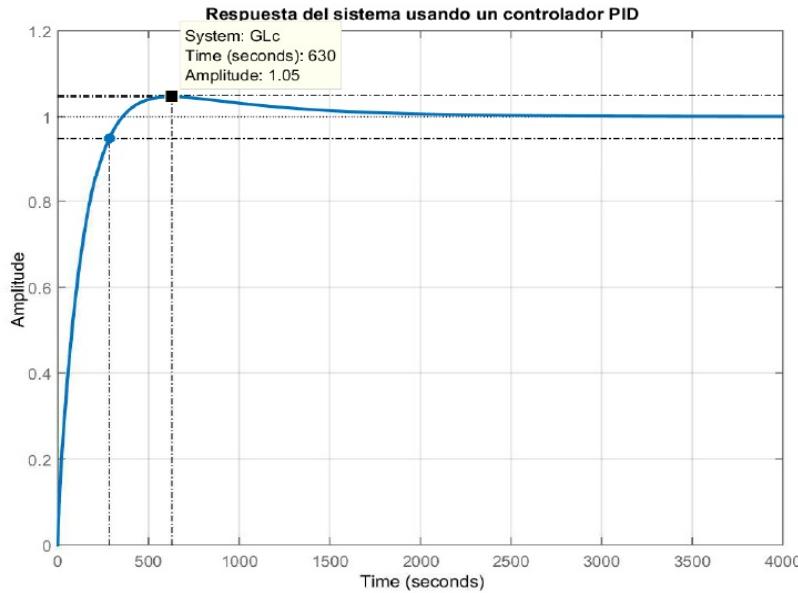


Figura 4.6: Respuesta del sistema a una entrada escalón usando el PID diseñado

4.4.3. Discretización del controlador PID

La discretización del controlador PID consiste en obtener la expresión matemática del controlador PID en tiempo discreto para que pueda ser implementado en un sistema digital. Para ello, se va a discretizar cada una de las acciones que definen al controlador PID y se agrupan las expresiones obtenidas en cada discretización. A continuación se detalla la discretización de cada acción:

- **Discretización de la acción proporcional:** es la más fácil de realizar ya que se basa en el muestreo de la señal de error.

$$m_p(n) = K_p e(t)|_{t=kT_s} = K_p e(kT_s) = K_p e(n) \quad (4.2)$$

- **Discretización de la acción integral:** se basa en la discretización de la integral. Existen diferentes métodos para poder discretizarla (integración hacia delante, integración hacia atrás, método de tustin...) [21]. Se ha decidido escoger el método de tustin ya que es una aproximación más precisa y permite conservar la estabilidad del sistema en tiempo discreto.

$$\begin{aligned} m_i(n) &= K_i \int_0^t e(t) dt \Big|_{t=kT_s} = K_i \sum_{k=1}^n T_s \frac{e(kT_s) + e([k+1]T_s)}{2} \\ &= m_i(n-1) + T_s \frac{e(kT_s) + e([k+1]T_s)}{2} \end{aligned} \quad (4.3)$$

- **Discretización de la acción derivativa:** se basa en la aplicación del método de Euler hacia atrás. De este modo, se consigue conservar la estabilidad.

$$m_d(n) = K_d \frac{de(t)}{dt} \Big|_{t=kT_s} = K_d \frac{e(kT_s) - e([k-1]T_s)}{T_s} \quad (4.4)$$

Por tanto, la señal de control en el instante n es la suma de las ecuaciones 4.2, 4.3 y 4.4. Para obtener una expresión más simplificada y fácil de implementar, se calcula la expresión de señal de control en $n-1$ y se hace la resta de la expresión en n con la de $n-1$. La expresión obtenida es la siguiente:

$$m(n) = m(n-1) + q_0 e(n) + q_1 e(n-1) + q_2 e(n-2) \quad (4.5)$$

donde los coeficientes q_0 , q_1 y q_2 tienen las siguientes expresiones:

$$q_0 = \left(K_p + \frac{K_d}{T_s} + \frac{K_i T_s}{2} \right) \quad q_1 = \left(-K_p - \frac{2K_d}{T_s} + \frac{K_i T_s}{2} \right) \quad q_2 = \frac{K_d}{T_s} \quad (4.6)$$

Para verificar que la discretización es válida, se ha simulado en MATLAB dicha discretización y se ha comparado con la respuesta de la figura 4.6. Dicha comparativa puede verse en figura 4.7. En dicha imagen se observa que la discretización se aproxima con bastante precisión a la respuesta continua.

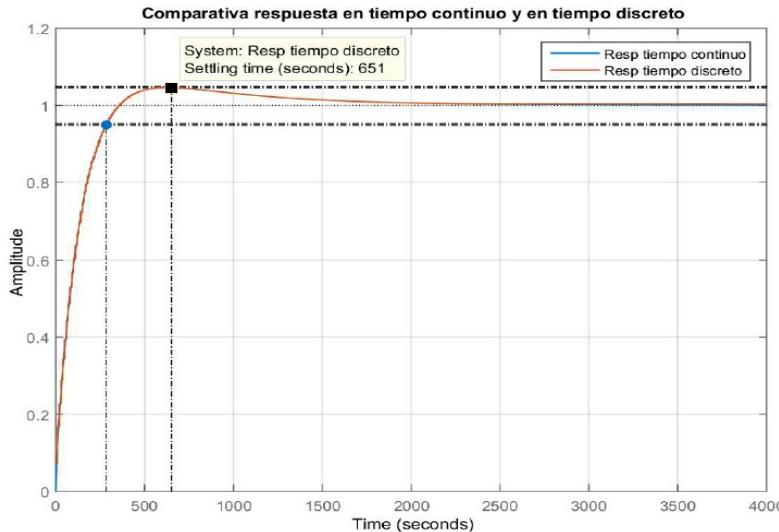


Figura 4.7: Respuesta del sistema a una entrada escalón con el PID discretizado

4.5. Bloque de generación de comandos

Este bloque se ha dividido en 2 subbloques para facilitar la implementación y modularidad. En la figura 4.8 se muestra un esquema de las entradas y salidas del bloque y los subbloques que lo componen.

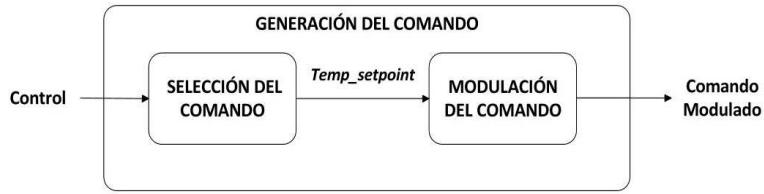


Figura 4.8: Esquema del bloque de generación de comandos

El subbloque de **selección del comando** selecciona el *setpoint*, a partir del valor de la señal de control y el rango de funcionamiento del sistema de refrigeración, y selecciona el comando asociado. El subbloque de **modulación del comando** modula ese comando para que pueda ser interpretado por el sistema de refrigeración.

4.5.1. Subbloque de selección del comando

Su función es seleccionar la temperatura del sistema de refrigeración a partir del valor de señal de control y teniendo en cuenta el rango de valores de funcionamiento. Dicho rango es un conjunto finito de valores discretos con una separación fija entre valores consecutivos. Teniendo en cuenta esto, se va a aplicar el siguiente criterio para seleccionar la temperatura:

- **Señal de control igual o superior a la temperatura máxima:** en este caso se escoge la temperatura máxima de funcionamiento.
- **Señal de control entre la temperatura mínima y máxima:** se escoge la temperatura obtenida del resultado de redondear el valor de la señal de control al entero más próximo.
- **Señal de control igual o inferior a la temperatura mínima:** en este caso se escoge la temperatura mínima de funcionamiento.

Hay que tener en cuenta que el valor de la señal de control está expresado en milésimas, por lo que es necesario convertirlo a unidades para poder seleccionar la temperatura de forma adecuada.

Después, una vez fijada la temperatura de funcionamiento, se selecciona el comando asociado a esa temperatura. Un comando es una secuencia de bits (0's y 1's) que representa un modo de funcionamiento que puede ejecutar el sistema de refrigeración.

Hace unos años, algunos miembros de *GreenLSI* realizaron un proyecto denominado Pimote [27]. Este proyecto consistía en el diseño e implementación de un mando a distancia universal. En ese trabajo se utilizó como banco de pruebas el sistema de refrigeración de la sala B039, que es el mismo que se utiliza en este trabajo, por lo

que están disponibles sus comandos de funcionamiento. Cada uno de estos comandos está almacenado en un fichero de texto y siguiendo un determinado protocolo.

Se ha decidido utilizar los comandos usados en el proyecto Pimote y el protocolo que utilizan. Dicho protocolo es sencillo y en él están perfectamente delimitados cada uno de los campos que componen el comando. La única modificación que se va a hacer en este trabajo es convertir cada fichero a formato binario para facilitar su uso en el sistema de actuación. En la figura 4.9 se muestra el esquema del protocolo.



Figura 4.9: Protocolo de almacenamiento del comando

A continuación se detallan cada uno de los campos del protocolo:

- **Bytes de relleno:** son 6 bytes y están reservados por si fuera necesario incluir un campo nuevo en la cabecera o ampliar el tamaño de un campo ya definido.
- **Longitud del Payload:** está formado por 1 byte e indica el número de bytes que ocupa el campo de datos del comando.
- **Período de bit:** está formado por 2 bytes y contiene el tiempo de bit del comando expresado en microsegundos.
- **Período de la portadora:** está formado por 1 byte y contiene el periodo de la portadora expresado en microsegundos.
- **Payload:** su tamaño varía según el comando seleccionado. Contiene la orden o modo de funcionamiento del aire acondicionado.

4.5.2. Subbloque de modulación del comando

Se función es modular el comando que se envía al aire acondicionado para que el receptor pueda interpretarlo y ejecutar la orden. En este trabajo se va a usar una modulación digital de amplitud usando como portadora una señal cuadrada de periodo $T_{carrier}$. En la figura 4.10 se muestra un esquema que representa la modulación de la secuencia de bits '0101'. A continuación se describe el proceso de modulación de cada bit:

- **Modulación de un '1':** se transmite una señal cuadrada de frecuencia igual a la portadora y durante un tiempo igual al periodo de bit .

- **Modulación de un ‘0’:** no se transmite ninguna señal durante un tiempo igual al periodo de bit.

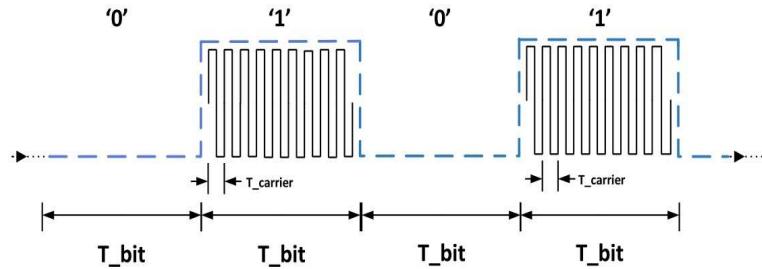


Figura 4.10: Ejemplo de modulación de una secuencia de bits 0101

4.6. Bloque de transmisión

Su función es enviar el comando al sistema de refrigeración. Para facilitar su implementación y modularidad, se ha dividido en 2 subbloques. En la figura 4.11 se muestra un esquema con sus entradas y salidas, así como de los subbloques que lo componen.

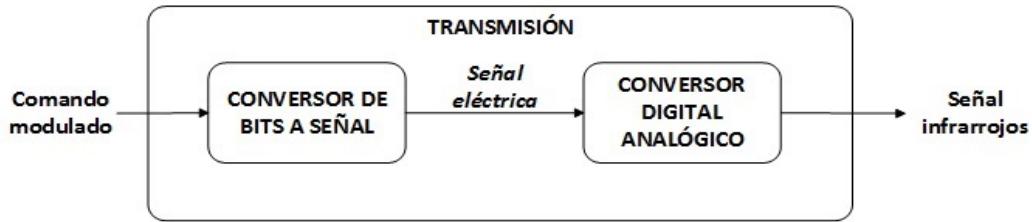


Figura 4.11: Diagrama del bloque de transmisión

Primero, el **conversor de bits a señal** genera una señal eléctrica a partir de la secuencia de bits del comando y posteriormente, el **el circuito emisor de infrarrojos** la convierte en una señal de infrarrojos para que el sistema de refrigeración pueda interpretarla. A continuación se explican ambos subbloques con más detalle.

4.6.1. Conversor de bits a señal

Convierte la secuencia de bits que representa el comando modulado, en una señal eléctrica. Dicha señal debe cumplir los requisitos de tiempo definidos en la modulación (periodo de bit y periodo de la portadora). La transmisión de cada bit debe hacerse de forma síncrona para controlar el tiempo de bit y el tiempo de portadora.

Para ello, se ha decidido utilizar el protocolo SPI *Serial Peripheral Interface* [28]. Es un protocolo de comunicación serie síncrono que permite la transferencia

de información entre un nodo principal llamado *nodo maestro* y uno o varios nodos secundarios llamados *nodos esclavos*. La transmisión de la información se realiza byte a byte, con un bit de parada entre transmisiones consecutivas. Un dispositivo SPI está formado por 4 pines:

- **MOSI:** pin que se utiliza para enviar información a los nodos esclavos.
- **MISO:** pin que se utiliza para recibir información de los nodos esclavos.
- **SCLK:** pin que emite una señal de reloj cuadrada, de frecuencia configurable, que se utiliza para sincronizar el envío de información entre los nodos.
- **CE:** pin que se utiliza para habilitar o deshabilitar la comunicación con los nodos esclavos.

La elección de este protocolo se debe a que se puede controlar el tiempo de duración de cada bit a través de la señal SCLK. Por tanto, se puede realizar la modulación correctamente y cumpliendo los requisitos de periodo de bit y periodo de portadora fijados en la etapa de modulación. La interfaz SPI emite una señal eléctrica por el pin MOSI cuando se desea enviar el bit ‘1’ y no emite nada cuando se desea enviar el bit ‘0’. El bit ‘1’ se asocia con un nivel de tensión alto (3,3 V o 5V), mientras que el bit ‘0’ se asocia con un nivel de tensión próximo a 0V.

4.6.2. Circuito emisor de infrarrojos

Convierte la señal eléctrica en una señal de infrarrojos y la emite al sistema de refrigeración. En la figura 4.12 puede verse el esquema del circuito.

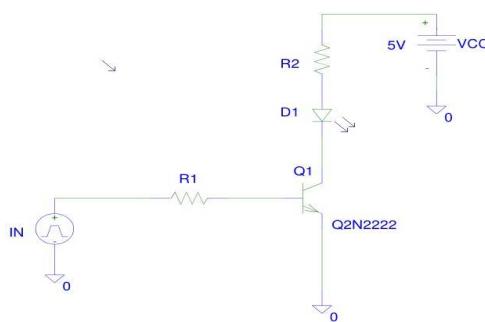


Figura 4.12: Esquema del circuito emisor de infrarrojos

El circuito se basa en un diodo LED de infrarrojos (IRLED), un transistor BJT y un conjunto de resistencias. La resistencia R1 se usa para obtener la corriente de base que permite al transistor entrar en la región de saturación y la resistencia R2 se usa para polarizar el diodo en su punto de trabajo. El transistor trabaja en saturación y corte para que el diodo pueda generar la señal a transmitir.

El funcionamiento del transistor es el siguiente: cuando no se recibe tensión en la entrada del circuito (equivale a un '0'), el transistor entra en corte y la corriente de base y colector son muy próximas a 0. Al no haber corriente de colector, no circula corriente por el diodo, luego no emite señal. Sin embargo, cuando se recibe un nivel de tensión en la entrada (equivale a un '1'), el transistor entra en la región de saturación y circula corriente a través del diodo. El diodo convierte esa corriente en una señal de infrarrojos y la emite al sistema de refrigeración.

4.7. Bloque de envío de datos

La función de este bloque es subir datos a la plataforma *Graphite*. En la figura 4.13 se muestra un esquema de sus entradas y salidas.

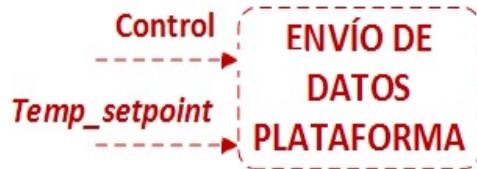


Figura 4.13: Esquema del bloque de envío de datos

El bloque posee dos entradas, el dato de temperatura de control y el dato de *setpoint*. Dichos valores se suben a la plataforma una vez se ha obtenido la temperatura de control y se ha seleccionado el *setpoint*. El objetivo es poder visualizar el comportamiento de estas señales y así poder evaluar el comportamiento del actuador.

Aunque no aparezca en el diagrama, en la fase de implementación se considerará la *temperatura óptima* como otra entrada. El motivo es que en este trabajo la *temperatura óptima* va a estar fijada por el usuario y es necesario subir dicho valor a la plataforma. Una vez el actuador esté integrado con el sistema de optimización del CPD, dicho valor será leído desde la plataforma y no será necesario subirlo.

Para poder subir los datos a la plataforma se ha tenido en cuenta los diferentes métodos disponibles para subir un dato, según la documentación de *Graphite* [26]. Se ha usado un formato de texto plano para subir los datos, ya que presenta un formato sencillo y es fácil de implementar. Dicho formato presenta la estructura:

```
<metric_path> <metric_value> <metric_timestamp>
```

El parámetro *<metric_path>* debe contener la dirección http y el puerto del servidor, junto con la ruta donde se va a escribir el dato. El parámetro *<metric_value>* debe contener el dato que se quiere subir a la plataforma. Por último, el parámetro *<metric_timestamp>* debe contener el instante de tiempo en el que se midió el dato.

Capítulo 5

Implementación

En este capítulo se va a detallar la implementación, tanto a nivel hardware como software, del sistema de actuación y de cada uno de los bloques que lo componen.

El sistema de actuación se ha implementado en una Raspberry PI 2 model B, ya que dispone de un buen procesador, un puerto ethernet para conexión a internet, una buena memoria y varios puertos GPIO e interfaces, incluido SPI. Cada uno de los bloques del sistema de actuación está implementado mediante un programa software que estará alojado en la Raspberry PI, a excepción del circuito de infrarrojos cuya implementación es hardware. En la figura 5.1 se muestra la Raspberry PI utilizada.



Figura 5.1: Raspberry PI 2 model B

El programa software está implementado en C ya que es un lenguaje de programación que posee características propias de los lenguajes de alto nivel y también permite manejar y gestionar la memoria y los puertos GPIO de la Raspberry PI de una forma sencilla.

Cada bloque esta formado por un fichero .c y un fichero .h. El fichero .c contiene el código fuente de las funciones públicas y estáticas, así como las dependencias con otras librerías. El fichero .h contiene las macros y estructuras definidas, así como el prototipo de las funciones públicas implementadas en el fichero .c. En los siguientes apartados se detalla la implementación de cada bloque.

5.1. Bloque de adquisición

La implementación de este bloque se encuentra en los ficheros **platformDown.c** y **platformDown.h**. Para facilitar dicha implementación, se han usado las librerías externas *curl* [29] y *jansson* [30] para realizar las peticiones http y manejar los objetos en formato JSON respectivamente. A continuación se muestra la documentación de este modulo.

Macros definidas

```
#define FACT_CONV_TEMP 1000 // Factor de conversión a milésimas.

#define PET_HTTP_OK 200 // Petición http satisfactoria.

#define CONEXION_OK 0 // Conexión satisfactoria con la url .

#define SIZE_MEM (5*1024) // Tam máx del buffer para el objeto JSON.

// Formato de la URL del objeto JSON.
#define URL_FORMAT "http://visualizee.die.upm.es:8000/render?
format=json&target=visualizee.greencpd.b039.rack.r0001.server.
host2.temperature.ambient.1.1000&from=-1min"
```

Estructuras Definidas

struct objJSON

```
// Estructura donde se almacena el objeto JSON
// descargado de la plataforma.

struct objJSON{
    char datos[SIZE_MEM]; // Array reservado para almacenar el objeto.
    int size; // Tamaño en bytes del objeto (Número de posiciones usadas).
};
```

struct temp_leida

```
// Estructura donde se almacena el dato de temperatura solicitado
// y que ha sido extraído previamente de un objeto JSON.

struct temp_leida{
    int valor; // valor de la temperatura
    int timestamp; // instante de medida de la temperatura
};
```

Funciones estáticas definidas

`escribir_objeto()`

```
static size_t escribir_objeto (void * ptr, size_t size, size_t
nmemb, struct objJSON* obj)

// Funcion estatica que escribe el objeto descargado
// en una estructura objJSON creada previamente para almacenarlo.

@param void* ptr // Puntero al objeto JSON que se quiere escribir.

@param size_t size // Posiciones de memoria que ocupa el obj JSON.

@param size_t nmemb // Bytes que ocupa una pos de memoria del obj.

@param struct objJSON* obj // Puntero a la estr que guardará el obj.

@return size_t // Devuelve el nºbytes copiados. Si hay error, da -1.
```

Funciones públicas definidas

`solicitar_objeto()`

```
void solicitar_objeto (const char * url, struct objJSON* obj);

// Función que realiza la petición http para obtener objeto JSON y copia
// el objeto en la estructura objJSON. Usa la librería curl para la pet http
// y la función estática escribir_objeto() para copiar el objeto

@param const char* url // Puntero a un array con la URL del obj JSON.

@param struct objJSON* obj // Puntero a la estr que guardará el obj.

@return void // No devuelve nada.
```

A continuación se incluye su pseudocódigo para facilitar su compresión.

```
static size_t escribir_objeto (void *ptr, size_t size, size_t
nmemb, struct objJSON * obj)
{
    // Declaracion de variables.
    curl_global_init (CURL_GLOBAL_ALL); // Inicializar librería.
    curl = curl_easy_init(); // Inicializa y crea el curl.
    if (Error) {
        // Error: curl no inicializado.
    }
    // Configuración del curl.
    curl_easy_setopt(curl, CURLOPT_URL, url);
```

```

curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, escribir_objeto);
curl_easy_setopt(curl, CURLOPT_WRITEDATA, obj->datos);

// Obtención del estado del curl y resultado de la petición.
estado = curl_easy_perform(curl);
curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &codigo);
if (estado == OK & codigo == PET_HTTP_OK) {
    // Todo correcto.
}
else{
    curl_easy_strerror(estado);    // Informar del error.
}
curl_easy_cleanup(curl);    // Cierre del curl.
curl_global_cleanup();    // Cierre de la librería.
}

```

extraer_temperatura()

```

int extraer_temperatura (struct objJSON* obj, struct temp_leida* temperatura)

// Función que procesa el objeto JSON y le extrae el dato de temperatura.
// Utiliza la librería jansson para extraer el dato del objeto JSON.

@param struct objJSON* obj // Puntero a la estr que contiene el obj.

@param struct temp_leida* temperatura
// Puntero a la estructura donde se va a escribir el dato extraído.

```

A continuación se incluye su pseudocódigo para facilitar su compresión.

```

int extraer_temperatura (struct objJSON* obj, struct temp_leida* temperatura)
{
    // Declaración de manejadores del objeto JSON, campos y errores.
    json_t* obj_handler, datapoints, valor, inst_medida;
    json_error_t errores;

    obj_handler = json_loads(); // Carga del objeto en el manejador.
    if (!obj_handler) {
        // Error: No se puede cargar;
        return -1;
    }
    // Extracción del datapoints.
    if (error) {
        // Error: No existe el datapoints
        return -1;
    }
    // Extracción del dato mas reciente.
    for(i=0; i< json_array_size(obj_handler); i++) {

```

```

    // Extracción del datapoint i;
    // Extracción de los atributos valor e inst_medida;
    if (OK) {
        // Copia del dato en la estructura temperatura;
    }
}
return 0;
}

```

5.2. Bloque de comparación

La implementación del bloque de comparación se encuentra en los ficheros **comparador.c** y **comparador.h**. Este módulo sólo dispone de la función *calcular_error()*. A continuación se incluye su documentación.

calcular_error()

```

int calcular_error (int temp_ref, int temp_med)
// Calcula la diferencia de la señal de referencia y la señal medida.

@param int temp_ref // Temperatura de referencia.

@param int temp_med // Temperatura medida.

@return int // Entero con el valor de la señal de error.

```

5.3. Bloque de control

La implementación de este bloque se encuentra en los ficheros **PID.c** y **PID.h**. A continuación se incluye la documentación de este módulo.

Macros definidas

```

#define KP 28000 // Constante proporcional expresado en milésimas.

#define KI 37 // Constante integral expresado en milésimas.

#define KD 300000 // Constante derivativa expresado en milésimas.

#define PERIOD 10000 // Periodo de muestreo expresado en milésimas.

#define FACT_CONV 1000 // Factor de conversion de unidades a milésimas.

```

Funciones definidas

```
calcular_tempcontrol()

int calcular_tempcontrol (int err, int err_1, int err_2, int control_1).

// Esta función implementa el algoritmo de un controlador PID discreto,
// y genera el valor de la señal de control en cada instante.

@param int err // Valor de la señal de error en n.
@param int err_1 // Valor de la señal de error en n-1.
@param int err_2 // Valor de la señal de error en n-2.
@param int control_1 // Valor de la señal de control en n-1.
@return int // Valor de la señal de control en n.
```

La implementación del PID se basa en la expresión matemática del controlador PID discreto que aparece en la sección 4.4.3. El proceso consiste en ir calculando resultados parciales de esa fórmula y sumarlos todos para obtener el resultado final. El orden de operación es el siguiente:

1. Se calculan los productos y divisiones que conforman los coeficientes q_0 , q_1 y q_2 .
2. Se calculan los coeficientes q_0 , q_1 y q_2 .
3. Se calculan los productos de cada coeficiente con los valores de la señal de error.
4. Se suman los resultados de esos productos con el valor de la señal de control en el instante $n - 1$, obteniendo el valor de la señal de control $m(n)$.

Esta función maneja números reales con una precisión de milésimas pero están expresados como números enteros. En el caso de las sumas y las restas no existen problemas de desbordamiento porque los números no son muy grandes. Sin embargo, en el caso de la multiplicación y división sí se manejan números muy grandes por lo que puede darse el caso de que al realizar este tipo de operaciones, se produzca desbordamiento u *overflow* y el resultado generado sea incorrecto. Para solucionar este problema, se ha decidido implementar la multiplicación y división del siguiente modo:

Implementación de la multiplicación

Supongamos 2 números reales A y B cada uno con una precisión de milésimas. A y B pueden ser expresados como la suma de su parte entera más su parte decimal.

$$A = [A_2 A_1 A_0, a_1 a_2 a_3] = \left(A_2 A_1 A_0 + \frac{a_1 a_2 a_3}{1000} \right)$$

$$B = [B_2 B_1 B_0, b_1 b_2 b_3] = \left(B_2 B_1 B_0 + \frac{b_1 b_2 b_3}{1000} \right)$$

Por tanto, el producto de A y B ($A * B$) se obtiene realizando los subproductos de cada término de A por cada término de B. Dichos subproductos son:

$$S_{p1} = A_2 A_1 A_0 * B_2 B_1 B_0 \rightarrow S_{p1}(\text{milésimas}) = S_{p1} * 1000$$

$$S_{p2} = A_2 A_1 A_0 * \left(\frac{b_2 b_1 b_0}{1000} \right) \rightarrow S_{p2}(\text{milésimas}) = A_2 A_1 A_0 * b_2 b_1 b_0$$

$$S_{p3} = \left(\frac{a_2 a_1 a_0}{1000} \right) * B_2 B_1 B_0 \rightarrow S_{p3}(\text{milésimas}) = a_2 a_1 a_0 * B_2 B_1 B_0$$

$$S_{p4} = \left(\frac{a_2 a_1 a_0}{1000} \right) * \left(\frac{b_2 b_1 b_0}{1000} \right) \rightarrow S_{p4}(\text{milésimas}) = \left(\frac{a_2 a_1 a_0 * b_2 b_1 b_0}{1000} \right)$$

Por tanto el resultado final es sumar cada uno de los subproductos expresado milésimas. De esta forma se evita el desbordamiento ya que la mayoría de los subproductos generan el resultado directamente en milésimas sin necesidad de aplicar factores de conversión. Aquellos que no lo generan se les aplica el factor una vez se ha realizado el subproducto, lo que evita el manejo de números muy grandes.

Implementación de la división

Supongamos 2 números A y B cada uno con una precisión de milésimas. No se puede hacer la división con ambos números en las mismas unidades porque se pierde la parte decimal o sería necesario el uso de variables reales. Por tanto, es necesario multiplicar el dividendo (en este caso A) por el factor de conversión de unidades a milésimas y dividir el resultado por el divisor (en este caso B). De este modo, el resultado obtenido es el mismo que si se hubiera hecho la división con números reales y truncado el resultado a milésimas. Además, se puede hacer la operación usando números enteros. A continuación se muestra un ejemplo de este proceso:

$$A = 13,135; \quad B = 4,328; \quad \rightarrow A/B = 3,0348 \sim 3,034 \text{ unidades} \rightarrow 3034 \text{ milésimas}$$

$$A = 13,135 * 1000; \quad B = 4,328; \quad \rightarrow A/B = 3034,8 \sim 3034 \text{ milésimas}$$

Hay que señalar que aunque este proceso se ha aplicado para números reales con una precisión de milésimas, es también aplicable para números reales con otra precisión aunque hay que tener en cuenta el tamaño de las variables que se estén usando.

5.4. Bloque de generación de comandos

La implementación de este bloque y de los subbloques que lo componen se encuentra en los ficheros **comandos.c** y **comandos.h**. La implementación de cada subbloque se ha realizado en diferentes funciones para facilitar su modularidad. A continuación se incluye la documentación de este módulo.

Macros definidas

```
#define FACT_CONV 1000 // Factor de conversión de unidades a milésimas.

#define TEMP_MAX_AC 32 // Temperatura máxima del aire acondicionado.

#define TEMP_MIN_AC 18 // Temperatura mínima del aire acondicionado.

#define REDONDEO_MILESIMAS 500 // Valor para redondear las milésimas.

// Formato de la ruta de acceso a los ficheros que contienen los comandos.
#define FILE_PATH_FORMAT "ficheros_tramas/tramas_zulo/%dzulo.txt
.bin"

// Tamaño máximo permitido de la ruta de acceso a los ficheros.
#define FILE_PATH_SIZE 50

#define BIT_MASK 0x80 //Máscara utilizada para manejar bits.

#define FACT_SEG_USEG 1000000 //Factor de conversion de seg a microseg.

#define BYTE1_DATOS 10 //1º byte de datos del comando.

// Byte del comando que contiene el número de bytes de datos.
#define BYTE_SIZE_CMD 6

#define MSB_TBIT 7 // Byte más significativo del periodo de bit.

#define LSB_TBIT 8 // Byte menos significativo del periodo de bit.

// Byte del comando que contiene el periodo de la portadora
#define BYTE_TCARRIER 9

// Número de bits de una transferencia spi, incluyendo el bit de parada.
#define NBITS_SPI 9

#define NBITS_BYTE 8 // Byte menos significativo del periodo de bit.

// Tamaño del array donde se almacena el comando sin modular.
#define TAM_CMD (2 x 1024) // 2KB

// Tamaño del array donde se almacena el comando sin modular.
#define TAM_CMDDMOD (10 x 1024) // 10KB
```

Estructuras definidas

struct comando

```
// struct comando. Estructura donde se va a almacenar
// el comando que hay que enviar pero sin estar modulado.

struct comando {
    unsigned char datos [TAM_CMD]; //Array donde se guarda el comando.
    long pos; // Posición del array que se está usando.
};
```

struct cmd_modulado

```
// struct cmd_modulado. Estructura que contiene
// el comando ya modulado.

struct cmd_modulado {
    unsigned char datos [TAM_CMDDMOD]; // Array donde se almacena
                                         // el comando modulado.
    long pos; // Posición del array que se está usando.
    int f_bit; // Frecuencia de bit usada para enviar el comando por SPI.
};
```

Funciones definidas

Cada subbloque se ha implementado en diferentes funciones para facilitar su modularidad. A continuación se documentan las funciones de cada subbloque.

5.4.1. Subbloque de selección del comando

La funcionalidad de este bloque se ha implementado mediante las funciones *seleccionar_setpoint()* y *obtener_comando()*. A continuación se documentan dichas funciones.

seleccionar_setpoint()

```
int seleccionar_setpoint (int temp_control)

// Esta función determina la temperatura del aire acondicionado
// a partir de la señal de control generada por el PID.

@param int temp_control // Valor de la señal de control en milésimas.

@return int // Devuelve un entero con el setpoint a fijar.
```

obtener_comando()

```
int obtener_comando (int temp_setpoint, struct comando* cmd)
// Esta función busca el fichero asociado al setpoint pasado como
// argumento y copia su contenido en un estructura de tipo comando.

@param int temp_setpoint // Valor del comando a seleccionar.
@param struct comando* cmd // Estructura que almacenará el comando.
@return int // Devuelve 0 si todo ha ido bien. Si hay error, da -1.
```

5.4.2. Subbloque de modulación del comando

Este subbloque se ha implementado con la función estática *modular_bit()* y la función *modular_comando()*. A continuación se documentan ambas funciones.

modular_bit()

```
static void modular_bit (int value, int pos_init, int
size_bit_mod, struct cmd_modulado* cmd)

// Función estática que realiza la modulación de un bit de un comando.

@param int value // Valor del bit que se quiere modular.
@param int pos_init // Posición (en una trans SPI) del 1º bit.
@param int size_bit_mod // Tamaño de la secuencia sin bits de parada.
@param struct cmd_modulado* cmd_mod // Puntero hacia el cmd mod.
@return void // No devuelve nada.
```

modular_comando()

```
int modular_comando(struct comando* cmd, struct cmd_modulado*
cmd_mod)

// Función que modula el comando a transmitir.

@param struct comando* cmd // Puntero hacia el comando sin modular.
@param struct cmd_modulado* cmd_mod // Puntero hacia el cmd mod.
@return int // Devuelve 0 si todo ha ido bien. En caso de error, da -1.
```

Como se ha mencionado en el apartado 4.5.2, la modulación se realiza mediante la transmisión o no de una señal cuadrada de periodo $T_{carrier}$ y con una duración igual al tiempo de bit, dependiendo de si se envía un '1' o un '0'. También se ha mencionado en la sección 4.6.1 que se va a utilizar la interfaz SPI disponible en la raspberry PI para transmitir el comando al circuito emisor de infrarrojos. La modulación de cada bit se implementa del siguiente modo:

1. **Modulación de un '0':** se implementa mediante una secuencia todo '0's.
2. **Modulación de un '1':** se implementa mediante una secuencia de '1's y '0's alternados.

Ambas secuencias poseen la misma longitud y es el resultado de dividir el doble del periodo de bit por el periodo de la portadora ($T_{carrier}$). El motivo de dividir por el doble del periodo de bit es para conseguir la frecuencia deseada.

Un factor que hay que considerar en la implementación de la modulación es que la interfaz SPI sólo permite transferencias byte a byte y que por cada byte transmitido, introduce un bit de parada. Por tanto, estos bits de parada deben ser considerados en la secuencia de modulación para que el comando modulado pueda ser interpretado por el aire acondicionado. Si no son considerados estos bits de parada, la modulación no se realiza correctamente y el aire acondicionado no podrá interpretar la orden.

Por tanto, por cada 9 bits de la secuencia de modulación, debe ser descontado un bit de dicha secuencia, que será sustituido por el bit de parada. Este proceso se irá realizando de forma simultánea al proceso de generación del comando modulado. Sin embargo, este proceso depende de la posición de inicio de la secuencia de modulación de cada bit, ya que puede incluir más o menos bits de parada.

Por tanto, el proceso de modulación consta de los siguientes pasos:

1. Se extrae de la cabecera el periodo de bit y el periodo de la portadora y se calcula la longitud de la secuencia de modulación de cada bit.
2. Se calcula la longitud total de la secuencia de modulación del comando y los bits de parada que van a ser usado en dicha secuencia.
3. Se realiza la modulación de cada bit. Para ello, se tiene en cuenta el valor del bit, la posición de inicio de dicha secuencia dentro de una transferencia SPI, su posición final y el número de bits de parada que se van a usar en la modulación de ese bit.
4. Por último, una vez se ha realizado la modulación del comando, se comprueba que el número de bytes que ocupa dicha secuencia coincide con el número de bytes esperados y si es así devuelve un 0, indicando que la modulación se ha realizado correctamente. En caso contrario, devuelve -1.

5.5. Bloque de transmisión

La implementación de este bloque es hardware y software. El conversor de bits a señal posee una implementación software, mientras que el circuito emisor de infrarrojos posee una implementación hardware. A continuación se detalla la implementación de ambos subbloques.

5.5.1. Conversor de bits a señal

La implementación de este subbloque se encuentra en los ficheros **spi.c** y **spi.h**. A continuación se incluye la documentación de este módulo.

Macros definidas

```
#define PORT_SCLK 11 // Puerto SCLK en Raspberry PI, según esquema GPIO.
#define PORT_MOSI 10 // Puerto MOSI en Raspberry PI, según esquema GPIO.
#define PORT_MISO 9 // Puerto MISO en Raspberry PI, según esquema GPIO.
#define PORT_CE 8 // Puerto CE en Raspberry PI, según esquema GPIO.
#define OUTPUT 1 // Valor para configurar un puerto como salida.
#define INPUT 0 // Valor para configurar un puerto como entrada.
#define OK 0 // El puerto es configurado como salida.
```

Funciones definidas

transmitir_comando()

```
int transmitir_comando (struct cmd_modulado* com)
// Función que realiza la transmisión del comando por la interfaz SPI.
@param struct cmd_modulado* mod. // Puntero al comando a transmitir.
@return int // Devuelve 0 si todo ha ido bien. En caso de error, da -1.
```

La función inicializa la librería pigpio y configura los puertos de la interfaz SPI. Después, crea un manejador SPI con el que realizar la transmisión y configura dicho dispositivo, incluyendo la frecuencia de bit. Por último, activa dicho dispositivo y transmite cada uno de los bytes del comando modulado.

5.5.2. Circuito emisor de infrarrojos

Este circuito se ha implementado en una placa de inserción y se ha utilizado un diodo LED de infrarrojos LD271, un transistor BJT npn y 2 resistencias. Las características del circuito son las siguientes:

Tensión de alimentación: $V_{CC} = 5V$.

Tensión y corriente del diodo: $V_D = 1,3V$ y $I_D = 130mA$.

BJT en corte: $I_C = I_B = 0$. Condición: $V_{BE} < V_{\gamma E}$.

BJT en saturación: $V_{BE} = V_{\gamma E}$, $V_{CE} = V_{CE(SAT)}$. Condiciones: $I_C < \beta I_B$, $I_B > 0$.

Parámetros transistor: $\beta = 100$, $V_{CE(SAT)} = 0,2V$ y $V_{\gamma E} = 0,7V$.

Señal de entrada: es una señal digital de valores $V_{HIGH} = 3,3V$ y $V_{LOW} = 0V$.

A continuación se muestra el análisis del circuito suponiendo que el transistor está en saturación. No es necesario hacerlo en corte porque en ese estado no circula corriente por el circuito y se alcanzaría dicho estado con cualquier valor de R1 y R2.

$$V_{R2} = V_{CC} - V_D - V_{CE(SAT)} = 5 - 1,3 - 0,2 = 3,5V$$

$$R_2 = \frac{V_{R2}}{I_D} = 26,92\Omega$$

$$I_B > \frac{I_C}{\beta} = 1,3mA \quad \text{Se elige} \quad I_B = 16mA$$

$$R_1 = \frac{V_{IN} - V_{BE(SAT)}}{I_B} = \frac{3,3 - 0,7}{0,016} = 162,5\Omega$$

Por tanto, los valores de los componentes son: $R_1 = 160\Omega \pm 5\%$ y $R_2 = 27\Omega \pm 5\%$

Por último, en la figura 5.2 se muestra el circuito de infrarrojos implementado. Hay que indicar que la resistencia R_1 se ha hecho conectando varias resistencias en serie que equivalen a su valor.

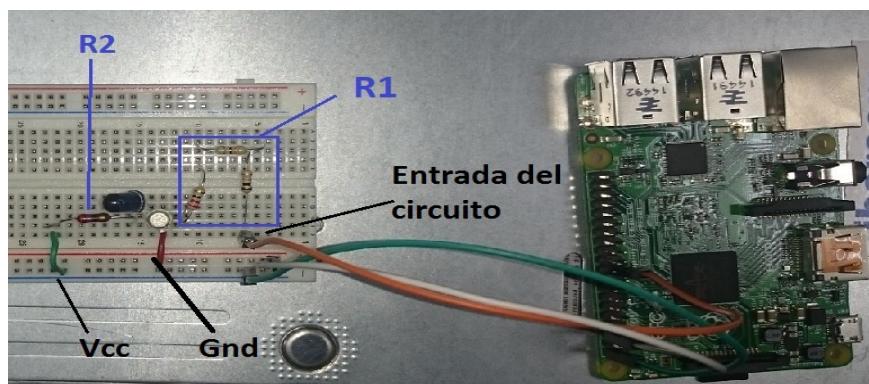


Figura 5.2: Imagen del circuito de infrarrojos

5.6. Bloque de envío de datos

La implementación de este bloque se encuentra en los ficheros **platformUp.c** y **platformUp.h**. Este módulo sólo contiene la función *enviar_datos()*. A continuación se incluye su documentación.

```
void enviar_datos(int temp_control, int temp_setpoint, int
temp_optimized, int timestamp).

// Función que se encarga de subir datos a Graphite.

@param int temp_control // Temperatura de control(señal de control).
@param int temp_setpoint // Temperatura del aire acondicionado.
@param int temp_optimized // Temperatura óptima.
@param int timestamp // Instante de lectura de los datos.
@return void // No devuelve nada.
```

5.7. Ciclo de ejecución

La implementación del ciclo de ejecución está en el fichero **actuador.c**. En este fichero se encuentra la función *main()* que se encarga de realizar el ciclo del actuador y coordinar cada una de las etapas. A continuación se muestra el pseudocódigo de dicha función.

```
int main(){

    // Definición e inicialización de variables
    while(1)
    {
        // Obtención del instante de inicio.
        //Limpieza de estructuras.
        // obtención de la temperatura de referencia.
        solicitar_objeto (URL_FORMAT, &obj);
        flag = extraer_temperatura (&obj, &temp_ref);
        if (flag) {
            //Error: No se ha podido leer la temperatura.
            return -1;
        }

        // obtención de la temperatura de medida.
        solicitar_objeto (URL_FORMAT, &obj);
        flag = extraer_temperatura (&obj, &temp_med);
```

```
if (flag) {
    //Error: No se ha podido leer la temperatura.
    return -1;
}

// Cálculo de la señal de control
error = calcular_error (temp_ref.valor, temp_med.valor);
control = calcular_tempcontrol (error, error_1, error_2,
                                control_1);

// Selección del comando y envío de datos a la plataforma.
temp_setpoint = seleccionar_setpoint (control);
enviar_datos(temp_setpoint, control, temp_ref.valor,
             current_iteration->tv.sec);

// Obtención del comando y modulación de éste.
flag = obtener_comando (temp_setpoint, &cmd_setpoint);
if (flag) {
    // Error: No se ha podido obtener el comando.
    return -1;
}

flag = modular_comando (&cmd_setpoint, &cmd_mod_setpoint);
if (flag) {
    // Error: No se ha podido modular el comando.
    return -1;
}

//Transmisión del comando por la interfaz SPI.
flag = transmitir_comando (&cmd_mod_setpoint);
if (flag) {
    // Error: No se ha podido transmitir el comando.
    return -1;
}

// Calcular intervalo de la siguiente iteración.
iteracion(&current_iteration, &next_iteration, &timeout)
;
}

return 0;
}
```


Capítulo 6

Test y resultados

En este capítulo se detallan las pruebas realizadas para verificar el correcto funcionamiento del sistema de actuación. Estas pruebas se han llevado a cabo en el banco de pruebas descrito en el capítulo 3, ya que es un entorno real y adecuado para probar el funcionamiento del actuador.

Las pruebas están orientadas a verificar el funcionamiento del actuador y comprobar que es capaz de controlar la temperatura de la sala y ajustarla al valor óptimo. El intervalo de temperaturas usado para hacer las pruebas es [18°C - 24°C], ya que este intervalo garantiza un correcto funcionamiento de la sala. Para realizar las pruebas, no se ha utilizado el dato proporcionado por el algoritmo de optimización, sino que dicho valor está fijado de forma manual. El objetivo es comprobar que el sistema consigue fijar la temperatura de la sala al valor óptimo y comprobar que funciona correctamente. Una vez se compruebe su funcionamiento, se procederá a su integración en el sistema de optimización.

El sensor de la sala tiene una precisión de $\pm 0,5^\circ\text{C}$, luego éste es el mínimo ajuste de temperatura que puede hacer el actuador. Por otro lado, hay que tener en cuenta que el actuador debe poder realizar ajustes de temperatura pequeños (de $\pm 0,5^\circ\text{C}$ o $\pm 1,0^\circ\text{C}$) y grandes ($\pm 3^\circ\text{C}$ o superior).

Por tanto, para comprobar que el actuador es capaz de controlar la temperatura y realizar ajustes tanto pequeños como grandes, se han hecho 4 tipos de pruebas:

1. Subir directamente la temperatura de la sala en al menos 3°C .
2. Bajar directamente la temperatura de la sala en al menos 3°C .
3. Subir la temperatura de la sala con incrementos de 0.5°C .
4. Bajar la temperatura de la sala con decrementos de 0.5°C .

Hay que aclarar que con estas pruebas se pretende verificar el funcionamiento en las situaciones más complejas. Se supone que si el sistema funciona correctamente en los casos extremos, también funcionará correctamente en los casos intermedios. A continuación se muestra la tabla 6 con las características de cada prueba realizada.

Por claridad, este capítulo sólo se incluye una prueba de cada tipo. En el anexo 8.2 se incluyen otras pruebas realizadas para comprobar el funcionamiento del actuador. Las conclusiones obtenidas de todos los experimentos (incluidos los del anexo), se muestran en este capítulo.

Num Exp	Tipo exp	Experimento	Parámetros PID
1	1	Subida de 20°C a 24°C	$K_p = 28$ $K_i = 0,037$ $K_d = 300$
2	3	Subida de 20°C a 24°en pasos de 0.5°C	$K_p = 28$ $K_i = 0,037$ $K_d = 300$
3	2	Bajada de 24°C a 21°C	$K_p = 28$ $K_i = 0,037$ $K_d = 300$
4	4	Bajada de 24°C a 22°C en pasos de 0.5°C	$K_p = 28$ $K_i = 0,037$ $K_d = 300$

Tabla 6.1: Características de los experimentos realizados

6.1. Resultados de las pruebas

A continuación se muestran unas capturas de *Graphite* que muestran los resultados obtenidos de los 4 primeros experimentos. En cada captura se representa la temperatura a la que se encuentra la sala (azul), el *setpoint* (verde), y la temperatura óptima (rojo).

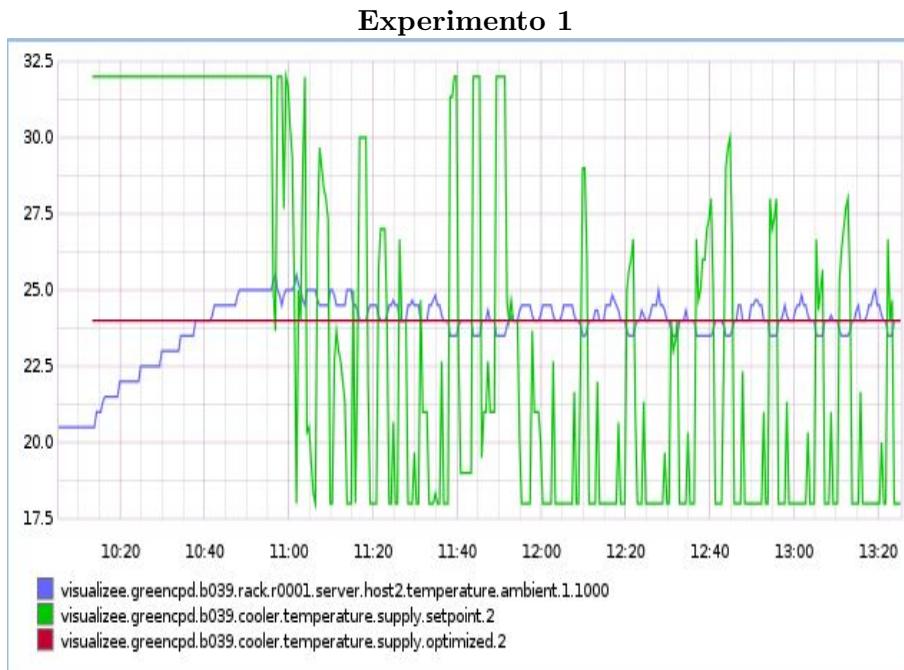


Figura 6.1: Gráfica con los resultados del experimento 1

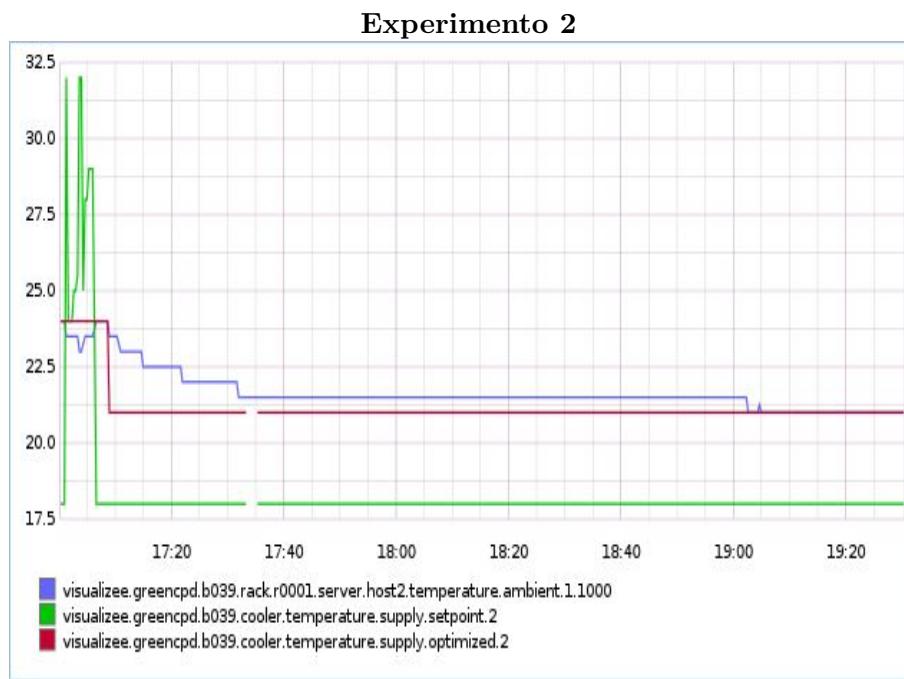


Figura 6.2: Gráfica con los resultados del experimento 2

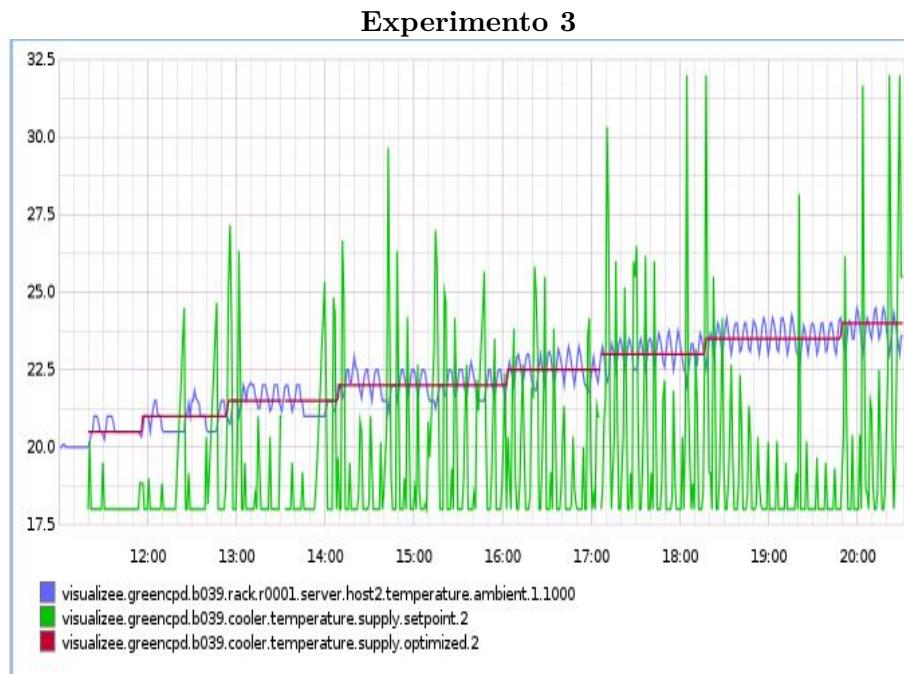


Figura 6.3: Gráfica con los resultados del experimento 3

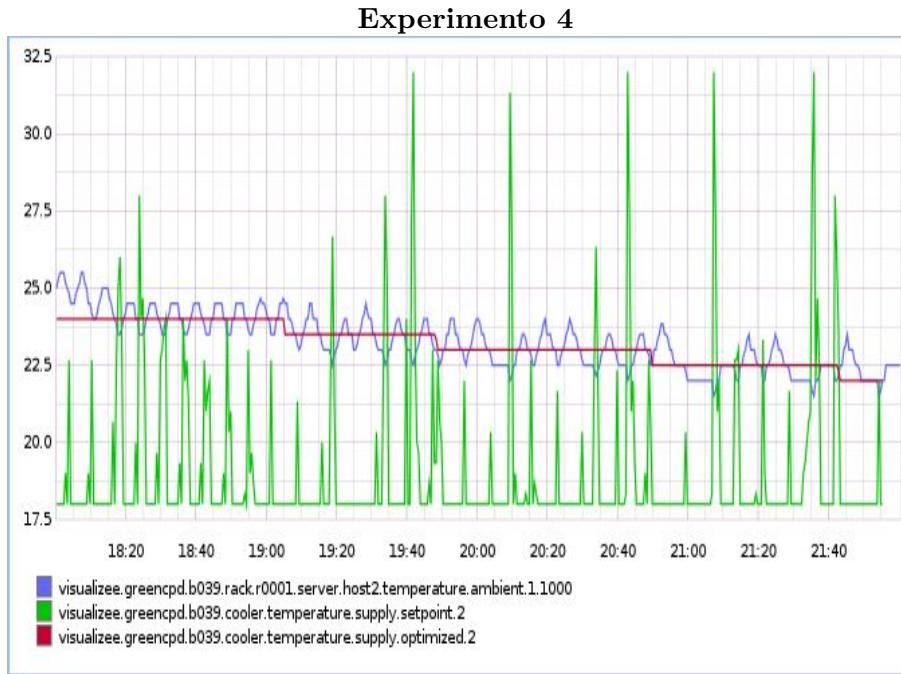


Figura 6.4: Gráfica con los resultados del experimento 4

Hay que aclarar que en ninguna de las gráficas se ha incluido la señal de control. El motivo es que el controlador PID diseñado no tiene ningún mecanismo antiwindup, lo que hace que la señal de control en ocasiones tome valores demasiado grandes o pequeños que impiden visualizar correctamente el resto de señales. Se ha intentado añadir un mecanismo antiwindup al controlador pero no se han obtenido los resultados esperados. Por tanto, se ha decidido no incluir dicho mecanismo en este trabajo y se plantea la inclusión de este mecanismo como una linea futura.

6.2. Análisis de los resultados

Observando las gráficas de cada uno de los experimentos se comprueba que el actuador funciona correctamente y es capaz de controlar la temperatura del sistema y ajustarla al valor óptimo. Además, el actuador es capaz de ajustar la temperatura directamente o de forma gradual, lo que permite realizar ajustes de temperatura tanto pequeños como grandes.

En todos los experimentos se observa un rizado en torno al valor óptimo. Este rizado es principalmente causado por la precisión del sensor. En principio, no supone un inconveniente a la hora de conseguir alcanzar la temperatura óptima, ya que suele ser pequeño y en la mayoría de las ocasiones coincide con la precisión del sensor.

Hay que destacar el efecto windup que se ha producido en los experimentos 1, 5, 6, 8 y 11. El sistema de refrigeración dispone de un valor máximo y mínimo

de funcionamiento y una vez que se alcanzan esos límites se satura. Sin embargo, el controlador PID diseñado es lineal por lo que el término integral sigue aumentando y creciendo a pesar de que el sistema de refrigeración esté saturado. Cuando se supera la temperatura óptima, el error cambia de signo y el término integral comienza a compensarse aunque tardará un cierto tiempo como consecuencia del error acumulado en la etapa de subida. Esto provoca que se produzca una sobreoscilación, como se observa al inicio de los experimentos mencionados.

Este efecto es habitual que ocurra cuando se produce variaciones muy grandes de temperatura que se aproximan a la longitud del intervalo. Esta situación es poco probable que ocurra en un CPD, ya que es más habitual que el ajuste de temperatura sea gradual que ir moviéndose de extremo a extremo. A pesar de este efecto, se observa que el sobreajuste no es muy elevado y en poco tiempo la temperatura se estabiliza en torno a su valor óptimo. Como ya se ha comentado, una posible mejora de este trabajo sería la implementación de un controlador PID con mecanismo antiwindup para corregir este problema.

También se puede observar cómo influye la temperatura del exterior en la sala. En los experimento 2 y 7 se observa que la temperatura decae rápidamente pero a partir de un cierto valor, la temperatura tarda más tiempo en disminuir. Este efecto es provocado debido a la influencia de la temperatura exterior. El ciclo de evaporación se realiza en el exterior del edificio y si la temperatura exterior es elevada la eficiencia del ciclo de refrigeración se reduce y esto hace que tarde más tiempo en bajar la temperatura de la sala. Además, hay que recordar que el sistema de refrigeración del banco de pruebas es un aire acondicionado doméstico que no está diseñado para este tipo de salas, por lo que existe una limitación física que es ajena al sistema de actuación. Este efecto también se produce durante el invierno aunque de forma inversa. Esta es una de las razones por la que no se ha podido comprobar el funcionamiento del actuador en el caso de subir y bajar la temperatura de extremo a extremo.

Capítulo 7

Conclusiones y Líneas Futuras

En este capítulo se exponen las conclusiones más importantes de este trabajo y cuáles son las bases para futuros trabajos relacionados.

7.1. Conclusiones

Una vez realizado el trabajo, las principales conclusiones que se extraen de él son:

- Se ha diseñado e implementado un sistema de actuación capaz de fijar la temperatura de una sala de servidores a un valor concreto, modificando la temperatura del sistema de refrigeración de dicha sala.
- Se ha realizado un diseño modular y fácilmente extendible a otros entornos con diferentes plataformas de monitorización y visualización de datos, unidades de refrigeración o con diferentes políticas de control.
- Se ha conseguido implementar un controlador PID sencillo que permite controlar la temperatura de la sala.
- Se ha logrado un sistema de actuación autónomo, capaz de responder a los cambios de temperatura de manera dinámica y estable.
- El sistema de actuación ha sido implantado en un entorno real (sala B039) y se integrado con la plataforma de monitorización y visualización *Graphite*. Se ha verificado su funcionamiento en dicho entorno y se ha demostrado que el actuador funciona correctamente en un entorno real y consigue fijar la temperatura de la sala al valor deseado y para variaciones de temperatura tanto pequeñas como grandes.

7.2. Lineas Futuras

Para terminar, se detallan las posibles acciones futuras que pueden llevarse a cabo, partiendo este trabajo:

- Diseñar el hardware propio del sistema de actuación e implementar el software en él. En este trabajo, se ha utilizado una Raspberry PI como soporte hardware.
- Analizar la posible implementación de otras políticas de control que permitan controlar la temperatura de la sala e implementarlas en el sistema de actuación de este trabajo para comprobar su funcionamiento.
- Optimizar el controlador PID existente, mediante un ajuste más fino de sus parámetros, añadir configuraciones que eviten el efecto windup o usar otro tipo de configuraciones PID más sofisticadas para así conseguir optimizar el funcionamiento del actuador.
- Integrar el sistema de actuación en el sistema de optimización de CPDs diseñado por *GreenLSI* y comprobar su funcionamiento en un entorno donde la temperatura óptima varía. En este trabajo, la temperatura óptima se elegía de manera arbitraria y era fijada por el usuario.
- Extender el sistema a varias unidades de refrigeración. El sistema diseñado en este trabajo actúa sobre un único sistema de refrigeración. Es recomendable estudiar cómo se puede integrar el actuador en un entorno donde hay varias unidades de refrigeración y cómo se puede elaborar una respuesta coordinada que logre que la sala alcance la temperatura óptima.

Capítulo 8

Anexos

8.1. Anexo 1. Estimación de la función de transferencia

Este anexo está dividido en 2 partes: la primera parte incluye toda la información sobre el proceso de estimación de la función de transferencia descrito en el apartado 4.4.1 y la segunda parte incluye el script de matlab elaborado para hacer este proceso.

8.1.1. Proceso de estimación de la función de transferencia

Para estimar la función de transferencia se ha realizado un experimento que consiste en aplicar una señal de entrada escalón de 18°C a 32°C (para cubrir el rango de funcionamiento del sistema de refrigeración) y se ha medido la salida generada por dicho experimento. Este experimento se ha realizado 13 veces durante varios días y a diferentes horas para poder caracterizar mejor el comportamiento dinámico la planta.

Para cada experimento se ha estimado la función de transferencia que mejor se ajusta a los datos generados en dicho experimento. Se han tenido en cuenta todas las combinaciones posibles de polos y ceros hasta orden 3. La selección de la función de transferencia se ha realizado en base al coeficiente de ajuste.

En la tabla 8.1 se muestra la expresión matemática de la función de transferencia obtenida en cada experimento. Esta tabla sólo incluye la combinación que tiene un mayor coeficiente de ajuste. El resto de combinaciones se han descartado porque es poco probable que una función que no se ajusta bien a sus propios datos pueda ajustarse mejor a los datos de los otros experimentos.

Num exp	Función estimada	Coef ajuste
1	$G(s) = \frac{6,624 \cdot 10^{-2}s^2 + 4,942 \cdot 10^{-4}s + 8,697 \cdot 10^{-7}}{s^3 + 0,5169s^2 + 3,318 \cdot 10^{-3}s + 1,123 \cdot 10^{-6}}$ $z_1 = -4,6 \cdot 10^{-3}, \quad z_2 = -2,8 \cdot 10^{-3}$ $p_1 = -0,5104, \quad p_2 = -6,1 \cdot 10^{-3},$ $p_3 = -3,583 \cdot 10^{-4}$	91,3831 %
2	$G(s) = \frac{8,686 \cdot 10^{-2}s + 1,213 \cdot 10^{-4}}{s + 1,592 \cdot 10^{-4}}$ $z_1 = -1,4 \cdot 10^{-3}$ $p_1 = -1,5924 \cdot 10^{-4}$	85,5679 %
3	$G(s) = \frac{4,382 \cdot 10^{-3}s + 4,952 \cdot 10^{-6}}{s^2 + 0,02972s + 6,397 \cdot 10^{-6}}$ $z_1 = -1,1 \cdot 10^{-3}$ $p_1 = -2,95 \cdot 10^{-2}, \quad p_2 = -2,167 \cdot 10^{-4}$	89,4146 %
4	$G(s) = \frac{4,976 \cdot 10^{-2}s^2 + 1,127 \cdot 10^{-3} + 1,805 \cdot 10^{-6}}{s^3 + 0,3519s^2 + 7,369 \cdot 10^{-3} + 2,314 \cdot 10^{-6}}$ $z_1 = -2,09 \cdot 10^{-2}, \quad z_2 = -1,7 \cdot 10^{-3}$ $p_1 = -0,3296, \quad p_2 = -2,20 \cdot 10^{-2},$ $p_3 = -3,189 \cdot 10^{-4}$	91,4906 %
5	$G(s) = \frac{2,033 \cdot 10^{-2}s^2 + 6,038 \cdot 10^{-4} + 8,775 \cdot 10^{-7}}{s^3 + 0,1792s^2 + 4,498 \cdot 10^{-3} + 1,191 \cdot 10^{-6}}$ $z_1 = -2,82 \cdot 10^{-2}, \quad z_2 = -1,5 \cdot 10^{-3}$ $p_1 = -0,1490, \quad p_2 = -2,99 \cdot 10^{-2},$ $p_3 = -2,6754 \cdot 10^{-4}$	88,7664 %
6	$G(s) = \frac{8,432 \cdot 10^{-5}s + 1,697 \cdot 10^{-9}}{s^3 + 0,282s^2 + 2,155 \cdot 10^{-4} + 1,046 \cdot 10^{-14}}$ $z_1 = -2,0124 \cdot 10^{-5}$ $p_1 = -0,2813, p_2 = -7,663 \cdot 10^{-4},$ $p_3 = -4,85 \cdot 10^{-11}$	90,0349 %

Num exp	Función estimada	Coef ajuste
7	$G(s) = \frac{8,62 \cdot 10^{-10}}{s^3 + 1,277 \cdot 10^{-3}s^2 + 2,333 \cdot 10^{-6}s + 1,109 \cdot 10^{-9}}$ $p_1 = -3,57 \cdot 10^{-4} + 0,0013i, \quad p_2 = -3,57 \cdot 10^{-4} - 0,0013i,$ $p_3 = -5,747 \cdot 10^{-4}$	84,4719 %
8	$G(s) = \frac{7,439 \cdot 10^{-2}s + 1,308 \cdot 10^{-4}}{s^2 + 0,5104s^2 + 1,612 \cdot 10^{-4}}$ $z_1 = -1,8 \cdot 10^{-3}$ $p_1 = -0,5101, \quad p_2 = -3,1601 \cdot 10^{-4}$	90,4214 %
9	$G(s) = \frac{7,628 \cdot 10^{-2}s^2 + 7,03 \cdot 10^{-4}s + 9,769 \cdot 10^{-7}}{s^3 + 0,548s^2 + 4,787 \cdot 10^{-3}s + 1,211 \cdot 10^{-6}}$ $z_1 = -7,5 \cdot 10^{-3}, \quad z_2 = -1,7 \cdot 10^{-3}$ $p_1 = -0,5392, \quad p_2 = -8,6 \cdot 10^{-3},$ $p_3 = -2,61 \cdot 10^{-4}$	91,0543 %
10	$G(s) = \frac{4,413 \cdot 10^{-7}s + 2,439 \cdot 10^{-9}}{s^3 + 8,157 \cdot 10^{-2}s^2 + 1,603 \cdot 10^{-5}s + 3,753 \cdot 10^{-9}}$ $z_1 = -5,5 \cdot 10^{-3}$ $p_1 = -8,14 \cdot 10^{-2}, \quad p_2 = -9,825 \cdot 10^{-5} + 1,9097 \cdot 10^{-4}i,$ $p_3 = -9,825 \cdot 10^{-5} - 1,9097 \cdot 10^{-4}i,$	84,86 %
11	$G(s) = \frac{0,5884s + 1,197 \cdot 10^{-3}}{s^3 + 4,457s^2 + 4,228s + 1,475 \cdot 10^{-3}}$ $z_1 = -2,03 \cdot 10^{-3}$ $p_1 = -3,0877, \quad p_2 = -1,3688,$ $p_3 = -3,4898 \cdot 10^{-4}$	89,8722 %
12	$G(s) = \frac{2,868 \cdot 10^{-4}}{s + 3,319 \cdot 10^{-4}} \quad p_1 = -3,3186 \cdot 10^{-4}$	89,0861 %
13	$G(s) = \frac{2,449 \cdot 10^{-10}}{s^3 + 9,255 \cdot 10^{-3}s^2 + 4,247 \cdot 10^{-6}s + 2,603 \cdot 10^{-10}}$ $p_1 = -8,8 \cdot 10^{-3}, \quad p_2 = -4,0788 \cdot 10^{-4},$ $p_3 = -7,2718 \cdot 10^{-5}$	91,05 %

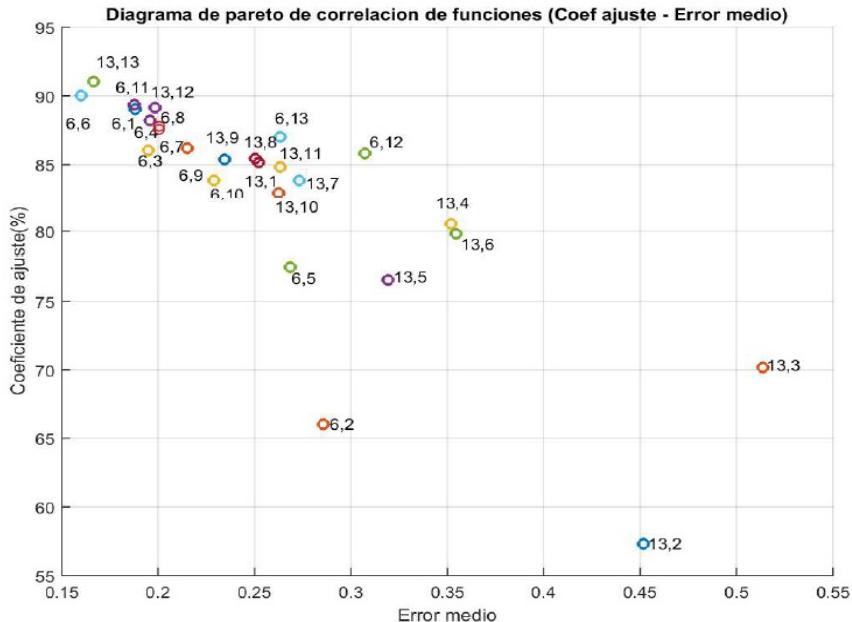
Tabla 8.1: Tabla con las funciones de transferencia estimadas

Una vez determinada las función de transferencia mejor ajustada en cada experimento, se mide cómo se ajusta cada una de estas funciones a los datos generados por el resto de experimentos. En la tabla 8.1.1 se muestra el coeficiente de ajuste de cada función con respecto a sus datos y a los datos de los otros experimentos. Se ha marcado en negro la función que presenta un mejor ajuste para cada experimento.

Analizando los datos de la tabla 8.1.1, se observa que en cada experimento se obtiene un mejor ajuste con los datos propios. El experimento que tiene un mayor coeficiente de ajuste es el número 4 con un valor del 91,4906 %. Sin embargo, puede verse en la tabla que los experimentos 6 y 13 tienen un coeficiente de ajuste medio superior al 80 % y que sus funciones tienen mejores coeficientes de ajuste que el resto. Además, el experimento 6 y el experimento 13 tienen unos coeficientes de ajuste propio de 90,0349 % y 91,05 % respectivamente, luego la variación con el experimento 4 es mínima. Por tanto, se decide que los experimentos 6 y 13 proporcionan una función de transferencia que se ajusta mejor a todos los experimentos.

Para decidir cuál de las 2 funciones se aproxima mejor, se va a calcular el error medio y el error máximo absoluto que tienen ambas funciones con respecto a los datos de cada experimento. Además, se van a representar los resultados obtenidos en un diagrama de pareto para visualizarlos mejor.

En la tabla 8.1.1 puede verse el coeficiente de ajuste, el error cuadrático medio y el máximo error absoluto de ambos experimentos y en las figuras 8.1 y 8.2 se muestra un diagrama de pareto coeficiente de ajuste - error medio y un diagrama de pareto coeficiente de ajuste - error máximo respectivamente.



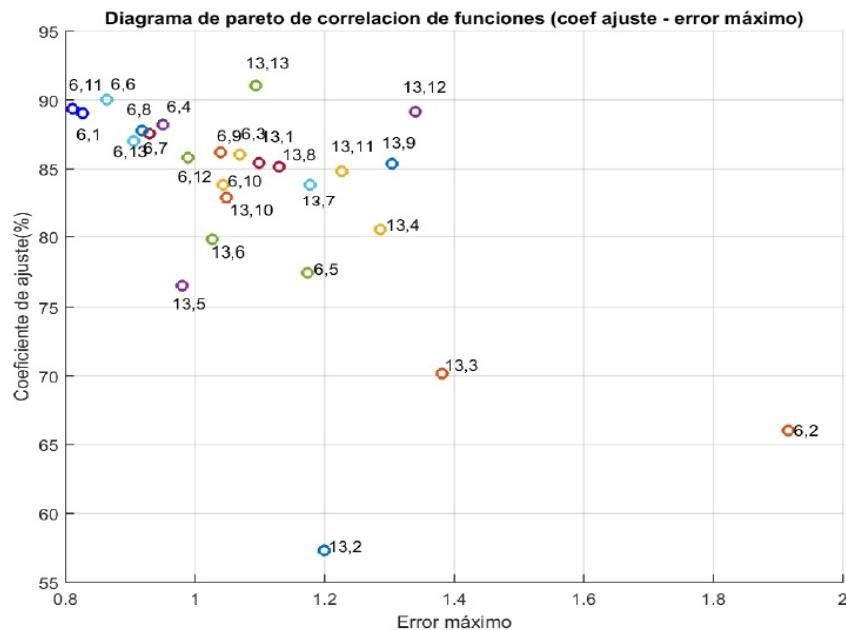


Figura 8.2: Diagrama de pareto coef de ajuste - error máximo experimentos 6 y 13

Examinando dicha tabla se observa que el experimento 6 presenta mejores resultados medios con respecto al experimento 13 en los 3 parámetros. El experimento 6 tiene mejores valores en la mayoría de los experimentos con respecto al experimento 13. En ambos diagramas de pareto se observa que hay un mayor número de casos del experimento 6 situados en la parte superior izquierda de la figura, que se corresponde con un mayor coeficiente de ajuste y un menor error. Por otro lado, se observa que una gran parte de casos del experimento 13 se encuentra en la zona central de la figura, lo que implica que tienen un menor coeficiente de ajuste y un mayor error.

Por tanto, se concluye que el experimento número 6 proporciona una función de transferencia estimada que se ajusta mejor a los datos de todos los experimentos que el resto de experimentos y se decide que dicha función es la que mejor caracteriza el comportamiento dinámico de la planta. A continuación se muestra la expresión de dicha función de transferencia:

$$G(s) = \frac{8,4315 \cdot 10^{-5}s + 1,6968 \cdot 10^{-9}}{s^3 + 0,2820s^2 + 2,1553 \cdot 10^{-4}s + 1,0462 \cdot 10^{-14}} \quad (8.1)$$

Coeficiente de ajuste de cada experimento con las funciones estimadas en cada experimento

Num Exp	1	2	3	4	5	6	7	8	9	10	11	12	13	Media
1	91,3831	7,7316	68,1859	91,3615	41,1466	90,1374	76,5276	74,2446	85,5091	83,1777	69,5568	30,6643	28,6190	62,2385
2	46,4539	85,5679	63,6155	45,7268	74,5264	45,9584	33,6166	32,5180	40,0218	53,1230	30,0077	5,4745	3,0797	39,5102
3	73,0536	46,4246	89,4146	73,4109	77,5897	73,0610	57,8418	56,0292	66,0866	82,5248	52,3741	20,8814	19,2382	58,2096
4	90,3382	4,1194	67,5867	91,4906	41,9392	88,6594	76,7551	74,3276	85,7107	82,9944	69,6904	32,8622	31,3778	62,1968
5	58,2029	77,8641	69,5411	53,1426	88,7664	52,0389	36,8298	36,0635	43,4967	63,5883	32,9378	-4,4508	-9,3537	42,4918
6	89,0111	66,0086	86,0577	88,2092	77,4787	90,0349	87,5776	87,7922	86,2082	83,8632	89,3537	85,8163	86,9912	84,5306
7	64,1301	-29,3008	40,0983	68,4246	-1,3793	67,5641	84,4719	85,7168	75,2369	51,2921	84,5548	41,7830	38,3973	48,8765
8	75,5006	-49,2439	33,4329	70,8841	10,6069	67,0687	89,3440	90,4214	78,5941	56,8271	88,3031	57,9744	58,9475	53,1866
9	89,6178	-14,5296	54,8458	86,3504	39,7781	83,3306	84,2851	81,8112	91,0543	77,8724	77,1474	48,0135	49,1641	63,1406
10	84,6609	67,2489	71,1708	84,5313	81,3005	85,2595	84,9659	85,4456	84,5013	84,8600	84,7497	63,8425	49,2469	77,2436
11	68,6527	-60,2874	25,9336	64,1865	-0,2220	60,4990	85,8828	88,6983	72,3367	48,8204	89,8722	61,1804	61,8088	48,1242
12	43,1090	-134,0588	-24,8806	26,2330	-35,0181	21,6890	52,6691	58,4728	35,7301	14,2683	64,2494	89,0861	85,9484	17,3676
13	85,4331	57,3050	70,1803	80,5133	76,5364	79,8458	83,8308	85,1552	85,3857	82,9430	84,8552	89,1679	91,0500	80,0960

Coeficiente de ajuste de los experimentos 6 y 13

Num Exp	1	2	3	4	5	6	7	8	9	10	11	12	13	Media
6	89,0111	66,0086	86,0577	88,2092	77,4787	90,0349	87,5776	87,7922	86,2082	83,8632	89,3537	85,8163	86,9912	84,5306
13	85,4331	57,3050	70,1803	80,5133	76,5364	79,8458	83,8308	85,1552	85,3857	82,9430	84,8552	89,1679	91,0500	80,0960

Error cuadrático medio de los experimentos 6 y 13

Num Exp	1	2	3	4	5	6	7	8	9	10	11	12	13	Media
6	0,1881	0,2859	0,1951	0,1956	0,2684	0,1599	0,2006	0,2150	0,2291	0,1875	0,3074	0,2632	0,2280	0,1477
13	0,2504	0,4515	0,5135	0,3518	0,3193	0,3546	0,2732	0,2523	0,2347	0,2625	0,2935	0,1985	0,1666	0,3105

Error máximo de los experimentos 6 y 13

Num Exp	1	2	3	4	5	6	7	8	9	10	11	12	13	Media
6	0,8270	1,9157	1,0095	0,9497	1,1739	0,8637	0,9297	0,9183	1,0394	1,0433	0,8114	0,9891	0,9058	1,0477
13	1,0991	1,1993	1,3817	1,2856	0,9804	1,0266	1,1769	1,1300	1,3033	1,0482	1,2258	1,3396	1,0941	1,1830

Tabla 8.2: Coeficiente de ajuste de cada función con el resto de experimentos

8.1.2. Scripts de matlab para hacer la estimación

Este apartado incluye los scripts de matlab que se han diseñado para calcular la estimación de la función de transferencia. Se han diseñado 4 scripts.

- **estimador.m:** este script se utiliza para extraer los datos de la plataforma y realizar la estimación de la función de transferencia para todas las combinaciones hasta orden 3.
- **seleccionarEstimacion.m:** este script se utiliza para seleccionar de cada experimento la función de transferencia que tiene un mejor coeficiente de ajuste.
- **analizador.m:** este script se utiliza para realizar la correlación entre todas las funciones de transferencia de todos los experimentos y seleccionar la función de transferencia que mejor se aproxima a todos los experimentos.
- **conv_hora_unix.m:** función que se utiliza para convertir la hora a formato unix. Esta función se utiliza en el script estimador.m.

Para mayor claridad y debido a la longitud de estos scripts, se ha decidido no incluir el código en esta memoria. En su lugar, se ha habilitado un repositorio en github donde se pueden consultar estos scripts. La dirección http de este repositorio es la siguiente: <https://github.com/jccalvo/TFG.git>

8.2. Anexo 2: Otros experimentos

Este anexo incluye algunos de los experimentos realizados en este TFG y que no han sido incluidos en el capítulo 6. Estos experimentos están clasificados siguiendo los criterios definidos en dicho capítulo. Existe algún experimento que cubre un caso intermedio y por lo tanto, no ha podido ser clasificado. Sin embargo, también es interesante para evaluar el funcionamiento del actuador. En la siguiente tabla se muestra una relación de los experimentos que se incluyen en este anexo.

Num Exp	Tipo exp	Experimento	Parámetros PID
5	1	Subida de 19°C a 24°C	$K_p = 28 \ K_i = 0,037 \ K_d = 300$
6	1	Subida de 19,5°C a 23°C	$K_p = 28 \ K_i = 0,037 \ K_d = 300$
7	1 y 2	Subida de 19,5°C a 22°C y bajada de 22°C a 18°C	$K_p = 28 \ K_i = 0,037 \ K_d = 300$
8		Subida de 21°C a 23°C y bajada de 23°C a 21°C	$K_p = 28 \ K_i = 0,037 \ K_d = 300$
9	3	Subida de 19°C a 23,5°C en pasos de 0,5°C	$K_p = 28 \ K_i = 0,037 \ K_d = 300$
10	3	Subida de 21,5°C a 23,5°C en pasos de 0,5°C	$K_p = 28 \ K_i = 0,037 \ K_d = 300$
11	1 y 4	Subida de 19,5°C a 23°C y bajada de 23°C a 21,5°C en pasos de 0,5°C	$K_p = 28 \ K_i = 0,037 \ K_d = 300$
12	3	Subida de 19,5°C a 21°C en pasos de 0,5°C	$K_p = 28 \ K_i = 0,037 \ K_d = 300$

Tabla 8.3: Características de los experimentos realizados

En cada gráfica aparece la temperatura de la sala (azul), la temperatura óptima (rojo) y el setpoint (verde). No se ha incluido en ninguna gráfica la señal de control debido a que en ocasiones toma valores muy grandes o muy pequeños que impiden ver con claridad el resto de señales. A continuación se muestran las gráficas de los experimentos de la tabla 8.2.

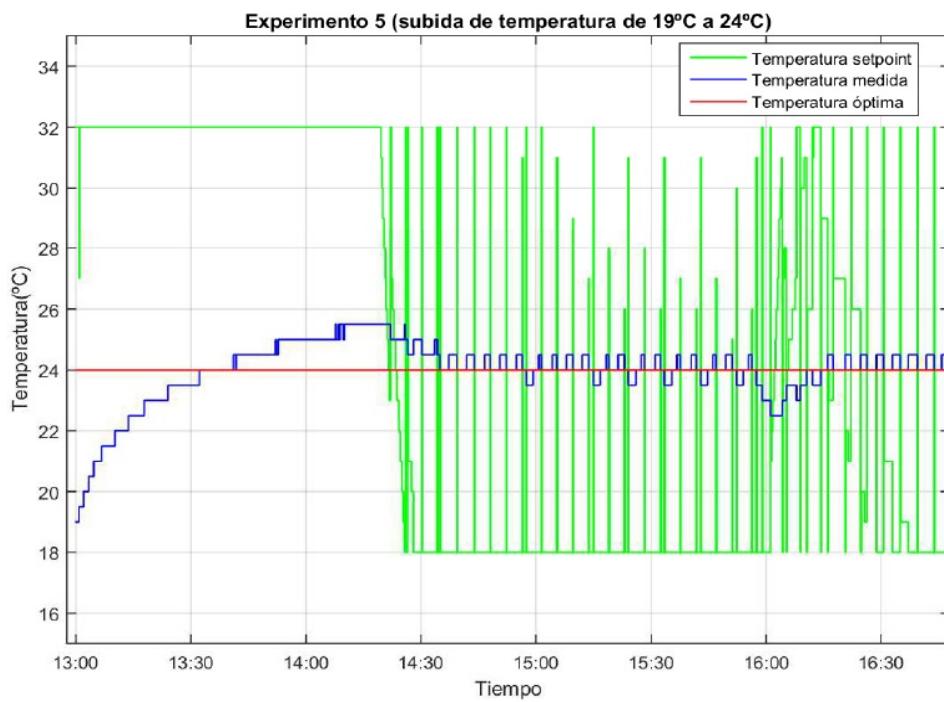


Figura 8.3: Gráfica con los resultados del experimento 5

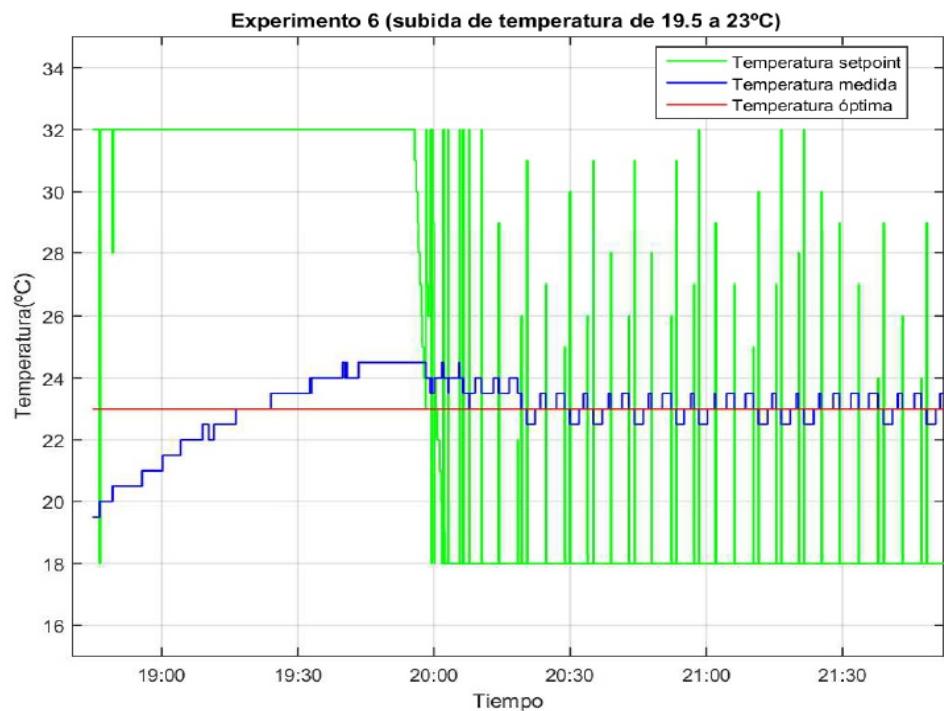


Figura 8.4: Gráfica con los resultados del experimento 6

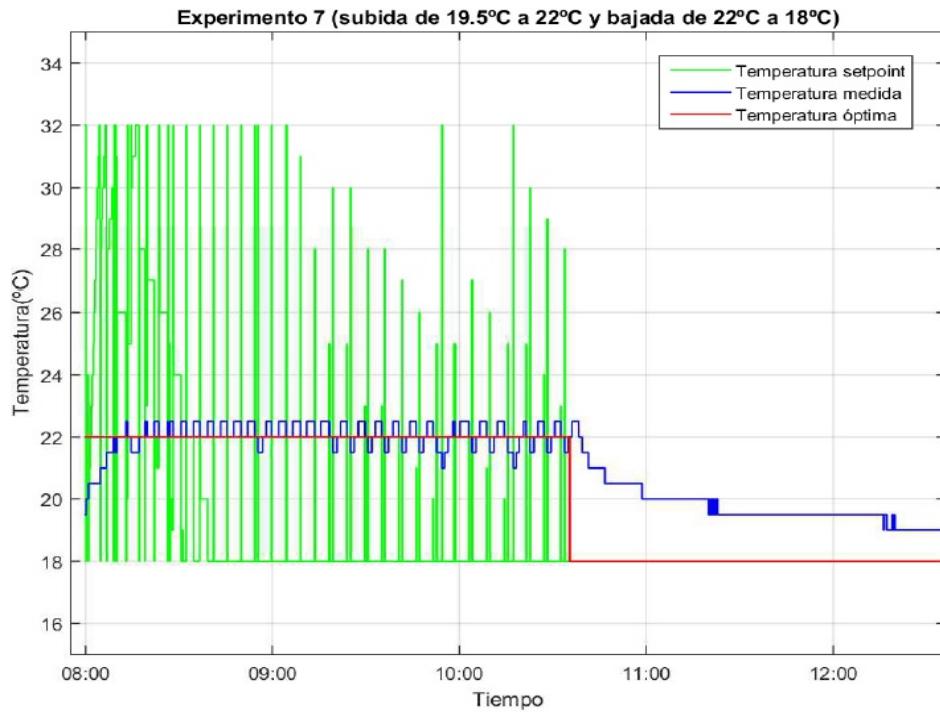


Figura 8.5: Gráfica con los resultados del experimento 7

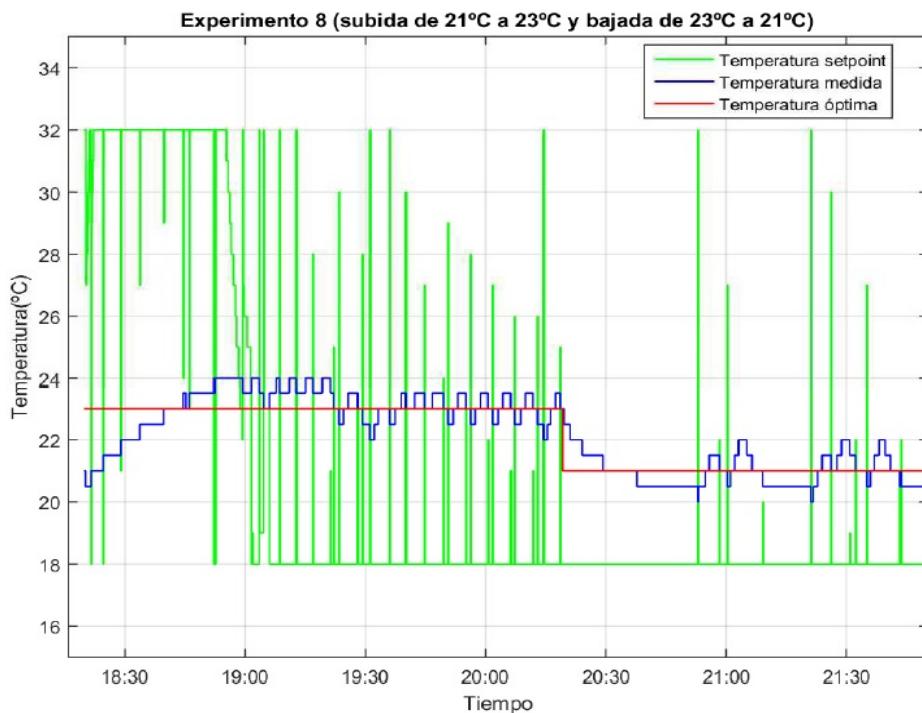


Figura 8.6: Gráfica con los resultados del experimento 8

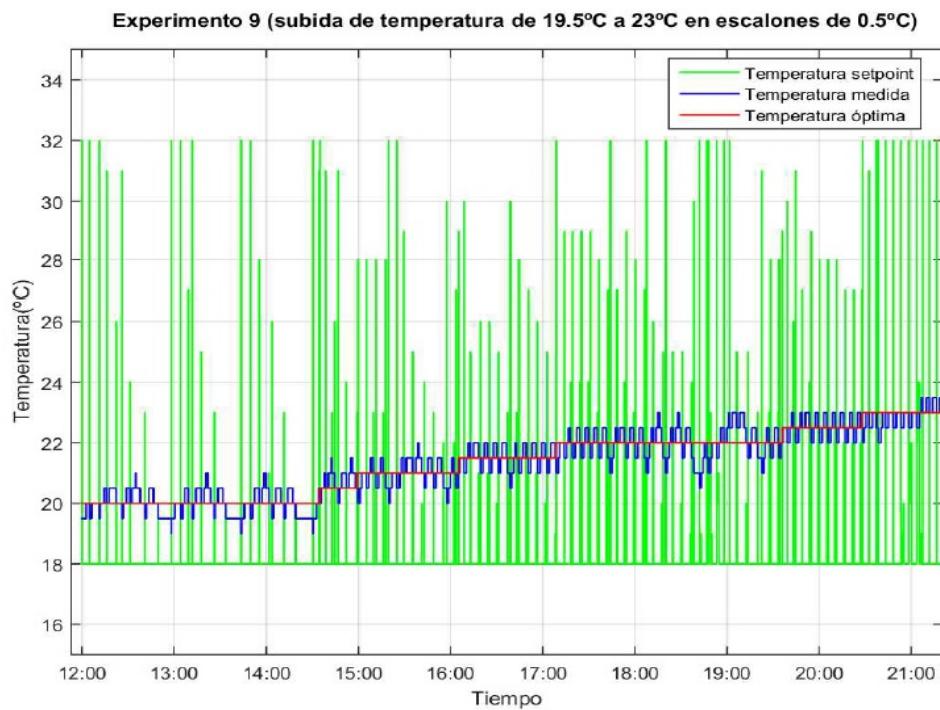


Figura 8.7: Gráfica con los resultados del experimento 9

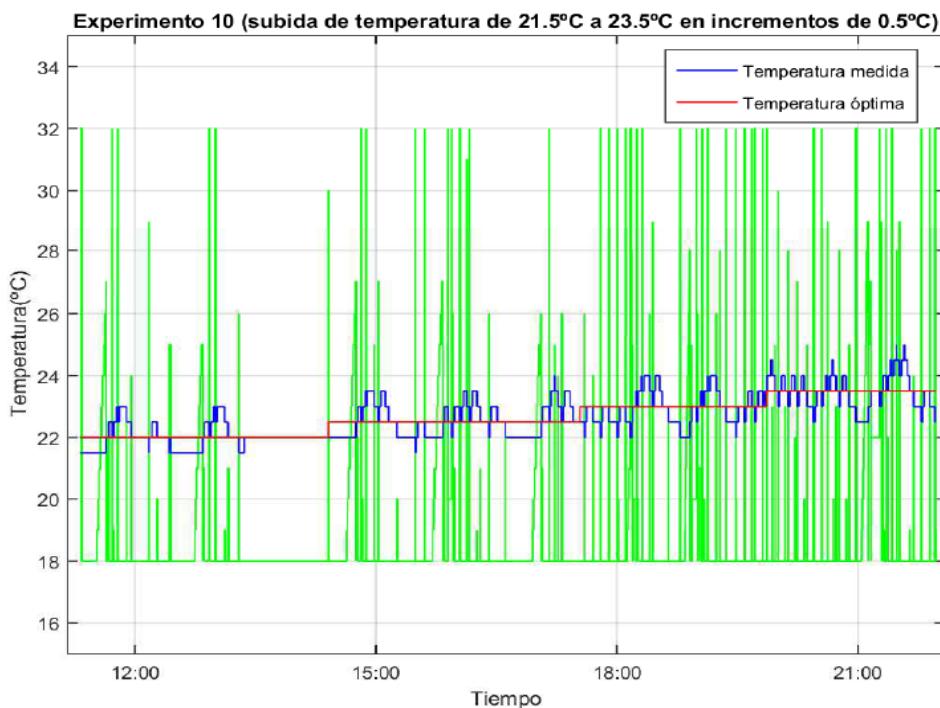


Figura 8.8: Gráfica con los resultados del experimento 10

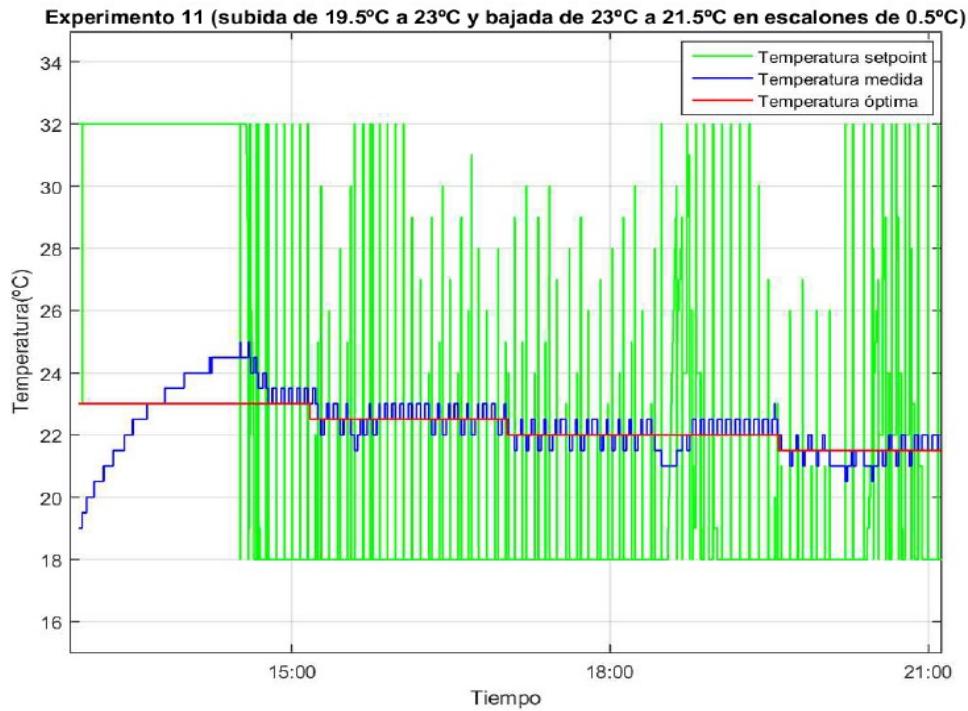


Figura 8.9: Gráfica con los resultados del experimento 11

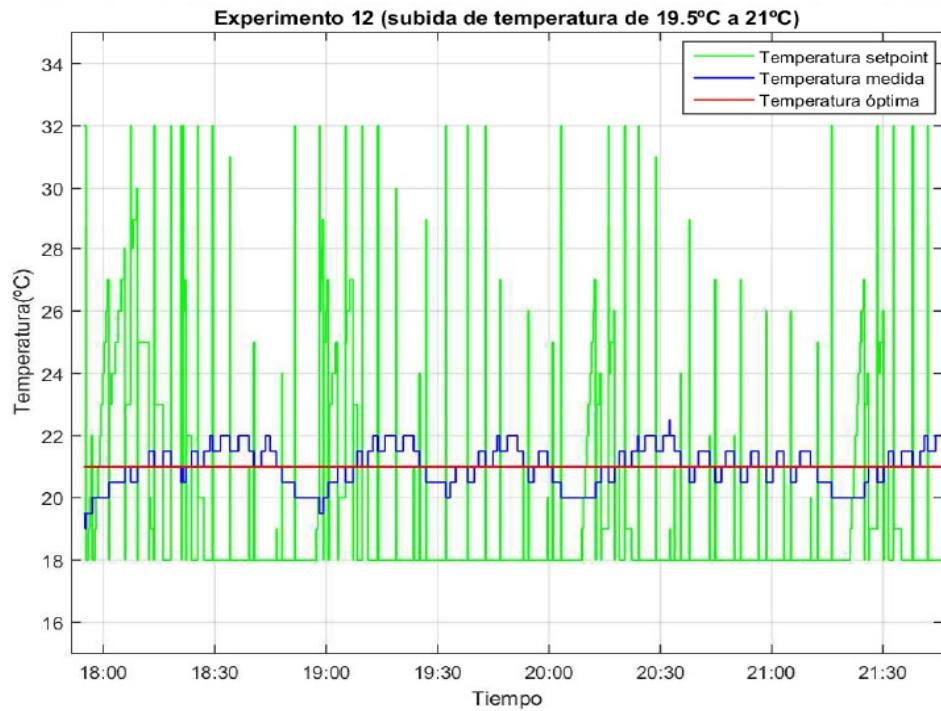


Figura 8.10: Gráfica con los resultados del experimento 12

8.3. Anexo 3: Script del controlador PID

En este anexo se incluye el script diseñado en matlab para ajustar el controlador PID y simular la respuesta del sistema ante ese controlador. El script es un único fichero denominado **PID.m**. En este script también se incluye el proceso de discretización del PID, usando la aproximación de tustin para evaluar si la discretización realizada es válida o no. A continuación se muestra el código de dicho script.

Este script también se encuentra almacenado en el repositorio github indicado en el anexo 8.1. La dirección http es: <https://github.com/jccalvo/TFG.git>

```
% - Script PID.m
%
% - Contiene el código para poder ajustar el controlador PID y
% - realizar la discretización de éste.
% - Nombre: Juan Carlos Calvo Sansegundo. Fecha: 04/07/2017.

% Paso 0: Carga de la función de transferencia estimada.

clc, clear;
load Exp6_18a32
H_num = datos.best.num;
H_den = datos.best.den;
H = tf(H_num,H_den);

[H_ceros,H_polos,H_ganancia]=tf2zp(H_num,H_den);

% Definición del vector de tiempos.
Ts = 10;
t_final =4000;
t = 0:Ts:t_final;
s_ref = ones(1,t_final/Ts + 1);

% Respuesta al escalón
lsim(H,s_ref,t);
grid on;
clear datos;

% Paso 1. Diagrama de bode la planta.

[MG, MF, WCG, WCF]=margin(H_num,H_den);
bode(H,{10^-6,10^6});
grid on;
hold off;

% Paso 2. Diagrama de nyquist de la planta

% Definición de la circunferencia unidad.
theta = 0:0.01:2*pi;
x = cos(theta);
```

```

y = sin(theta);
figure;
plot(x,y);

% Dibujar los polos y ceros del sistema.
n_polos = size(H_polos);
polos = ones([2 n_polos]);
for i=1:n_polos
    polos(1,i)=real(H_polos(i));
    polos(2,i)=imag(H_polos(i));
end;

n_ceros = size(H_ceros);
ceros = ones([2 n_ceros]);
for i=1:n_ceros
    ceros(1,i)=real(H_ceros(i));
    ceros(2,i)=imag(H_ceros(i));
end;

hold on
plot(polos(1,:),polos(2,:),'xr');
plot(ceros(1,:),ceros(2,:),'or');

% Inclusión del diagrama de Nyquist.
nyquist(H);
axis([-10^-3 10^-3 -10^-3 10^-3]);
hold off;

% Paso 3. Definición del controlador PID y sistema realimentado.
Kp = 28;      % Ganancia proporcional.
Ki = 0.037;    % Ganancia integral.
Kd = 300;      % Ganancia derivativa.
td = Kd/Kp;    % Tiempo integral.
ti = Kp/Ki;    % Tiempo derivativo.

Gc_num =[td*ti ti 1];
Gc_den =[ti 0];
Gc = Kp*tf(Gc_num,Gc_den);

% Definición de las funciones de transferencia del sistema.
GLa = Gc*H;    % F.T de lazo abierto.
GD = Gc*H;      % F.T de lazo directo.
GLc = tf(GLa.num{1},GLa.num{1}+GLa.den{1}); % F.T de lazo cerrado.

step(GLc,t);
grid on;

% Paso 4: Estabilidad del sistema con lazo cerrado.

% Dibujo de la circunferencia unidad.
figure;
plot(x,y,'g');

```

```
% Obtención y dibujo de polos y ceros.
[GLc_ceros,GLc_polos,GLc_ganancia]=tf2zp(GLc.num{1},GLc.den{1});

n_polos = size(GLc_polos);
polos = ones([2 n_polos]);
for i=1:n_polos
    polos(1,i)=real(GLc_polos(i));
    polos(2,i)=imag(GLc_polos(i));
end;

n_ceros = size(GLc_ceros);
ceros = ones([2 n_ceros]);
for i=1:n_ceros
    ceros(1,i)=real(GLc_ceros(i));
    ceros(2,i)=imag(GLc_ceros(i));
end;

hold on
plot(poles(1,:),poles(2,:),'xr');
plot(ceros(1,:),ceros(2,:),'or');

p = 1 + Gc*H;
nyquist(Gc*H);
hold off;

% Paso 5. Discretización del controlador PID y comparativa
% con la respuesta en tiempo continuo.

GLc_ss = ss(GLc);
GLcd_tus = c2d(GLc_ss,10,'tustin'); % Tustin;

step(GLc)
hold on
step(GLcd_tus)
hold off

title('Comparativa respuesta en tiempo continuo y en tiempo
discreto')
legend('Resp tiempo continuo','Resp tiempo discreto');
grid on;
```


Bibliografía

- [1] EMC corporation e IDC (international data corporation), "*The digital universe of opportunities*", 2014. <https://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf>
- [2] M. Grao Txapartegi y Dr. Eric Mounier. "*Data Center Technologies. New Technologies and Architectures for Efficient Data Centers*", Yole Développement, 2015. http://www.i-micronews.com/images/Flyers/Power/Yole_New_Technologies_and_Architectures_for_Efficient_Data_Centers_Flyer_web.pdf
- [3] EYP Mission Critical Facilities Inc., New York, 2011.
- [4] Rychard L. Sawyer, "*Calculating power requirements for data centers*", APC y Schneider Electric, 2011. http://www.apc.com/salestools/VAVR-5TDTEF/VAVR-5TDTEF_R1_EN.pdf
- [5] V. Avelar, D. Azevedo y A. French. "*Pue: A comprehensive examination of the metric*", 2014. https://datacenters.lbl.gov/sites/all/files/WP49-PUE%20A%20Comprehensive%20Examination%20of%20the%20Metric_v6.pdf
- [6] M. Zapater Sancho, P. Arroba García, J. M. Moya Fernández y Z. Bankovic. "*A state-of-the-art on energy efficiency in today's datacentres: Researcher's contributions and practical approaches*", Vol. XII, Oct 2011.
- [7] P. Chauhan y M. Gupta. "*Energy Aware Cloud Computing Using Dynamic Voltage Frequency Scaling*", International Journal of Computer Science And Technology (IJCST), Vol 5, Issue 4, pp 195-199, Oct-Dec 2014. <http://www.ijcst.com/vol54/2/49-Pooja-Chauhan.pdf>
- [8] Silicon Vale Leadership Group, "*Dynamic Power Management: Adjusting Data Center Capacity in Real-Time Energy Efficient Data Center Demonstration Project*", 2009. http://svlg.org/wp-content/uploads/2012/12/PowerAssure_cs.pdf
- [9] R. Buyya, A. Beloglazov y J. Abawajy. "*Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges*", PDPTA, 2010. <https://pdfs.semanticscholar.org/65f3/92ea232878e416b077605427ff62d7fe0677.pdf>

- [10] Hainan Zhang, Shuangquan Shao, Hongbo Xu, Huiming Zou y Changqing Tian. "Free cooling in a data centers: A review", Renewable and Sustainable Energy Reviews 35, pp 171-182, 2014. http://ac.els-cdn.com/S1364032114002445/1-s2.0-S1364032114002445-main.pdf?_tid=5780469c-30ef-11e7-bdaa-0000aab0f26&acdnat=1493919198_8346bfe1c779548bcc2e857ac03500bc
- [11] Dan Hoffman. "10 Techniques for improving data center power efficiency", Net app vision paper, Nov 2007. <http://www.uk.insight.com/content/dam/insight/EMEA/uk/shop/netapp/netappimproving-data-center-power-efficiency.pdf>
- [12] Y. Xie y W. L. Hung. "Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (mpsoc) design", Journal of VLSI signal processing systems for signal, image and video technology, vol. 45, no. 3, pp. 177-189, 2006.
- [13] GreenLSI. <http://www.greenlsi.die.upm.es>
- [14] M. Zapater, J. L. Ayala y J. M. Moya. "Greendisc: a hw/sw energy optimization framework in globally distributed computation". Ubiquitous Computing and Ambient Intelligence, pp 1-8, 2012. http://oa.upm.es/16826/1/INVE_MEM_2012_137570.pdf
- [15] J. Pagan Ortiz, M. Zapater Sancho, O. Cubo, P. Arroba García, V. Martín Ayuso y J. M. Moya. A cyber-physical approach to combined hw-sw monitoring for improving energy efficiency in data centers, 2013. http://oa.upm.es/29886/1/INVE_MEM_2013_167142.pdf
- [16] M. ZAPATER, O. TUNCER, J. AYALA, J. MOYA, K. VAIDYANATHAN, K. GROSS and A. COSKUN. "Leakage-aware cooling management for improving server energy efficiency", IEEE Transactions on Parallel and Distributed Systems, vol 26, Issue:10, pp 2764-2777, oct 2015.
- [17] Patricia Arroba, Jose M. Moya, Jose L. Ayala y Rajkumar Buyya. "Proactive Power and Thermal Aware Optimizations for Energy-Efficient Cloud Computing", Design automation and test in europe (DATE), 2016.
- [18] Katsuhiko Ogata. "Ingeniería de control moderna", Ed Pearson Education, 5º edición, 2010.
- [19] Benjamin C. Kuo. "Sistemas de control automático", Ed Prentice Hall, 7º Edición, 1996.
- [20] Karl J. Astrom y Tore Hagglund. "Control PID avanzado", Ed Pearson Education, 2009.
- [21] Bogdan M. Wilamowski & J. David Irwin. "The Industrial Electronics Handbook: Control and Mechatronics", 2º edition, CRC Press, 2011.
- [22] Fabricante de controladores Omron. <https://www.ia.omron.com/support/guide/53/introduction.html>

- [23] Fabricante de controladores Omega. <http://www.omega.com/prodinfo/temperaturecontrollers.html>
- [24] Fabricante de controladores Coulton. http://www.coulton.com/QandA_controller
- [25] Fabricante de controladores Imopc. http://www.imopc.es/content.php?p=spotlight_TempControl&lang=ES
- [26] Documentación *Graphite*. <http://graphite.readthedocs.io/en/latest/index.html>
- [27] Sara Álvarez Vinagre y Alfredo Tendero Casanova. *Memoria proyecto Pimote SDG2*, GreenLSI, Departamento de Ingeniería Electrónica (DIE), Universidad Politécnica de Madrid, 2013.
- [28] Protocolo de comunicación SPI. <http://www.i-micro.com/pdf/articulos/spi.pdf>
- [29] Librería *curl* para peticiones HTTP en C. <https://curl.haxx.se/libcurl/>
- [30] Librería *jansson* para manejar objetos JSON en C. <https://jansson.readthedocs.io/en/2.10/>