Práctica 8:
Control de temperatura en lazo cerrado de una lámpara incandecente usando Arduino y la Raspberry Pi

Fundamentos de Sistemas Embebidos

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno aprenderá a controlar la temperatura de un entorno cerrado calentado una carga resistiva de alta potencia opto-acoplada a un microcontrolador por medio de un detector de cruce por cero, un TRIAC, y un sensor de temperatura LM35.

2. Introducción

La presente práctica resume los pasos a seguir para modular la temperatura de un entorno cerrado usando un sensor de temperatura LM35 y un foco incandescente o carga resistiva de alta potencia modulada mediante un detector de cruce por cero y un TRIAC. En particular, se interesa en la implementación de un controlador proporcional e interfaz I^2C entre un Arduino y una Raspberry Pi.

2.1. Lámparas, eficiencia y calor

En general, una lámpara cuenta con tres características fundamentales que definen su desempeño:

i) **Brillo**: Medido en *lúmenes*, el brillo determina la cantidad de iluminación que la lámpara provee; es decir, la cantidad de lúmenes es directamente proporcional a la iluminación que la lámpara proveerá. De hecho, el lúmen (abreviado lm) es la unidad del Sistema Internacional para medir el brillo, medida de la cantida de luz visible emitida por una fuente por unidad de tiempo; y que no debe ser confundido con la potencia total irradiada que contempla todas las frecuenciasen el espectro electromagnético, visibles o no.

ii) **Potencia**: Medida en *watts*, se refiere a la potencia de consumo del dispositivo, aproximado por el producto del voltaje por la corriente. Debe diferenciarse de la potencia de trabajo que corresponde al trabajo realizado por el dispositivo, en este caso la cantidad de energía convertida en luz, y de la potencia total disipada que corresponde a diferencia entre la potencia suministrada y el trabajo realizado, es decir la energía perdida como calor y absorbida por los materiales.

iii) **Temperatura**: Medida en *Kelvin*, se refiere al color de la luz que emite el dispositivo y está asociada a la temperatura en Kelvin del filamento en una lámpara incandecente y a la percepción subjetiva o sensación de calor que esta luz produce. Esto produce un efecto contradictorio entre los valores numéricos y su descripción. Contrario al sentido común, se le llama luz cálida a las lámparas que emiten luz en el rango de 2700–3000K y que emiten una mayor cantidad de fotones en la zona rojiza del espectro de luz visible, mientras que las lámparas de luz fría irradian en el rango de 5000–6500K que corresponde a la zona azulada del espectro de luz visible. Por último, están las lámparas de luz neutra que irradian en la zona amarillo-verdosa del espectro de luz visible en el rango de los 3500–4100K. Es importante hacer notar que, contrario al sentido común, de estas tres características y para los objetivos de esta práctica, la temperatura de una lámpara es el factor menos relevante a tomar en cuenta, puesto que es la que menos contribuye a la generación de calor.

Tabla 1: Eficiencia comparativa de lámparas[1]

Foco	**Consumo [W]**				**Eficiencia** $\left[\frac{\mathbf{lm}}{\mathbf{W}}\right]$	**Eficiencia [%]**
	1500lm	1100lm	800lm	450lm		
Incandescente	100	75	60	40	15.0	2.1
Fluorescente	20	15	11	7	75.0	10.9
LED	13	9	8	5	125.0	18.3

La eficiencia de una lámpara se determina como la cantidad de lúmenes que esta puede suministrar por Watt. Una lámpara teórica perfecta tendría que ser capaz de convertir el 100 % de la energía proporcionada (electrones) en luz (fotones), valor que se encuentra determinado en $683^{lm}/_{w}$. Este valor se utiliza para calcular la eficiencia real de cualquier lámpara del tipo que sea, como los ejemplos mostrados en la Tabla 1.

Con una eficiencia del 2.1 %, la lámpara incandescente se vuelve la opción más eficiente para generar calor (que no luz), ya que por cada 100W suministrados, aproximadamente 98W se convertirán en calor. Además, otros tipos de lámparas funcionan por medio de una electrónica compleja basada en inductores o *clamppers* que, además de ser no-lineales, son incompatibles con el método de modulación de potencia por detección de cruce por cero que se estudió en la práctica anterior.

Aparece entonces la pregunta de por qué no usar una resistencia convencional de alta potencia para estufa capaz de convertir el 99.9 % de la energía suministrada en calor. Esto es debido a varias razones. Primero, por costos, ya que una lámpara incandescente es mucho más barata que otro tipo de generadores de claor. Segundo, porque es mucho más sencillo aprender a controlar un dispositivo observable a simple vista en el cual el calor disipado será proporcional de forma lineal al brillo emitido. El tercer factor es la seguridad, ya que las resistencias para estufa generan demasiado calor demasiado rápido y pueden ocasionar quemaduras y encender llamas en segundos. Y, por último, porque las resistencias para estufa están diseñadas para operar con un termostato y una carga y no para calentar un entorno por convección, por lo que habría que acoplarle un disipador con ventilador.

Luego entonces, corresponde estimar el incremento de temperatura por segundo para un volúmen de $1dm^3$ (un litro). Se sabe que un foco incandescente opera como un resistor lineal y que 1 Watt equivale a 1 Julio por segundo. Por otro lado una caloría está definida como la cantidad de calor requerida para elevar la temperatura de un gramo de agua a 4°C un $\Delta T = 1^oC$ en condiciones de presión de una atmósfera estándar, aproximadamente unos $4.184J$.

Sin embargo, el incremento de temperatura por unidad de calor varía de material a material (por ejemplo, el metal se calienta más rápido que el agua). Por ello, es necesario calcular el incremento de temperatura que que tendrá un determinado volúmen de aire por cada caloría de energía suministrada, lo cual se logra considerando el calor específico de cada material. Se sabe que el calor específico del agua es:[2]

$$c_{p_{\text{agua}}} = 1\frac{cal}{gK} = 4.1813\frac{J}{gK} \tag{1}$$

mientras que el calor específico del aire es:

$$c_{p_{\text{aire}}} = 1.012\frac{J}{gK} \tag{2}$$

por lo que igualando valores con la fórmula del calor específico $Q = mc_p\Delta T$ se tiene que:

$$m_{\text{agua}}c_{p_{\text{agua}}}\Delta T_{\text{agua}} = Q = m_{\text{aire}}c_{p_{\text{aire}}}\Delta T_{\text{aire}}$$

considerando un volúmen fijo de un litro ($1L = 1 \times 10^{-3}m^3$) y la densidad $\rho = \frac{m}{V}$ de cada material:

$$\begin{aligned} \cancel{V_{\text{agua}}}\Big|_{1L}\rho_{\text{agua}}c_{p_{\text{agua}}}\Delta T_{\text{agua}} &= Q = \cancel{V_{\text{aire}}}\Big|_{1L}\rho_{\text{aire}}c_{p_{\text{aire}}}\Delta T_{\text{aire}} \\ \rho_{\text{agua}}c_{p_{\text{agua}}}\Delta T_{\text{agua}} &= Q = \rho_{\text{aire}}c_{p_{\text{aire}}}\Delta T_{\text{aire}} \end{aligned}$$

[2]El lector notará que el valor proporcionado del calor específico del agua no conincide con el valor de una caloría $1cal = 4.1813\frac{J}{gK} \neq 4.1813\frac{J}{gK} = c_{p_{\text{agua}}}$. Esto se debe a que el valor usado para calcular el incremento de temperatura de un volumen de aire considera una temperatura ambiente promedio de 25°C, es decir, agua a 25°C y no los 4°C usados para conocer el equivalente mecánico del calor calculado por Joule.

considerando $\rho_{\text{agua}} = 1000\frac{kg}{m^3}$ y como $\Delta^o C = \Delta K$, se tiene:

$$\begin{aligned}
\cancel{\rho}^{1000\frac{kg}{m^3}}_{\text{agua}} c_{p_{\text{agua}}} \Delta T_{\text{agua}} &= \rho_{\text{aire}} c_{p_{\text{aire}}} \Delta T_{\text{aire}} \\
\left(1000\frac{kg}{m^3}\right) c_{p_{\text{agua}}} &= \rho_{\text{aire}} c_{p_{\text{aire}}} \Delta T_{\text{aire}}
\end{aligned}$$

despejando

$$\frac{\Delta T_{\text{aire}}}{\Delta T_{\text{agua}}} = \frac{c_{p_{\text{agua}}}}{\rho_{\text{aire}} c_{p_{\text{aire}}}} \times \left(1 \times 10^3 \left[\frac{kg}{m^3}\right]\right) \tag{3}$$

reemplazando valores donde la densidad promedio del aire de $1.225\frac{kg}{m^3}$ y los calores específicos del agua y del aire (véase ecuaciones (1) y (2)) se obtiene:

$$\begin{aligned}
\frac{\Delta T_{\text{aire}}}{\Delta T_{\text{agua}}} &= \frac{4.1813\frac{\cancel{J}}{\cancel{g}\cancel{K}}}{1.012\frac{\cancel{J}}{\cancel{g}\cancel{K}} \cdot 1.225\frac{\cancel{kg}}{\cancel{m^3}}} \times \left(1 \times 10^3 \left[\frac{\cancel{kg}}{\cancel{m^3}}\right]\right) \\
&= 3.37 \times 10^3 [1]
\end{aligned}$$

o bien:

$$\Delta T_{\text{aire}} = 3.37 \times 10^3 \Delta T_{\text{agua}} \tag{4}$$

Ahora bien, si se administran 100 watts de potencia durante 1 segundo a 1 gramo de agua a temperatura ambiente, éste elevaría su temperatura en aproximadamente 24°C asumiendo un comportamiento lineal ($100Ws = 100J = 23.9cal$); pero si se tratase de un litro de agua ($\rho_{agua} = 1 \times 10^3 \frac{kg}{m^3} = 1\frac{kg}{L}$), éste sólo elevaría su temperatura en 0.024°C por tratarse de mil veces más agua. Sin embargo, interesa calentar aire y no agua. Por fortuna, la ecuación (4) nos permite conocer cuál sería el incremento de temperatura en el mismo volumen de aire. Así, aplicar 100 watts de pontencia durante 1 segundo a 1 litro de aire a temperatura ambiente elevaría la temperatura del aire en:

$$\begin{aligned}
\Delta T_{\text{aire}} &= 3.37 \times 10^3 \Delta T_{\text{agua}} \Big|_{0.024^o C} \\
&= (3.37 \times 10^3)(0.024)\,[^o C] \\
&= (3370)(0.024)\,[^o C] \\
&= 80.88^o C
\end{aligned}$$

Siguiendo este mismo razonamiento, es posible generalizar una fórmula, la Ecuación (7), que permite estimar el incremento de temperatura ΔT de un volumen de aire (en litros) por watt de energía suministrado cada segundo.

$$\Delta T_{\text{aire}} = 3.37 \times 10^3 \Delta T_{\text{agua}} \Big|_{2.39 \times 10^{-4}\frac{^oC}{Ws}}$$

es decir:

$$\begin{aligned}
\Delta T &= w_f t \big(3.37 \times 10^3\big)\big(2.39 \times 10^{-4}\big)\left[\tfrac{^oC}{Ws}\right] && (5) \\
&= w_f t \big(3.37 \times 10^3\big)\big(0.239\big)\left[\tfrac{^oC}{Ws}\right] && (6) \\
&= 80.54 \times 10^{-2}\left[\tfrac{^oC}{Ws}\right] w_f t && (7)
\end{aligned}$$

donde ΔT es el incremento de temperatura en un volumen de un litro de aire, w_f es la potencia de disipación del foco y t es el tiempo durante el cual la potencia es suministrada.

Es importante mencionar que la fórmula de la Ecuación (7) es únicamente una aproximación burda que se basa en asunciones bastante fuertes, como por ejemplo el considerar la densidad y el calor específico de los materiales como constante, lo cual es altamente inexacto en el caso de gases, los cuales tienden a expandirse y reducir su densidad al ser calentados. Si se intentara implementar un control de lazo abierto sería necesario un modelo matemático mucho más exacto[3] para que la temperatura real fuera lo más próxima posible al valor deseado; pero como se implementará un controlador de lazo cerrado, esta aproximación lineal será suficiente.

2.2. Control

Un sistema de control cerrado es aquel que utiliza un sensor para medir la salida del sistema, calcular el error respecto al valor deseado, y así suministrar la entrada necesaria para reducir el error lo más posible. Así cuando se habla de control cerrado de manera implícita se habla de sistemas retroalimentados: aquellos en los que la salida actual depende tanto de la entrada como del error entre el valor real y el valor deseado, tal como semuestra en la Figura 1.

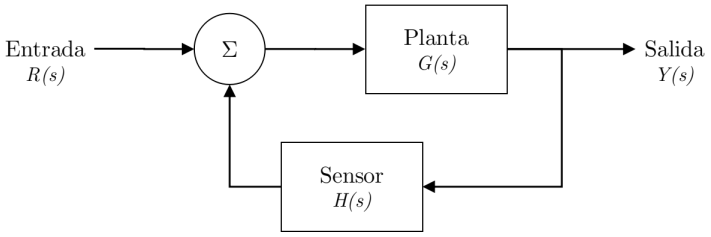

Figura 1: Un sistema en lazo cerrado

Para controlar un sistema con un control de lazo cerrado (en adelante *sistema de control*) se necesita de tres componentes fundamentales: un sensor que convierta la salida del sistema en una señal $B(s)$ que el controlador pueda interpretar, un amplificador operacional en modo resta que calcule el error $E(s)$ entre el valor deseado o entrada $R(s)$ y el valor sensado $B(s)$, y un dispositivo controlador $G_c(s)$ que transforme el error $E(s)$ en la entrada $V(s)$ que necesita el sistema controlado o *planta* $G_p(s)$ para producir la salida deseada $Y(s)$, tal como se ejemplifica en la Figura 2.

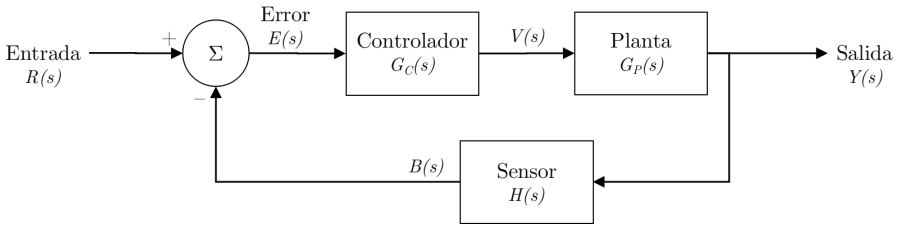

Figura 2: Un sistema de control en lazo cerrado

Debido a que los modelos de este tipo de sistemas son aproximados con funciones diferenciales, es común se utilicen transformadas como la de Laplace o Fourier para simplificar el análisis y poder modelar el controlador de forma un poco más fácil, además de que interesa mucho más conocer la respuesta en frecuencia del sistema.

Matemáticamente:

[3] Un modelo matemático más preciso tendría que, por lo menos, modelar el volúmen de aire con la ecuación del gas ideal $PV = nRT$ junto con las contribuciones de energía derivadas del recambio de aire en el medio no sellado, lo que invariablemente llevaría a una ecuación integrodiferencial.

$$Y(s) = E(s)G(s) \tag{a}$$

$$G(s) = G_c(s)G_p(s) \tag{b}$$

$$E(s) = R(s) - B(s) \tag{c}$$

$$B(s) = Y(s)H(s) \tag{d}$$

Sustituyendo las ecuaciones (c) y (b) en la ecuación (a):

$$\begin{aligned} Y(s) &= \big(G_c(s)G_p(s)\big)\big(R(s) - B(s)\big) \\ &= G_c(s)G_p(s)R(s) - G_c(s)G_p(s)B(s) \end{aligned} \tag{e}$$

reemplazando las ecuación (d) en la ecuación (e):

$$Y(s) = G_c(s)G_p(s)R(s) - G_c(s)G_p(s)\Big[Y(s)H(s)\Big] \tag{f}$$

agrupando y reordenando:

$$\begin{aligned} Y(s) &= G_c(s)G_p(s)R(s) - G_c(s)G_p(s)Y(s)H(s) \\ Y(s) + G_c(s)G_p(s)Y(s)H(s) &= G_c(s)G_p(s)R(s) \\ Y(s)\Big[1 + G_c(s)G_p(s)H(s)\Big] &= G_c(s)G_p(s)R(s) \\ Y(s) &= \frac{G_c(s)G_p(s)R(s)}{1 + G_c(s)G_p(s)H(s)} \end{aligned}$$

Con lo que finalmente se obtiene la *función de transferencia de lazo cerrado* $T(s)$ para un sistema retroalimentado con control cerrado:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{G_c(s)G_p(s)}{1 + G_c(s)G_p(s)H(s)} \tag{8}$$

Esta aproximación tiene un problema fundamental: la solución analítica de este tipo de ecuaciones suele ser computacionalmente muy costoso, lo que implica que se requerirán procesadores más potentes con un mayor consumo. Para resolver este problema, el análisis se realiza no en el contínuo, sino que se discretiza al sistema y se modela un control en tiempo discreto (o digital) usando la transformada Z. Con este tipo de modelado, la solución de integrales se convierte en suma de valores y las diferenciales en restas, lo que reduce enormemente el costo computacional de la solución del modelo del sistema, típicamente reduciéndolo a unos cuantos productos de matrices.

2.2.1. Control On/Off

Un *On/Off control* o control por encendido-apagado es la técnicas de control má simple que existe. Su principio de funcionamiento se basa en definir regiones de operación fuera de las cuales la operación de la planta se detiene. En su versión más simple, un controlador on-off hará funcionar a la planta a máxima potencia hasta que la salida registrada supere un valor umbral (*threshold*), momento en el que se reducirá la potencia de la planta, normalmente cortando el suministro de enegía o *apagándola*. La planta permanecerá apagada hasta que la salida del sistema se reduzca por debajo del umbral de operación, momento en el que el controlador volverá a poner la planta en operación a máxima potencia.

Este tipo de controlador se observa comúnmente en sistemas que responden lentamente a cambios y que tienden a almacenar mucha energía, como por ejemplo los calentadores de agua estacionarios donde el quemador se enciende hasta que la temperatura del agua se eleva hasta cierto nivel (ej. 80°C) y no se volverá a encender hasta que la temperatura registrada descienda por debajo de un umbral de operación (ej. 60°C). Además, el mecanismo de doble umbral del calentador evita que el quemador se esté encendiendo constantemente con cambios pequeños de temperatura.

Formalmente,

$$V(t) = \begin{cases} r(t) & \text{si } |e(t)| \geq k \\ 0 & \text{si } |e(t)| < k \end{cases} \tag{9}$$

El inconveniente principal de este tipo de controlador es que genera transiciones abruptas que tienden a desgastar los materiales, especialmente cuando se opera a altas frecuencias. Además, con este tipo de controlador no es posible modular la entrada del sistema dentro del rango de operación, dado que las transiciones sólo se producen en los umbrales, por lo que una operación suave es imposible. Es por esto que este tipo de controladores es raramente usado en aplicaciones industriales.

2.2.2. Control proporcional

Un control es proporcional cuando la salida $v(t)$ del controlador es proporcional al error $e(t)$:

$$v(t) = K_P e(t)$$

o en el dominio de la frecuencia:

$$V(s) = K_P E(s)$$

por lo tanto

$$G_c(s) = \frac{V(s)}{E(s)} = K_P \tag{10}$$

Al otorgar una señal de entrada proporcional al error, éste tipo de control opera como un amplificador de error que, al corregirse, afecta negativamente la respuesta transitoria del sistema. Es por esto que, a pesar de que un controlador P es fácil de implementar y ajustar, suele venir acompañado de otros elementos que compensen la amplificación del error y las variaciones transitorias.

Cuando se utiliza un control proporcional [1]:

- El tiempo de elevación experimenta una pequeña reducción.
- El máximo pico de sobreimpulso se incrementa.
- El amortiguamiento se reduce.
- El tiempo de asentamiento cambia en pequeña proporción.
- El error de estado estable disminuye con incrementos de ganancia.
- El tipo de sistema permanece igual.

2.2.3. Control integral

Un control es integral cuando la salida $v(t)$ del controlador es proporcional a la integral del error $e(t)$:

$$v(t) = K_I \int e(t)dt$$

o en el dominio de la frecuencia:

$$V(s) = \frac{K_I}{s} E(s)$$

por lo tanto

$$G_c(s) = \frac{V(s)}{E(s)} = \frac{K_I}{s} \tag{11}$$

El objetivo de un control integral es el de reducir el error de estado estable, aunque esta corrección viene a costa de sobrecorregir el error, por lo que la respuesta del sistema tiende a oscilatoria o incluso a volverse inestable. Es por esto que un controlador I suele venir acompañado de otros elementos que ayuden a estabilizar el sistema y a reducir las oscilaciones.

Cuando se combina con un controlador tipo proporcional para formar un control PI [1]:

- El amortiguamiento se reduce.
- El máximo pico de sobreimpulso se incrementa.
- Decrece el tiempo de elevación.
- Se mejoran los márgenes de ganancia y fase.
- El tipo de sistema se incrementa en una unidad.
- El error de estado estable mejora por el incremento del tipo de sistema.

2.2.4. Control derivativo

Un control es derivativo cuando la salida $v(t)$ del controlador es proporcional a la derivada del error $e(t)$:

$$v(t) = K_D \, \frac{d\, e(t)}{dt}$$

o en el dominio de la frecuencia:

$$V(s) = K_D \; s \; E(s)$$

por lo tanto

$$G_c(s) = \frac{V(s)}{E(s)} = K_D s \tag{12}$$

Debido a que la derivada del error respecto al tiempo es la velocidad del error, un control derivativo se anticipará a los errores antes de que estos se produzcan, amortiguando oscilaciones. Sin embargo, un controlador D es insensible a errores de estado estable, por lo que suele acompañarse de otros elementos que sí corrijan el error de estado estable.

Cuando se combina con un controlador tipo proporcional para formar un control PD [1]:

- El amortiguamiento se incrementa.
- El máximo pico de sobreimpulso se reduce.
- El tiempo de elevación experimenta pequeños cambios.
- Se mejoran el margen de ganancia y el margen de fase.
- El error de estado estable presenta pequeños cambios.
- El tipo de sistema permanece igual.

2.2.5. Control PID digital

Un control *proporcional-integral-derivativo* o PID incorpora las mejores características de los controles PI y PD, motivo por el cual ha sido uno de las técnicas de control continuo más utilizados en la industria por más de siete décadas [1–3].

Con base en las ecuaciones Ecuaciones (10) y (12) y subsección 2.2.3 el controlador sería de la forma:

$$G_c(s) = \frac{V(s)}{E(s)} = K_P + \frac{K_I}{s} + K_D s \tag{13}$$

Sin embargo, como se mencionó anteriormente en la Subsección 2.2, integrar señales en tiempo contínuo es computacionalmente muy costoso. Además, la señal del sensor pasará por un convertidor analógico-digital que discretizará la señal, por lo que tiene más sentido implementar un controlador discreto. Luego entonces, interesa conocer el valor (también discreto) que se proporcionará a la planta para obtener la señal deseada en el sensor. En el contínuo esta sería:

$$v(t) = K_P\, e(t) + K_I \int e(t)\, dt + K_D \frac{d\, e(t)}{dt}$$

Si se muestrea el error a una intervalos de tiempo constante con frecunecia lo suficientemente alta, es posible convertir la subsección 2.2.5 en una ecuación en diferencias sobre la k-ésima muestra tomada:

$$v[k] = K_P\, e[k] + K_I \sum_{j=0}^{k} e[j] + K_D \big(e[k] - e[k-1]\big)$$

donde

$e[k]$	el error actual
$e[k-1]$	el error anterior

Además, si se considera $P_I[k] = K_I \sum_{j=0}^{k} e[j]$ entonces:

$$\begin{aligned} P_I[k] &= K_I \sum_{j=0}^{k} e[j] \\ &= K_I\, e[k] + P_i[k-1] \end{aligned}$$

Así, aproximar un control digital para un sistema simple puede ser tan sencillo como almacenar en memoria los últimos valores sensados del error y el acumulado de la componente integral del PID, lo que reduce el cómputo a sólo unas cuantas operaciones aritméticas simples: sumas, restas y productos. No obstante, es necesario recordar que ésto no funcionará con sistemas que tengan una dinámica compleja, particularmente los no-lineales.

3. Material

Se asume que el alumno cuenta con un una Raspberry Pi con sistema operativo Raspbian e interprete de Python instalado. Se aconseja encarecidamente el uso de *git* como programa de control de versiones.

Además, el alumno necesitará los alambrados de la Práctica 6 y la Práctica 7.

3.1. Lista de componentes

- 1 Arduino UNO o Arduino Mega
- 1 sensor de temperatura LM35 en encapsulado TO-220 o TO-92
- 1 TRIAC BT138 o BT139
- 4 diodos 1N4007 o puente rectificador equivalente
- 2 diodos 1N914
- 1 optoacoplador MOC 3021
- 1 optoacoplador 4N25
- 1 foco incandescente (NO AHORRADOR NI LED)
- 1 resistencia de 15kΩ
- 1 resistencia de 18kΩ
- 1 resistencia de 12kΩ[4]
- 3 resistencia de 10kΩ
- 2 resistencia de 4k7Ω
- 1 resistencia de 1kΩ
- 2 resistencia de 470Ω
- 2 resistencia de 330Ω
- 1 condensador de 0.1μF
- 1 pecera o caja de acrílico transparente o de cartulina blanca

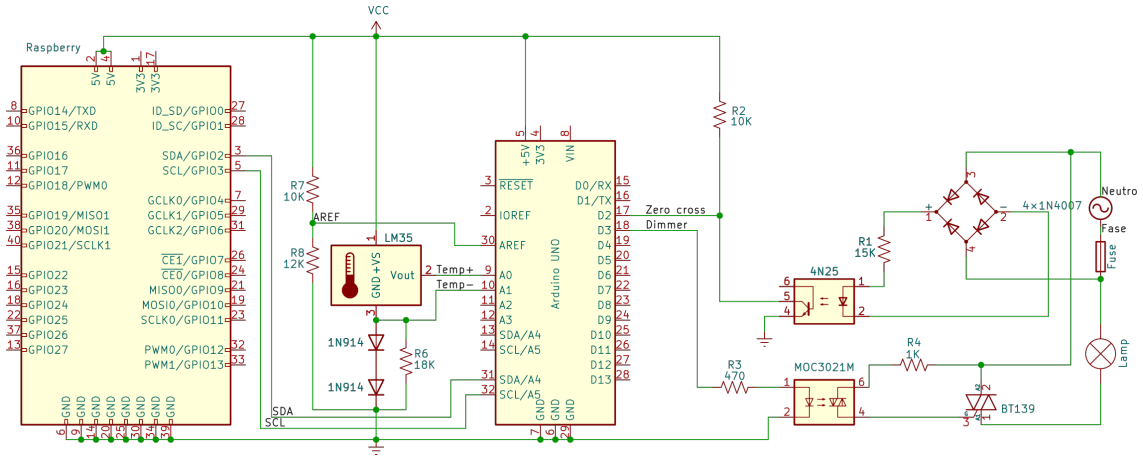

Figura 3: Circuito AC-DC optoacoplado completo para control cerrado de temperatura.

- 1 protoboard o circuito impreso equivalente
- 1 fuente de alimentación regulada a 5V y al menos 2 amperios de salida
- Cables y conectores varios

4. Instrucciones

1. Alambre el circuito de la Práctica 6.
2. Alambre el circuito de la Práctica 7.
3. Verifique que el sensor de temperatura registra las temperaturas adecuadamente.
4. Verifique que *dimmer* modula la potencia de la lámpara incandescente adecuadamente.
5. Integre ambos circuitos e implemente los controladores descritos en las Subsecciones 4.1 y 4.2.
6. Realice los experimentos propuestos en la sección 5.

4.1. Implementación un controlador On/Off

Alambre cuidadosamente los circuitos de las prácticas 6 y 7 para crear un circuito AC-DC optoacoplado completo para control cerrado de temperatura usando un foco incandescente operado con un LM35 y un circuito de variación de fase mediante detección de cruce por cero. El circuito completo alambrado debe verse como el de la Figura 3.

Importante

Asegúrese de verificar con un multímetro que el circuito de AC está debidamente aislado y que no se tienen valores mayores a 5V en el segmento de DC. De otro modo podría quemar su Arduino y su Raspberry Pi.

De acuerdo con el alambrado de la Figura 3, el Arduino ha de realizar tres funciones: i) digitalización del valor de temperatura analógico sensado por el LM35, ii) la detección del cruce por cero de la onda sinusoidal de la línea eléctrica, y iii) la modulación de potencia del foco incandescente mediante variación de fase ajustada por el retardo en el disparo del TRIAC. Al igual que en las prácticas anteriores, el Arduino estará configurado como dispositivo esclavo y estas operaciones estarán al servicio del dispositivo maestro: la Raspberry Pi. Así, una operación de lectura proporcionará al dispositivo maestro la temperatura sensada, mientras que una operación de escritura enviará la potencia con la que encenderá el foco como un valor flotante entre 0 y 100, quedando a cargo del Arduino el cálculo de la variación de fase y la sincronización de las operaciones de hardware.

[4]La resistencia de 12kΩ puede reemplazarse con resistencias de 13kΩ a 20kΩ dependiendo del voltaje de los diodos.

Nota: La implementación del código del Arduino se deja a cargo del estudiante.

Se puede proceder entonces a revisar el código de control que se ejecutará en la Raspberry Pi. En este caso se implementará un controlador On/Off tal como se describe en la Subsección 2.2.1.

Lo primero es incluir los paquetes necesarios y proveer cuatro parámetros importantes: 1) la dirección del dispositivo esclavo, 2) la temperatura deseada, 3) el umbral o valor de encendido, y 4) el umbral o valor de apagado tal como se muestra en el Código de Ejemplo 1.

Código ejemplo 1: `controller-on-off.py:9-20` — Parámetros

```
import smbus2
import struct
import time

DESIRED_TEMPERATURE = 50.0
MAX_POWER = 100.0
# Define thresholds:
ONTHRES = DESIRED_TEMPERATURE - 10
OFFTHRES = DESIRED_TEMPERATURE + 5

# Arduino's I2C device address
SLAVE_ADDR = 0x0A # I2C Address of Arduino 1
```

El controlador con los parámetros descritos anteriormente requerirá de dos funciones auxiliares: *readTemperature* (véase Código de Ejemplo 2) y *writePower* (véase Código de Ejemplo 3). La primer función corresponde a una operación de lectura en la cual el Arduino proporcionará la temperatura registrada por el sensor en grados centígrados codificado como un valor flotante (4 bytes en little endian). La segunda función es una operación de escritura en la cual la Raspberry Pi proporcionará al arduino el factor de potencia de 0 a 100 codificada como un valor flotante (4 bytes en little endian).

Código ejemplo 2: `controller-on-off.py:26-36` — Función *readTemperature*

```
def readTemperature():
  try:
    # Creates a message object to read 4 bytes from SLAVE_ADDR
    msg = smbus2.i2c_msg.read(SLAVE_ADDR, 4)
    i2c.i2c_rdwr(msg)  # Performs write
    data = list(msg)   # Converts stream to list
    ba = bytearray()
    for c in data:
      ba.append(int(c))
    temp = struct.unpack('<f', ba)
    # print('Received temp: {} = {}'.format(data, temp))
```

Código ejemplo 3: `controller-on-off.py:38-45` — Función *writePower*

```
  except:
    return None

def writePower(pwr):
  try:
    data = struct.pack('<f', pwr) # Packs number as float
    # Creates a message object to write 4 bytes from SLAVE_ADDR
    msg = smbus2.i2c_msg.write(SLAVE_ADDR, data)
```

Por último queda revisar la implementación del controlador mostrada en el Código de Ejemplo 4. El control de temperatura se ejecuta en un bucle infinito de tres partes dentro de la función principal *main*. 1) Primero se lee la temperatura en °C del arduino, 2) después esta se compara con los umbrales. Si la temperatura es menor al valor de encendido la lámpara se enciende a máxima potencia y permanecerá encendida hasta que se supera el valor de apagado. Por último, 3) el controlador entra en un estado de espera (1 segundo) donde no se realiza ninguna acción. En cualquier otro caso no se hace nada.

Código ejemplo 4: `controller-on-off.py:47-67` — Función *main*

```
except:
print("-= On/Off controller =-")
print("  Desired temperature: {:0.2f}°C".format(DESIRED_TEMPERATURE))
while True:
  try:
    current = readTemperature()
    print("\r  Current temperature: {:0.2f}°C".format(current), end="")

    # If temperature is lower than on threshold, turn on (MAX power).
    if current <= ONTHRES:
      writePower(MAX_POWER)
    # If temperature is greater than off threshold, turn off.
    elif current >= OFFTHRES:
      writePower(0)
    # 1 sec so controller can act.
    sleep(1)
```

4.2. Implementación un controlador P

Alambre cuidadosamente los circuitos de las prácticas 6 y 7 para crear un circuito AC-DC optoacoplado completo para control cerrado de temperatura usando un foco incandescente operado con un LM35 y un circuito de variación de fase mediante detección de cruce por cero. El circuito completo alambrado debe verse como el de la Figura 3.

Importante

Asegúrese de verificar con un multímetro que el circuito de AC está debidamente aislado y que no se tienen valores mayores a 5V en el segmento de DC. De otro modo podría quemar su Arduino y su Raspberry Pi.

El controlador proporcional a implementar opera de manera muy similar al controlador On/Off implementado en la Subsección 4.1, por lo que también requerirá de las funciones auxiliares *readTemperature* (véase Código de Ejemplo 2) y *writePower* (véase Código de Ejemplo 3), y operará dentro de un bucle infinito. Sin embargo, a diferencia del controlador anterior, el controlador P calcula primero el error (la diferencia entre temperatura deseada y la temperatura medida) y multiplica éste por la constante de contorl proporcional `PI` para obtener el factor de potencia a suministrar a la planta que llevará al sistema producir la salida deseada, tal como se muestra en el Código de Ejemplo 5.

Código ejemplo 5: `controller-p.py:46-64` — Función *main*

```
except:
print("-= P controller =-")
print("  Desired temperature: {:0.2f}°C".format(DESIRED_TEMPERATURE))
while True:
  try:
    current = readTemperature()
    power = KP * error
    writePower(power)
    # Wait so controller can act.
    sleep(0.25)
```

Nótese que a diferencia del controlador On/Off el controlador proporcional realiza acciones siempre. Además, el tiempo de espera es mucho menor.[5]

[5]Este tiempo de espera es necesario para permitir a la planta actuar, ya que la temperatura se eleva con relativa lentitud en comparación con otros cambios físicos. En general este podrá reducirse hasta un mínimo de apriximadamente 9ms, periodo de media onda de la sinusoidal de AC en una línea de 60Hz.

5. Experimentos

1. [2pts] Implemente el controlador on-off descrito en las Subsección 4.1 y apéndice A y verifique que la temperatura dentro de la caja se mantenga en 50°C ± 10°C. Reporte el conjunto de valores *threshold* utilizados por el controlador.

2. [2pts] Con base en la teoría de la Subsección 2.2.2 y el código presentado en el Apéndice A, desarrolle un controlador proporcional que mantenga la temperatura dentro de la caja entre 45°C y 55°C. Reporte el valor de K_P y el valor medio de potencia suministrado al elemento calefactor (planta).

3. [2pts] Modifique el programa anterior para que el controlador mantenga la temperatura ingresada por línea de comandos con una incertidumbre de ±5°C. El rango de operación es entre 30°C y 100°C.[6]

4. [2pts] Con base en la teoría de la Subsecciones 2.2.2 y 2.2.3 y el código presentado en el Apéndice A, desarrolle un controlador proporcional-integral (PI) que mantenga la temperatura dentro de la caja entre 45°C y 55°C. Reporte los valores de K_P y K_I así como el valor medio de potencia suministrado al elemento calefactor (planta).

5. [2pts] Tomando como condición inicial T_0 la temperatura ambiente y $T_f = 50^oC$, grafique la respuesta del sistema $y(t) = T(t)$ con cada uno de los controladores implementados en el intervalo $0s \leq t \leq 180s$, reportando las constante K_P utilizada.
   - Controlador On/Off (1 pt).
   - Controlador P. Reporte la K_P (1 pt).

Puntos Extra

- [+2pts] Con base en la teoría de las Subsecciones 2.2.2 y 2.2.4 y el código presentado en el Apéndice A, desarrolle un controlador proporcional-derivativo (PD) que mantenga la temperatura dentro de la caja entre 45°C y 55°C.
- [+2pts] Con base en la teoría de las Subsecciones 2.2.2 a 2.2.5 y el código presentado en el Apéndice A, desarrolle un controlador proporcional-integral-derivativo (PID) que mantenga la temperatura dentro de la caja entre 45°C y 55°C. Reporte los valores de K_P, K_I y K_D así como el valor medio de potencia suministrado al elemento calefactor (planta).
- [+2pts] Optimice las constantes del controlador para que el error $\left|e[k]\right| \leq 2^oC$.
- [+2pts] Con base en lo aprendido, modifique el código del punto 3 para que la Raspberry Pi sirva una página web donde se pueda modificar con un control gráfico la temperatura deseada del sistema.
- [+2pts] Modifique el servidor web implementado para que éste muestre una gráfica en tiempo real con la temperatura registrada por el sensor del sistema.
- [+5pts] Tomando como base la gráfica realizada en el punto 5, anexe las gráficas de la respuesta del sistema $y(t) = T(t)$ con cada uno de los controladores implementados con $0s \leq t \leq 180s$, reportando las constantes utilizadas:
  - Controlador PI. Reporte las K_P, K_I utilizadas (1 pt).
  - Controlador PD. Reporte las K_P y K_D utilizadas (2 pts).
  - Controlador PID. Reporte las K_P, K_I y K_D utilizadas (2 pts).

[6] El valor mínimo de operación no podrá ser menor a la temperatura ambiente.

Referencias

[1] Ricardo Hernández Gaviño. *Introducción a los sistemas de control: conceptos, aplicaciones y simulación con matlab*. Pearson Educación, 2010.

[2] Aidan O'dwyer. *Handbook of PI and PID controller tuning rules*. World Scientific, 2009.

[3] Katsuhiko Ogata. *Ingeniería de control moderna*. Pearson Educación, 2003.

[4] *MOC3021 Random-Phase Optoisolator TRIAC Driver Output*. Fairchild Semiconductors, 8 2010. Revised: January, 2010.

[5] *4N25 Optocoupler, Phototransistor Output, with Base Connection*. Vishay Semiconductors, 8 2010. Revised: January, 2010.

[6] The Arduino Project. Introduction to the arduino board, 2020. https://www.arduino.cc/en/reference/board, Last accessed on 2020-03-01.

[7] I2C Info: A Two-wire Serial Protocol. I2c info – i2c bus, interface and protocol, 2020. https://i2c.info/, Last accessed on 2020-03-01.

[8] Muhammad H Rashid. Power electronic, devices, circuits, and applications. *Handbook, Second Edition, Burlington*, 2006.

[9] Carlos Valdivia Miranda. *Sistemas de control continuos y discretos*. Editorial Paraninfo, 2012.

[10] Katsuhiko Ogata. *Sistemas de control en tiempo discreto*. Pearson educación, 1996.

A. Programa Ejemplo: `controller-on-off.cpp`

src/controller–on–off.py

```
DESIRED_TEMPERATURE = 50.0
MAX_POWER = 100.0
# Define thresholds:
ONTHRES = DESIRED_TEMPERATURE - 10
OFFTHRES = DESIRED_TEMPERATURE + 5

# Arduino's I2C device address
SLAVE_ADDR = 0x0A # I2C Address of Arduino 1

# Initialize the I2C bus;
# RPI version 1 requires smbus.SMBus(0)
i2c = smbus2.SMBus(1)

def readTemperature():
  try:
    # Creates a message object to read 4 bytes from SLAVE_ADDR
    msg = smbus2.i2c_msg.read(SLAVE_ADDR, 4)
    i2c.i2c_rdwr(msg)  # Performs write
    data = list(msg)   # Converts stream to list
    ba = bytearray()
    for c in data:
      ba.append(int(c))
    temp = struct.unpack('<f', ba)
    # print('Received temp: {} = {}'.format(data, temp))
    return temp
  except:
    return None

def writePower(pwr):
  try:
    data = struct.pack('<f', pwr) # Packs number as float
    # Creates a message object to write 4 bytes from SLAVE_ADDR
    msg = smbus2.i2c_msg.write(SLAVE_ADDR, data)
    i2c.i2c_rdwr(msg)  # Performs write
  except:
    pass

def main():
  print("-= On/Off controller =-")
  print("  Desired temperature: {:0.2f}°C".format(DESIRED_TEMPERATURE))

  while True:
    try:
      # Read and report current temperature
      current = readTemperature()
      print("\r  Current temperature: {:0.2f}°C".format(current), end="")

      # If temperature is lower than on threshold, turn on (MAX power).
      if current <= ONTHRES:
        writePower(MAX_POWER)
      # If temperature is greater than off threshold, turn off.
      elif current >= OFFTHRES:
        writePower(0)
      # 1 sec so controller can act.
      sleep(1)
    except:
      print("\tError!")
      writePower(0)

if __name__ == '__main__':
  main()
```

B. Programa Ejemplo: `controller-p.cpp`

src/controller–p.py

```
DESIRED_TEMPERATURE = 50.0
KP = 1

# Arduino's I2C device address
SLAVE_ADDR = 0x0A # I2C Address of Arduino 1

# Initialize the I2C bus;
# RPI version 1 requires smbus.SMBus(0)
i2c = smbus2.SMBus(1)



def readTemperature():
  try:
    # Creates a message object to read 4 bytes from SLAVE_ADDR
    msg = smbus2.i2c_msg.read(SLAVE_ADDR, 4)
    i2c.i2c_rdwr(msg)  # Performs write
    data = list(msg)   # Converts stream to list
    ba = bytearray()
    for c in data:
      ba.append(int(c))
    temp = struct.unpack('<f', ba)
    # print('Received temp: {} = {}'.format(data, temp))
    return temp
  except:
    return None

def writePower(pwr):
  try:
    data = struct.pack('<f', pwr) # Packs number as float
    # Creates a message object to write 4 bytes from SLAVE_ADDR
    msg = smbus2.i2c_msg.write(SLAVE_ADDR, data)
    i2c.i2c_rdwr(msg)  # Performs write
  except:
    pass

def main():
  print("-= P controller =-")
  print("  Desired temperature: {:0.2f}°C".format(DESIRED_TEMPERATURE))

  while True:
    try:
      # Read and report current temperature
      current = readTemperature()
      print("\r  Current temperature: {:0.2f}°C".format(current), end="")
      # Calculate error: E(s) = R(s) - B(s)
      error = DESIRED_TEMPERATURE - current
      # Calculate plant input: V(s) = KP × E(s)
      power = KP * error
      writePower(power)
      # Wait so controller can act.
      sleep(0.25)
    except:
      print("\tError!")
      writePower(0)

if __name__ == '__main__':
  main()
```