



Vaporizador

Basado en Arduino

Máster en Ingeniería Electromecánica

Microcontroladores y Lógica Programable



Jaime Salazar Lahera

junio 2017

Vaporizador Basado en Arduino

0 General

Se trata de un vaporizador de muestras vegetales o aceites esenciales que volatiliza el material en un horno pequeño. Consiste principalmente de un Arduino Uno R3, un Arduino Uno Wifi, un horno, y un sensor de temperatura (Figura 0.0, Anexo?).

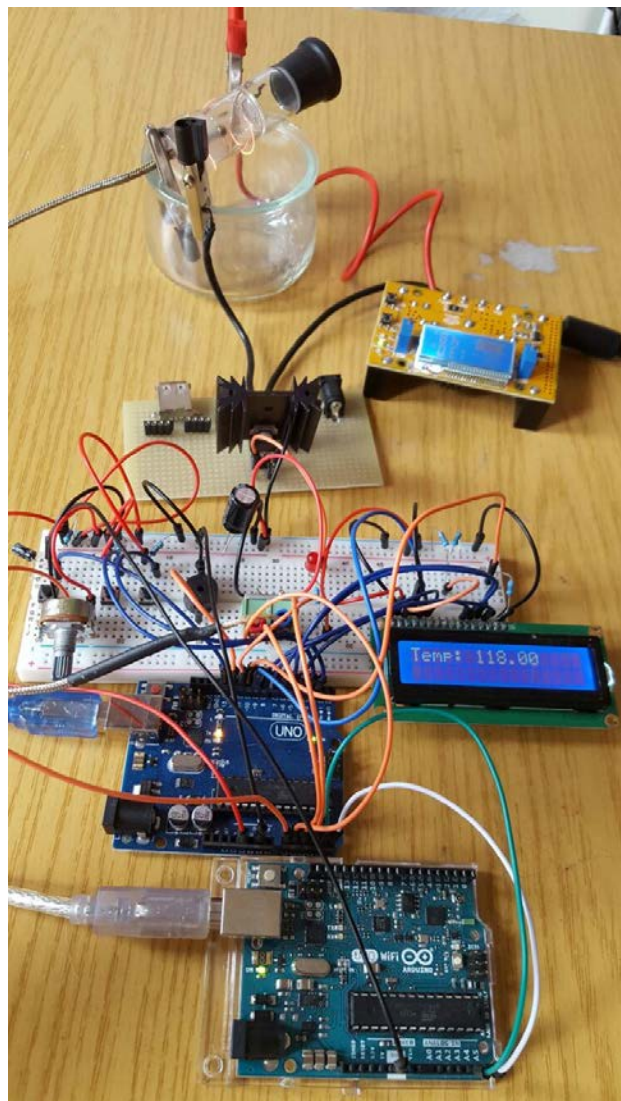


Fig. 0.0 Prototipo

El Arduino Uno Wifi recibe las señales de los pulsadores, del regulador de temperatura, y de los sensores para controlar un zumbador, una pantalla LCD, y un horno (Figura 0.1).

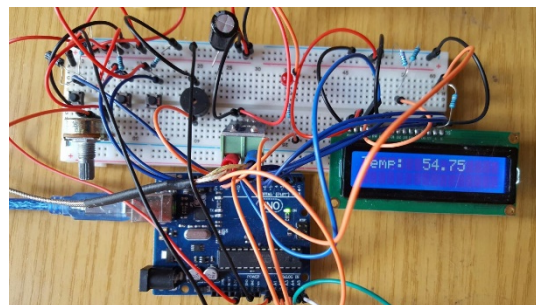


Fig. 0.1 Horno

El Arduino Uno Wifi, basado en un módulo ESP8266, facilita las comunicaciones con una interfaz externa (ordenador, teléfono móvil, etc.). El horno es un compartimento de cristal que calienta el aire en su interior y por convección se arrastra los vapores de la muestra hacia el exterior (Figura 0.1). Un sensor de temperatura ayuda a mantener un seguimiento de la temperatura del horno.

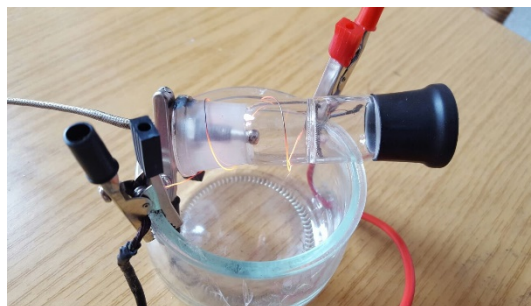


Fig. 0.2 Horno

La Figura 0.3 es un esquema final desarrollado en la página www.circuits.io. Su selección limitada de componentes obliga a representar, por ejemplo, al sensor de temperatura de 5 patillas con sólo 3 o el transistor de forma TO-220 como un pequeño transistor PN2222. No obstante, se presenta como una idea básica del circuito en cuestión.

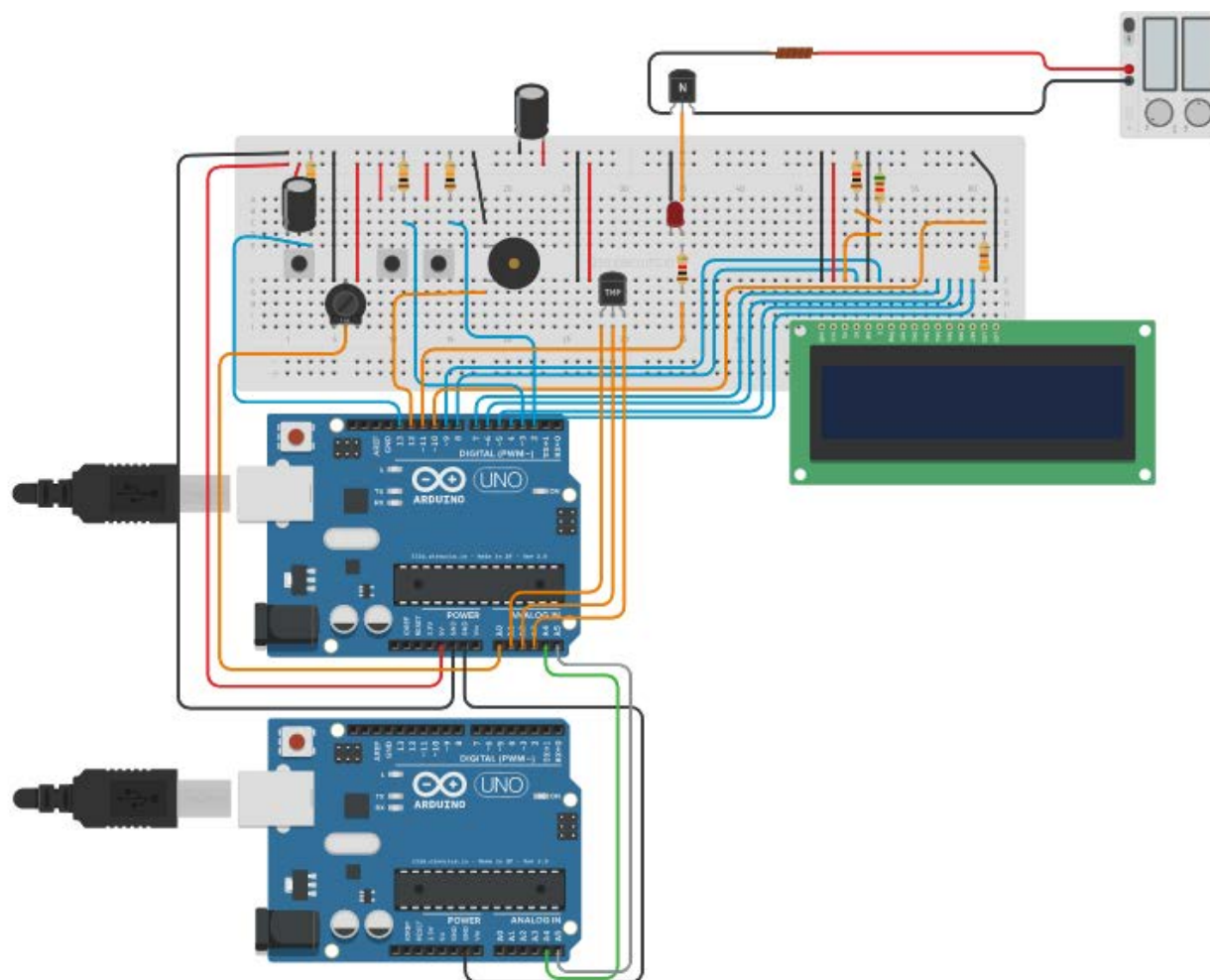


Fig. 0.3 Esquema final

Las etapas principales del proyecto son las siguientes:

1. Pulsador ON	4
2. Pantalla LCD	7
3. Regulador de temperatura	8
4. Sensor de temperatura	11
5. Horno	18
6. Control PID de temperatura	22
7. Comunicaciones	30

Cada etapa resume primero el circuito y su código final (*Descripción*), seguido por apuntes complementarios sobre su desarrollo (*Proceso*), y las fuentes utilizadas (*Referencias*).

1 Pulsador ON

Descripción

El pulsador ON enciende o apaga el vaporizador dependiendo de si el sistema actualmente está apagado o encendido, respectivamente. Con la función `attachPinChangeInterrupt()` de la librería `PinChangeInt.h`¹, se prepara una interrupción en la patilla digital 11, sin tener que limitarse a las patillas 2 y 3 que se utilizan por defecto para interrupciones externas. De esta forma, se reservan estas dos patillas para los pulsadores de temperatura. La interrupción se lanza con un flanco positivo del pulsador ON mediante su parámetro 'RISING' y llama a la función `onOff()`. El rebote se ha eliminado con un condensador electrolítico de 1 μ F.

Dentro de `onOff()` se encuentran `on()` y `off()`, de formato y propósito prácticamente opuestos. Respectivamente, éstas activan o desactivan al temporizador de seguridad, emiten un tono alto y corto o bajo y largo mediante `tone()`³, encienden o apagan la pantalla LCD, y establecen el valor de la marca booleana `ON` (Figura 1.0).

Dentro del `loop()`, la marca `ON` habilita al horno, la pantalla LCD, y el envío de datos de temperatura y, además, mediante con `millis()` se "ralentiza" la frecuencia de ejecución de estas funciones para reducir el gasto energético (Figura 1.1). Se permite recibir el control remoto aún apagado por si llega un commando de encendido.

```

6 void onOff()
7 {
8     if(!ON)
9     {
10         on();
11     }
12     else
13     {
14         off();
15     }
16 }
17
18 void on()
19 {
20     TIMSK1 |= (1 << OCIE1A); // empezar la temporización de seguridad
21     tone(ZUMB, 400, 200);    // pitido alto y corto
22     digitalWrite(LCD, HIGH); // encender pantalla LCD
23     ON = true;
24 }
25
26 void off()
27 {
28     TIMSK1 &= ~(1 << OCIE1A); // detener la temporización de seguridad
29     tone(ZUMB, 200, 400);    // pitido bajo y largo
30     digitalWrite(LCD, LOW);  // apagar pantalla LCD
31     ON = false;
32 }

```

Fig. 1.0 Función `onOff()`

```

87 void loop()
88 {
89     recibirControl(); // recibir órdenes de control a través de I2C
90
91     // Sólo ejecutar al estar encendido
92     if (ON)
93     {
94         // Ralentizar el loop()
95         if (millis() - tRef >= 5000)
96         {
97             calentarHorno(PWM_PID()); // calentar horno según la salida PWM del PID
98
99             LCDtemp(temp()); // mostrar la temperatura actual
100
101             enviarTemp(); // enviar temperatura por I2C
102
103             tRef = millis();
104         }
105     }
106 }

```

Fig. 1.1 Función `loop()`

Como medida de seguridad, nada más encenderse el vaporizador se inicia una temporización de 30 minutos con el temporizador 1, tras los cuales se apaga el sistema. En el `setup()` la función `inicializarISR()` inicializa a esta ISR (Figura 1.2) deshabilitando primero a las interrupciones globales y reseteando los registros TCCR1A y TCCR1B. A continuación, se establece una comparación con un valor de 1 segundo, el modo CTC del temporizador, y un pre-escalado de 1024. Finalmente se habilita esta interrupción, así como las interrupciones globales⁴ (Figura 1.1). En particular, OCR1A se compara constantemente con TCNT1, que almacena la cuenta actual del temporizador, ambos siendo registros de 16 bits⁵. El valor de OCR1A = 15624 surge de la siguiente fórmula, donde $t = 1$ es el tiempo deseado de 1 segundo y $r = 6,4 \cdot 10^{-5}$ segundos es la resolución del temporizador⁴:

$$t = r \cdot (OCR1A + 1) \Rightarrow OCR1A = \frac{t}{r} - 1 = \frac{1 \text{ seg}}{6,4 \cdot 10^{-5} \text{ seg}} - 1 = 15624$$

```

9 void inicializarISR()
10 {
11     cli();           // deshabilitar interrupciones globales
12     TCCR1A = 0;      // resetear el registro TCCR1A entero a 0
13     TCCR1B = 0;      // resetear el registro TCCR1B entero a 0
14
15     OCR1A = 15624;    // establecer una comparación con este valor, que asegura interrupciones cada 1 segundo
16     TCCR1B |= (1 << WGM12); // modo CTC
17     TCCR1B |= (1 << CS10);  // prescaler de 1024
18     TCCR1B |= (1 << CS12);  // prescaler de 1024
19     TIMSK1 |= (1 << OCIE1A); // habilitar la interrupción de comparación
20     sei();               // habilitar interrupciones globales
21 }

```

Fig. 1.2 Inicialización de la ISR

Con un TCCR1A completamente reseteado y un TCCR1B cuyo bit WGM12 se pone a 1, se establece la combinación WGM1[3:0] = 0100, codificando el modo 4 de la Tabla 20-6 (modo CTC) con OCR1A como TOP y su actualización inmediata⁵.

Con un TCCR1B generalmente

reseteado y una activación de sus bits CS10 y CS12, los tres primeros bits del registro TCCR1B son de la forma de 101, codificando para un pre-escalado de 1024. Finalmente, se habilita esta interrupción de comparación en la máscara TIMSK1 al poner su bit OCIE1A a 1 y se habilitan las interrupciones generales para finalizar la inicialización de la interrupción (Figura 1.2).

Esta inicialización lanza la ISR cada segundo, añadiendo 1 al contador de segundos. Cada 1800 interrupciones (cada 30 minutos) se reinicia la cuenta y se llama a `off()` para apagar el vaporizador (Figura 1.3), si el usuario se deja el vaporizador encendido, a la media hora se apaga el dispositivo y se evita también el gasto energético.

```

23 // Cada 1800 interrupciones se lanza la función off()
24 ISR(TIMR1_COMPA_vect)
25 {
26     segundos++;           // añadir 1 segundo al contador
27
28     if (segundos == tiempoApagado) // si pasan 1800 segundos, apagar el sistema
29     {
30         segundos = 0;      // reiniciar la cuenta
31         off();             // apagar el sistema
32     }
33 }

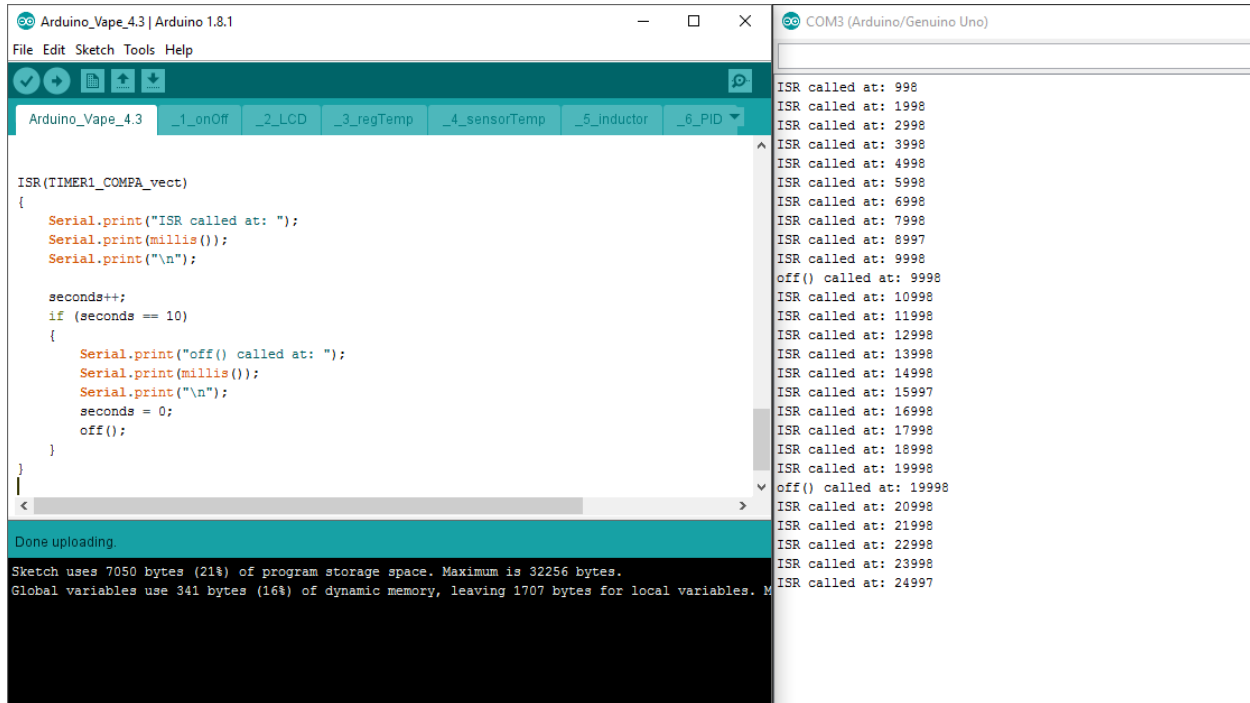
```

Fig. 1.3 ISR

Proceso

Para reducir el uso de materiales, es preferible eliminar el rebote por medio del software pero las funciones `millis()` o `delay()` no son muy compatibles con las interrupciones⁶.

También era preferible encontrar un valor del registro OCR1A que produjese una interrupción a los 30 minutos, pero el límite de los temporizadores de unos pocos segundos² ha resultado en el método de interrupciones de un 1 segundo descrito anteriormente. Haciendo pruebas de 10 segundos se observa en la Figura 1.4 que la primera interrupción se lanza a los 2 milisegundos, probablemente el tiempo que transcurre entre el encendido del Arduino, su recorrido por `setup()`, y los primeros ciclos del `loop()`.



The screenshot shows the Arduino IDE interface. The main window displays the code for an Interrupt Service Routine (ISR) named `ISR(TIM1_COMPA_vect)`. The code includes serial printing of the time when the ISR is called and when the `off()` function is called, along with a counter `seconds` that increments until it reaches 10. The serial monitor on the right shows the output of the program, displaying the times when the ISR is called and when `off()` is called, confirming the timing of the interrupts.

```
ISR(TIM1_COMPA_vect)
{
  Serial.print("ISR called at: ");
  Serial.print(millis());
  Serial.print("\n");

  seconds++;
  if (seconds == 10)
  {
    Serial.print("off() called at: ");
    Serial.print(millis());
    Serial.print("\n");
    seconds = 0;
    off();
  }
}
```

Serial Monitor Output:

```
ISR called at: 998
ISR called at: 1998
ISR called at: 2998
ISR called at: 3998
ISR called at: 4998
ISR called at: 5998
ISR called at: 6998
ISR called at: 7998
ISR called at: 8997
ISR called at: 9998
off() called at: 9998
ISR called at: 10998
ISR called at: 11998
ISR called at: 12998
ISR called at: 13998
ISR called at: 14998
ISR called at: 15997
ISR called at: 16998
ISR called at: 17998
ISR called at: 18998
ISR called at: 19998
off() called at: 19998
ISR called at: 20998
ISR called at: 21998
ISR called at: 22998
ISR called at: 23998
ISR called at: 24997
```

Fig. 1.4 Prueba del ISR

Ya que el Arduino en principio sigue contando y ejecutando las interrupciones pese a la llamada a `off()`, conviene desactivar la máscara TIMSK1 en dicha función para reducir cualquier gasto energético asociado a estas interrupciones innecesarias, aunque se trate de un gasto mínimo.

Referencias

1. <http://playground.arduino.cc/Main/PinChangeInterrupt>
2. <https://brainy-bits.com/blogs/tutorials/make-any-arduino-pin-an-interrupt-pin>
3. <https://www.arduino.cc/en/reference/tone>
4. <http://www.engblaze.com/microcontroller-tutorial-avr-and-arduino-timer-interrupts/>
5. http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf
6. <https://arduino.stackexchange.com/questions/22212/using-millis-and-micros-inside-an-interrupt-routine>
7. <https://forum.arduino.cc/index.php?topic=419606.0>

2 Pantalla LCD

Descripción

La pantalla LCD (16x2) se enciende con el pulsador ON ya que éste manda 5 V al ánodo de la pantalla (Sección 1). Inmediatamente en el `setup()` la función `LCDinit()` muestra una pantalla de bienvenida (Figura 2.1). En el `loop()` cada 5 segundos se llama a la función `LCDtemp()` (Figura 1.1) para que muestre en pantalla el resultado de la función `temp()` proveniente del sensor de temperatura (Sección 4). De esta forma, se refresca la pantalla al mismo ritmo al que se actualiza la temperatura actual. Por último, la pantalla también muestra el valor de temperatura deseado cada vez que se modifica mediante el regulador de temperatura (Sección 3).

```
6 // Bienvenida
7 void LCDinit()
8 {
9     lcd.setCursor(3,0);
10    lcd.print("Bienvenid@");
11 }
```

Fig. 2.1 Pantalla de bienvenida

```
8 #include "PinChangeInt.h"
9 #include <PID_v1.h>
10 #include <MsTimer2.h>
11 #include <LiquidCrystal.h>
12 #include <Wire.h>
13
14
15 // Inicializar los pins del LCD
16 LiquidCrystal lcd(9, 8, 7, 6, 5, 4);
```

Fig. 2.2 Pantalla de bienvenida

Con la librería de `LiquidCrystal.h` se inicializa la pantalla LCD con su 6 patillas (Figura 2.2). En el fichero “_2_LCD.ino” se crea la función `LCDtemp()`, que recibe el valor de la temperatura actual según el sensor de temperatura y la muestra en pantalla. Varía la posición del valor según su número de dígitos para alinear la coma decimal (Figura 2.3).

```
14 // Mostrar la temperatura actual
15 void LCDtemp(double temp)
16 {
17     if (ON)
18     {
19         lcd.clear();           // borra la pantalla de bienvenida la primera vez que se enciende el vaporizador
20     }
21
22     lcd.setCursor(0,0);
23     lcd.print("Temp: ");
24
25     // Posicionar al valor de la temperatura actual según su número de dígitos
26     if (temp < 100)
27     {
28         lcd.setCursor(7,0);
29     }
30     else
31     {
32         lcd.setCursor(6,0);
33     }
34
35     lcd.print(temp);           // imprimir la temperatura actual
36 }
```

Fig. 2.3 Temperatura actual en pantalla

Proceso

La luz de fondo de la pantalla se ha establecido con un divisor de tensión para simplificar los materiales, aunque un potenciómetro produce un mejor contraste. Los valores de las resistencias del divisor se basaron en las medidas del divisor formado por el potenciómetro. Al estar las demás patillas ocupadas con los demás componentes, no se ha podido utilizar una patilla PWM para controlar la luz de fondo, aunque sería preferible para reducir aún más el uso de materiales.

El parpadeo de la pantalla de temperatura actual se solucionó con la ralentización del `loop()`, ya que la función `lcd.print()` estaba intentando imprimir y re-imprimir a un ritmo excesivo¹. Los problemas a la hora de mostrar la consigna se detallan en la siguiente sección ya que están muy ligados a esa interrupción.

Referencias

1. <http://forum.arduino.cc/index.php?topic=138249.0>

3 Regulador de Temperatura

Descripción

Para establecer la temperatura deseada se puede utilizar pulsadores o un potenciómetro. Con los pulsadores se incrementa o decremента la consigna actual al lanzar las interrupciones externas 0 y 1, asociadas a las patillas 2 y 3, que llaman a las funciones `incrementSP()` y `decrementSP()`, respectivamente. Estas funciones casi idénticas primero deshabilitan al resto de funciones del sistema mediante el marcador `ON`. De esta forma, mientras se esté modificando la consigna ni el horno ni su controlador PID están activos. La función inmediatamente muestra el valor de la consigna actual y, mientras el usuario siga presionando el pulsador en cuestión, aumenta o disminuye el valor de `SPactual`, mientras se actualiza el valor de la consigna en pantalla con `LCDtemp(SPactual)` (Figura 3.0). El bucle `while()` junto con `delay()` resulta en un barrido de temperaturas mientras se mantenga presionado al pulsador. El uso de estas funciones dentro de una interrupción se detalla en la siguiente sección (*Proceso*). Una vez se deja de pulsar el botón empieza una cuenta en el temporizador 2 gracias a la librería de `MsTimer2.h`¹. Ésta establece una resolución de 1 milisegundo en T2 y permite establecer en el `setup()` una interrupción cada 800 milisegundos que llame a la función `habilitar()`. Ahora se puede iniciar la cuenta en otras zonas del código con `MsTimer2::start()`, que aparece en las funciones del fichero “_3_SP.ino”.

```
6 void incrementSP()
7 {
8     ON = false;           // pausar las funciones del vaporizador
9     LCDtemp(SPactual);
10
11     while (digitalRead(TEMPAS) == HIGH)
12     {
13         SPactual++;
14         LCDtemp(SPactual);
15         delay(26000);
16     }
17     MsTimer2::start();    // iniciar cuenta para rehabilitar la marca ON
18 }
19 }
```

Fig. 3.0 Incremento de temperatura por pulsador

```
73 void habilitar()
74 {
75     ON = true;
76     tone(ZUMB, 600, 200); // confirmar la nueva consigna
77     lcd.clear();
78     tRef = intervalo;     // acelerar la transición
79     MsTimer2::stop();     // detener el temporizador
80 }
```

Fig. 3.1 Función habilitar()

Una vez transcurridos los 800 milisegundos, la función `habilitar()` (Figura 3.1) re-establece al marcador `ON`, por lo que se podrá resumir las demás funciones del vaporizador. De esta forma, el horno y su PID vuelven a arrancar cuando ya se ha decidido la nueva consigna. La función emite un zumbido para confirmar el cambio de consigna, borra la pantalla LCD de la consigna y acelera la transición de vuelta a pantalla de la temperatura actual. Por último, detiene la temporización.



Por otro lado, se puede variar la consigna con un potenciómetro de 10K, cuyo pin A0 genera una interrupción preparada en el `setup()` con la función `attachPinChangeInterrupt()` que detecta un cambio en el pin del regulador para llamar a la función `regTemp()` (Figura 3.2). Esta función interpola la lectura analógica del potenciómetro entre 180 y 200 °C con la función `map()`. En seguida compara lecturas previas y actuales y, con la ayuda de un contador que actúa como temporizador, detecta cuando ha pasado un determinado “tiempo” desde la última vez que se movió el regulador. El uso de `while()` también se detalla en el siguiente sub-apartado.

```
40 void regTemp()
41 {
42     ON = false;          // pausar las funciones del vaporizador
43     LCDtemp(SPactual);
44
45     int SPactual1;
46     int SPactual2;
47     int cont = 0;
48
49     while (cont < 250 && SPactual1 == SPactual2)    // comparar valores del potenciómetro
50     {
51         SPactual2 = map(analogRead(REGTEMP), 0, 1023, 180, 280);    // interpolar lectura analógica entre 180 y 280 °C
52         LCDtemp(SPactual2);    // mostrar último valor
53
54         SPactual1 = SPactual2;    // almacenar última lectura
55         cont++;    // contador sirve como temporizador
56     }
57
58     SPactual = SPactual2;    // actualizar consigna con último valor de potenciómetro
59
60     MsTimer2::start();
61 }
```

Fig. 3.2 Incremento de temperatura por potenciómetro

La última forma de modificar la consigna es mediante la interfaz remota a través de la comunicación I²C (Sección 7). Para ello se recibe el valor de la consigna como parámetro de la función `enterSP()`, que actualiza a la variable global `SPactual` (Figura 3.3).

```
65 void enterSP(int SPdeseada)
66 {
67     SPactual = SPdeseada;
68     LCDtemp(SPactual);
69     MsTimer2::start();
70 }
```

Fig. 3.3 Asignación de la consigna por control remoto

Proceso

Las interrupciones de los pulsadores aumentan o disminuyen la consigna de uno en uno con cada pulsación. El uso de `while()` dentro de la interrupción, permite que además, tras un flanco inicial, uno pulso constante produzca un rápido ascenso o descenso de la consigna hasta que se deje de presionar el pulsador. Sin un `delay()`, este bucle produce ascenso o descenso demasiado rápido y un valor de 2600 como parámetro consigue un barrido de temperaturas a un ritmo satisfactorio. No se ha podido conseguir el mismo efecto con las funciones `millis()` ni `micros()`, las cuales no son muy útiles dentro de una interrupción².

El uso de `while()` dentro de la interrupción del potenciómetro permite la comparación continua de valores durante suficiente tiempo. De esta forma, el procesador no está constantemente saltando entre el `loop()` y la interrupción, lo cual en las pruebas iniciales resultaba en saltos erráticos en la consigna al mover el regulador. La Figura 3.4 muestra una prueba inicial de las lecturas analógicas.

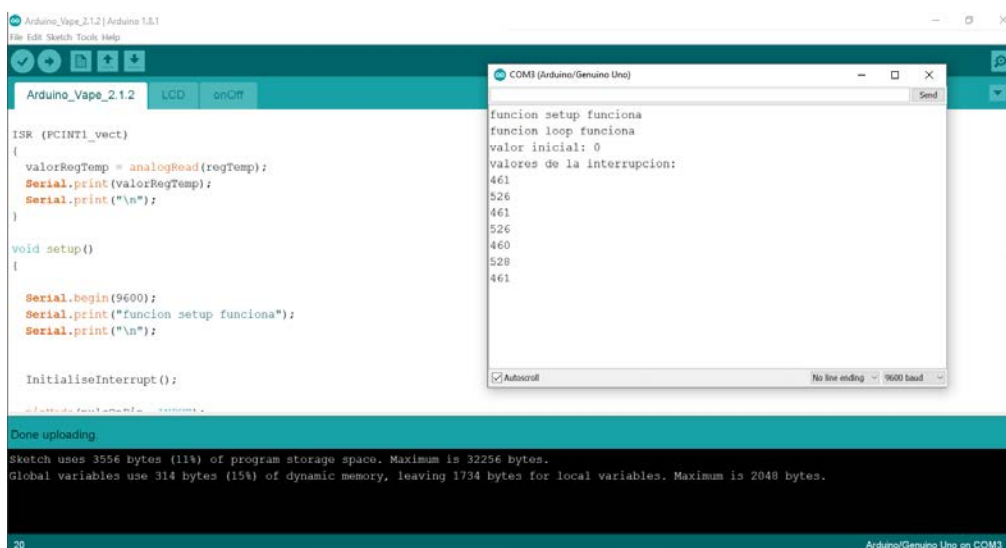


Fig. 3.4 Saltos erráticos entre lecturas analógicas

Los valores de interpolación del potenciómetro son valores de temperatura típicos para vaporizadores herbales y el horno está dimensionado para producir este rango aproximadamente (Sección 5).

En realidad, se preferiría utilizar la interrupción externa 0 para el pulsador ON ya que éste debería tener la máxima prioridad siempre. Sin embargo, por no cruzar cables ni separar a las patillas de los pulsadores TempMás y TempMenos se decidió dejarlo así. No es probable que la jerarquía de interrupciones descrita en la Figura 3.5 tenga mucho efecto en este proyecto ya que un usuario no suele pulsar más de una entrada en un momento dado.

1	Reset		
2	External Interrupt Request 0 (pin D2)	(INT0_vect)	Pulsador TempMás
3	External Interrupt Request 1 (pin D3)	(INT1_vect)	Pulsador TempMenos
4	Pin Change Interrupt Request 0 (pins D8 to D13)	(PCINT0_vect)	Pulsador ON
5	Pin Change Interrupt Request 1 (pins A0 to A5)	(PCINT1_vect)	Regulador de temperatura
6	Pin Change Interrupt Request 2 (pins D0 to D7)	(PCINT2_vect)	
7	Watchdog Time-out Interrupt	(WDT_vect)	
8	Timer/Counter2 Compare Match A	(TIMER2_COMPA_vect)	
9	Timer/Counter2 Compare Match B	(TIMER2_COMPB_vect)	
10	Timer/Counter2 Overflow	(TIMER2_OVF_vect)	MsTimer2
11	Timer/Counter1 Capture Event	(TIMER1_CAPT_vect)	
12	Timer/Counter1 Compare Match A	(TIMER1_COMPA_vect)	ISR

Fig. 3.5 Prioridad de las interrupciones³

Referencias

1. <https://playground.arduino.cc/Main/MsTimer2>
2. <https://arduino.stackexchange.com/questions/22212/using-millis-and-micros-inside-an-interrupt-routine>
3. <https://stackoverflow.com/questions/5111393/do-interrupts-interrupt-other-interrupts-on-arduino>

4 Sensor de Temperatura

Descripción del Hardware

Un sensor MAX6675 detecta la temperatura del horno cada 5 segundos y, junto con el inductor (Sección 5), permite cerrar el bucle de control PID (Sección 6). Su sonda es capaz de detectar la fuente de calor (entre 0°C y 1023.75°C) con su unión caliente mientras mantiene alejada la unión fría (entre -20°C y 85°C)¹, es decir, el chip MAX6675 (Figura 4.0). Este chip está conectado a las patillas analógicas del Arduino, aunque normalmente se conectan a las digitales (Figura 4.1). Los termopares de tipo K, como este sensor, trabajan bien en atmósferas oxidantes², una característica muy conveniente para un sensor de flujos de aire caliente.

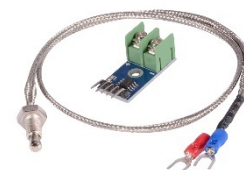


Fig. 4.0 Sonda y chip del sensor de temperatura MAX6675
<http://www.xcsource->

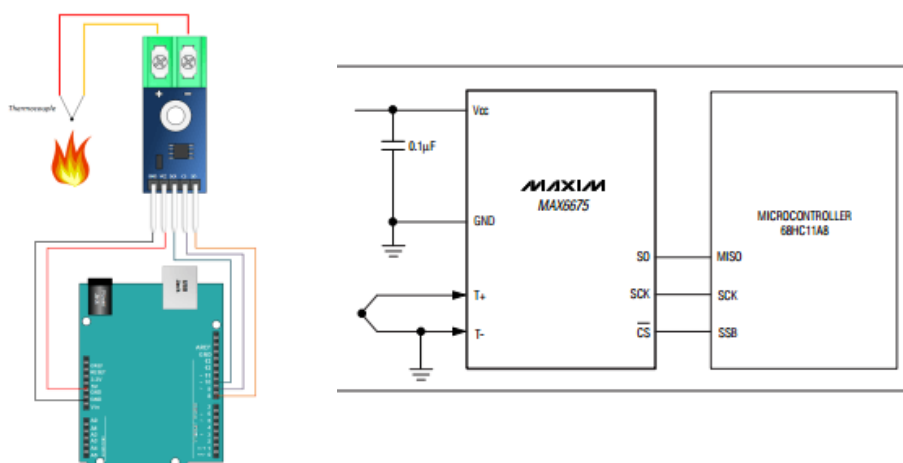


Fig. 4.1 Imagen⁴, cableado⁴, y diagrama¹ del sensor de temperatura
Nota: En este proyecto, se usan patillas analógicas en lugar de digitales

Al tratarse de un termopar de tipo K, tiene una entrada T- de alumel (níquel-aluminio) y T+ de cromel (níquel-cromo). La Figura 4.2 también muestra su patilla de alimentación (V_{cc}), de tierra (GND), una salida Serial Data Output (SO), y las entradas Serial Clock Input (SCK) y Chip Select (CS)¹.

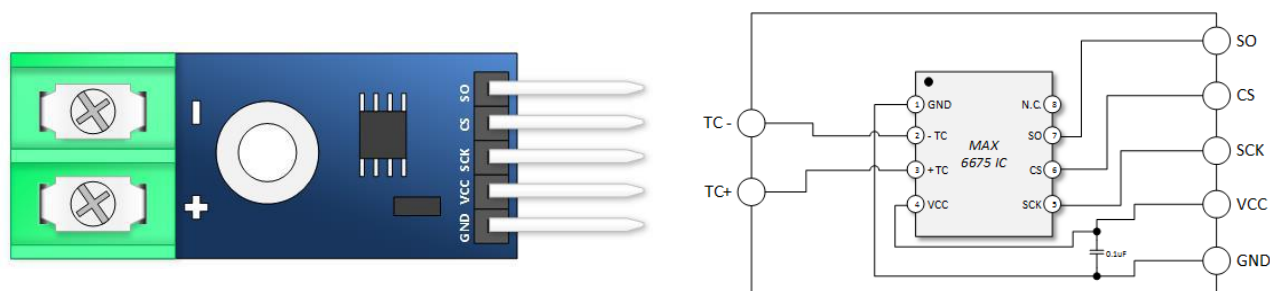


Fig. 4.2 Detalle de las patillas del sensor³

La Figura 4.14 contiene al circuito entero del chip MAX6675, el cual se alimenta mediante la fuente V_{CC} , cuyo ruido se recomienda minimizar mediante un condensador cerámico de $0.1 \mu F$ ¹ (Figura 4.4). Éste se coloca cerca del pin de alimentación (Figura 4.4), como es típico de los condensadores de paso, también conocidos como de derivación (Figura 4.4)⁵.

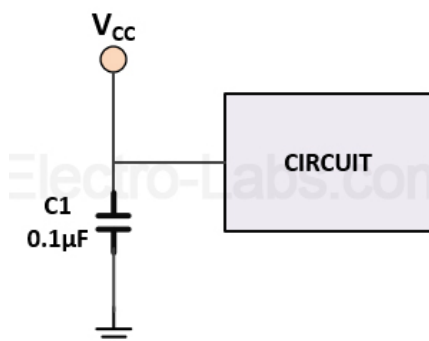


Fig. 4.3 Posición típica de un condensador de paso⁵

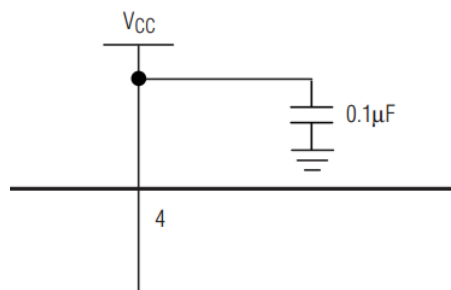


Fig. 4.4 Detalle de la conexión V_{CC} ¹

Este tipo de condensador limpia la señal de entrada, una tensión c.c. con pequeñas oscilaciones c.a., para producir una tensión c.c. pura (Figura 4.5 y 4.6).

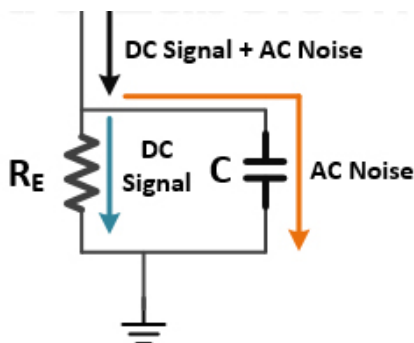


Fig. 4.5 Señal c.c. limpiada por condensador⁵

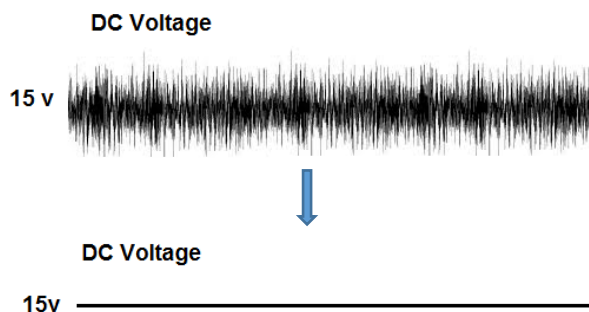


Fig. 4.6 Limpieza de una señal c.c.⁶

Esta tensión alimenta al amplificador A1, descrito en el manual como un amplificador de bajo ruido (LNA)¹, que recibe las señales del termopar T_+ y T_- (Figura 4.7). Estas entradas pasan primero por los conmutadores S1, S2, y S3, que no se detallan en la documentación oficial, pero podrían tratarse de interruptores chopper que trocean la señal continua para producir una señal alterna, que es más fácil de manejar dentro de un amplificador⁷. Aunque el diagrama representa a este amplificador como un amplificador operacional (op-amp) estándar, casi siempre se trata de 3 op-amps (Figura 4.8) que forman un amplificador de instrumentación⁷.

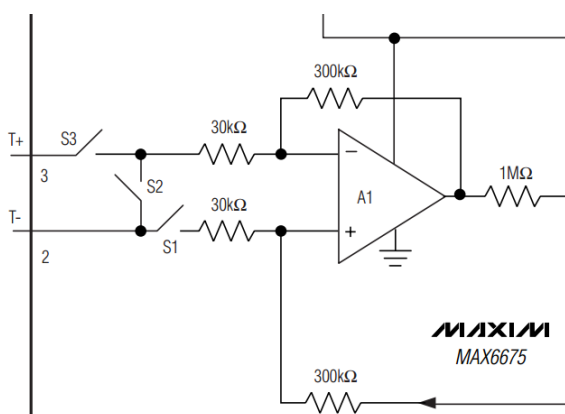


Fig. 4.7 Detalle del amplificador A1¹

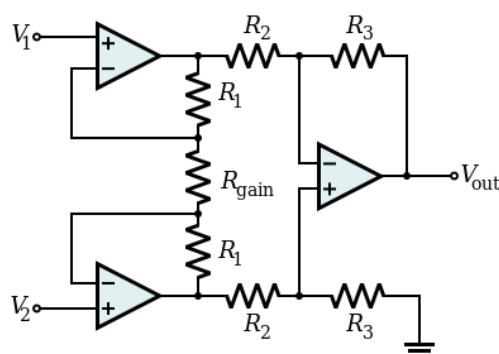


Fig. 4.8 Amplificador de instrumentación⁸

Los dos primeros op-amps funcionan como buffers para cada entrada (+,-) y proporcionan una tensión entre las dos entradas (justo antes de las resistencias $R_2 = 30\text{ k}\Omega$) de: $V_{intermedia} = (V_2 - V_1) \left(1 + \frac{2R_1}{R_{gain}}\right)$ ⁹. Como no se conocen los valores de las resistencias R_1 ni R_{gain} , no se puede calcular la ganancia de esta etapa, pero se sabe que los amplificadores de instrumentación suelen ser amplificadores diferenciales con un alto factor rechazo al modo común y una pequeña ganancia de modo común $\left(\frac{R_2}{R_3} = \frac{30\text{ k}\Omega}{300\text{ k}\Omega} = 0.1\right)$. Es decir, la primera etapa del amplificador A1 es muy capaz de rechazar el ruido ambiental común a las dos entradas sin amplificarlo mucho¹⁰. En la segunda etapa, la tensión intermedia caerá por igual al cruzar las resistencias R_2 , por lo que llegará la misma diferencia a las entradas del tercer op-amp. Éste producirá una tensión de salida $V_o = (V_2 - V_1) \left(1 + \frac{2R_1}{R_{gain}}\right) \left(\frac{R_3}{R_2}\right)$ ⁹ con una ganancia de $\frac{R_3}{R_2} = \frac{300\text{ k}\Omega}{30\text{ k}\Omega} = 10$. Finalmente, para evitar el bias del amplificador, se introduce en su entrada no inversora una tensión de referencia (Figura 4.9). Con esto se corrige el hecho de que, para una tensión diferencial de 0 V en su entrada, algo de tensión estaba apareciendo a su salida¹¹. En resumen, al situar al LNA A1 cerca de la fuente de la señal, su ganancia general amplifica la señal y la hace más robusta frente al ruido que se pueda originar en las siguientes etapas del circuito¹².

Antes de llevar la salida de A1 al convertidor analógico-digital, conviene pasarla por un filtro de paso bajo RC para reducir el ruido fuera del ancho de banda deseado¹³ (Figura 4.10). Según el teorema de muestreo de Nyquist, la máxima frecuencia contenida en la señal de entrada debe ser menor o igual a la mitad de la frecuencia de muestreo¹⁴. Al permitir sólo el paso de bajas frecuencias, este filtro anti-alias se asegura de que las frecuencias por encima de $f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi(1\text{ M}\Omega)(20\text{ pF})} = 7957.75\text{ Hz}$ (ruido) no se van a muestrear¹⁴ (Figura 4.10). Esta señal filtrada se pasa por el op-amp seguidor de tensión A2 (Figura 4.10), que proporciona esta misma tensión en su salida. Gracias a su alta impedancia de entrada y baja impedancia de salida, sirve como buffer que aísla al termopar del chip y permite medir la tensión del termopar utilizando una intensidad muy pequeña que no afecte a la medición del sensor¹⁵.

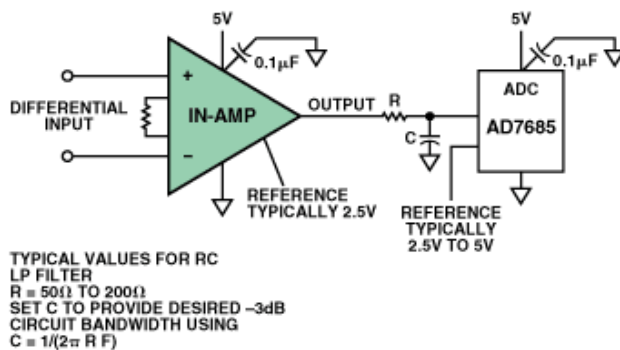


Fig. 4.9 Filtro anti-aliasing¹³

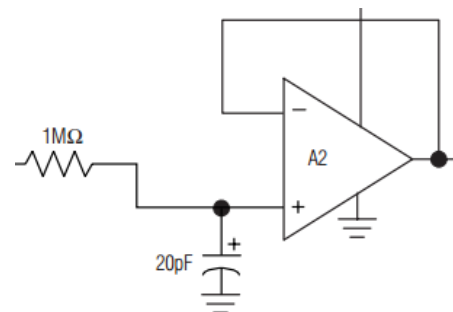


Fig. 4.10 Detalle del amplificador A2¹

Como los termopares producen una diferencia de tensión debido al efecto Seebeck (el in-amp A1 es esencialmente un amplificador diferencial), esta diferencia se debe sumar a una referencia para producir el valor absoluto de tensión¹⁶ (Figura 4.11). Con esta compensación por unión fría, el termopar puede mantener una buena precisión en su unión caliente, aunque fluctúe la temperatura en su unión fría¹ (Figura 4.12). El diodo de compensación convierte la lectura de la temperatura ambiental en un voltaje, el cual se suma a la salida de A2¹ antes de entrar en el convertidor analógico-digital (ADC en la Figura 4.13). Al tratarse de un termopar de tipo K, la tensión varía $41 \mu\text{V}/^\circ\text{C}$ ¹⁷, por lo que se aproxima su ecuación característica a la ecuación lineal $V_{OUT} = \left(41 \frac{\mu\text{V}}{^\circ\text{C}}\right) \cdot (T_R - T_{AMB})$ ¹. Finalmente, en el ADC se suma sus dos entradas y manda el resultado de 12 bits a la patilla SO¹, donde 000000000000 = 0°C y 111111111111 = 1023.75°C .

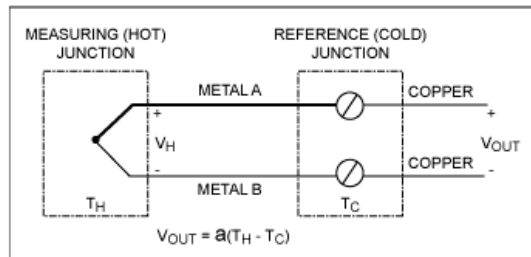


Fig. 4.12 Unión caliente y fría

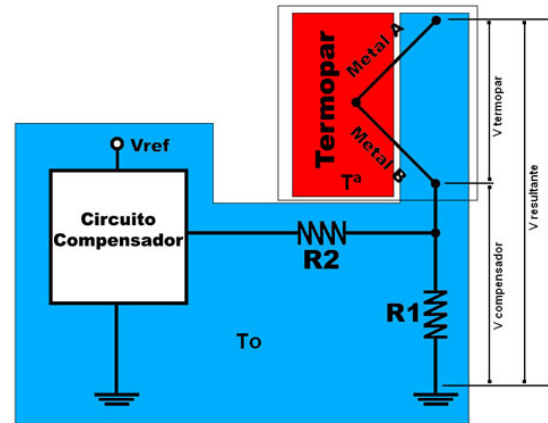


Fig. 4.11 Compensación por unión fría¹⁶

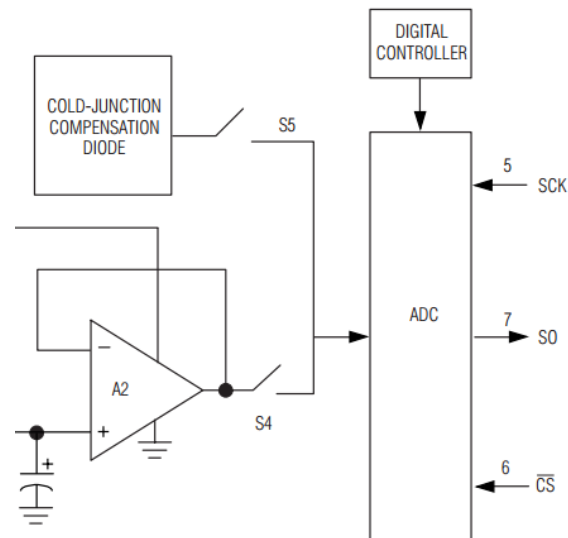


Fig. 4.13 Tensión relativa + referencia

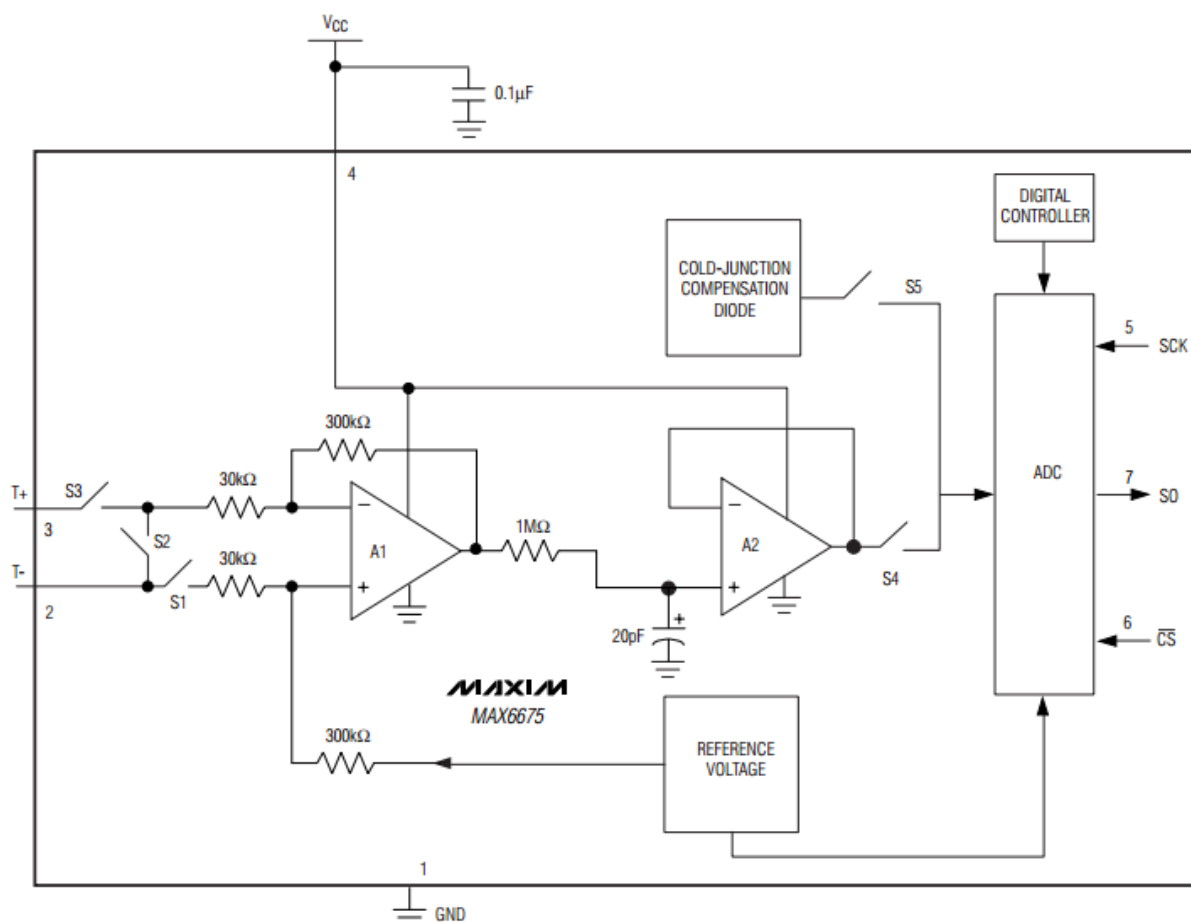


Fig. 4.14 Circuito interno del MAX6675¹

El chip envía la señal de SO al Arduino a través de la interfaz serie. La comunicación SPI se habilita al deshabilitar la patilla CS y, al aplicar una señal de reloj en la patilla SCK, se lee el resultado en SO. La palabra SO contiene 16 bits (Figura 4.15).

DUMMY SIGN BIT	12-BIT TEMPERATURE READING												THERMOCOUPLE INPUT	DEVICE ID	STATE
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	MSB											LSB		0	Three- state

Fig. 4.15 Bits del registro SO¹

Por lo tanto, se requiere 16 ciclos del reloj para una lectura completa¹. Con el flanco negativo de CS, la conversión en el ADC se detiene inmediatamente y se empieza a escribir los bits D15 – D0 (Figura 4.16). El primer bit no tiene significado y se pone siempre a 0. Los bits D14 a D3 contienen los 12 bits que codifican el valor de la temperatura. El bit D2 normalmente está a 0 y sólo se pone a 1 si se está utilizando el termopar en configuración abierta (con T- a tierra), el cual no es el caso aquí. El bit D1 proporciona un identificador del dispositivo y el D0 es un tri-estado¹, útil a la hora de conectar varios dispositivos mediante la interfaz serie¹⁸.

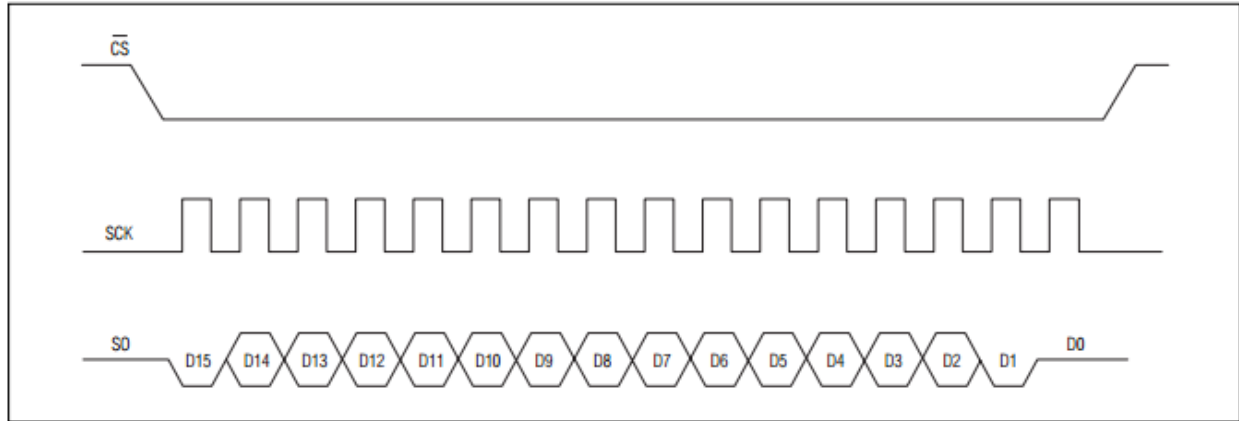


Fig. 4.16 Escritura del registro SO¹

Descripción del Software

Para este sensor hace falta la librería `max6675.h`¹⁹ que nos permite crear el objeto `termopar` y asociarle 3 patillas digitales. Por razones de disponibilidad, se ha optado por usar las patillas analógicas A1, A2, y A3 como si fueran las digitales 15, 16, y 17, respectivamente^{20, 21} (Figura 4.15).

```
int CLK = 15; // A1 = D15
int CS = 16; // A2 = D16
int SO = 17; // A3 = D17

MAX6675 termopar(CLK, CS, SO);
```

Fig. 4.15 Inicialización del termopar

La Figura 4.16 muestra la función `temp()`, que comprueba que no se ha excedido la temperatura de seguridad de 280 °C, en cuyo caso establece un intervalo de muestreo de 3 segundos con `millis()`, para utilizar la función `readCelsius()` de la librería del chip⁴. En caso contrario, se llama a la función `off()` para apagar el sistema.

```
17 double temp()
18 {
19     if (termopar.readCelsius() <= 290) // asegurarse de que no se ha excedido la temperatura máxima de seguridad
20     {
21         if (millis() - tempTref >= 3000)
22         {
23             return termopar.readCelsius();
24             tempTref = tempTref + intervaloActualizarTemp;
25         }
26     }
27     else // apagar si detecta temperatura excesiva
28     {
29         off();
30     }
31 }
```

Fig. 4.16 Función de lectura de temperatura

Un análisis aproximado del reloj serie de 4,3 MHz (Figura 4.18) indica que, para una lectura completa de temperatura (16 ciclos de SO), se puede escribir 268.750 escrituras completas de SO:

$$\frac{4,3 \cdot 10^6 \frac{\text{ciclos de reloj}}{\text{seg}}}{16 \frac{\text{ciclos de reloj}}{\text{escritura completa de SO}}} = 268.750 \frac{\text{escrituras completas de SO}}{\text{seg}}$$

O bien, cada escritura tarda aproximadamente 3,72 microsegundos:

$$\left(268.750 \frac{\text{escrituras completas de SO}}{\text{seg}}\right)^{-1} = 3,72 \cdot 10^{-6} \frac{\text{seg}}{\text{escrituras completas de SO}}$$

Para confirmar, de la Figura 4.18 se calcula que cada ciclo de SCK tarda $t_{CH} + t_{CL} = 100 \text{ ns} + 100 \text{ ns} = 2 \cdot 10^{-7}$ segundos por ciclo de SCK, por lo que los 16 ciclos del SO duran $3,2 \times 10^{-6}$ segundos, muy similar al valor calculado anteriormente. La diferencia entre estos dos valores puede deberse a que se ha elegido los valores mínimos de la Tabla 4 para la subida y bajada de SCK, y se ha ignorado el tiempo entre la caída de CS y la subida de SCK (t_{CSS}), la caída de CS y la habilitación de SO (t_{DV}), la subida de CS y la deshabilitación de SO (t_{TR}), y la caída de SCK y la validación de SO (t_{DO}). De todas formas, la rapidez de este sensor es más que suficiente para el intervalo de muestreo de 3 segundos.

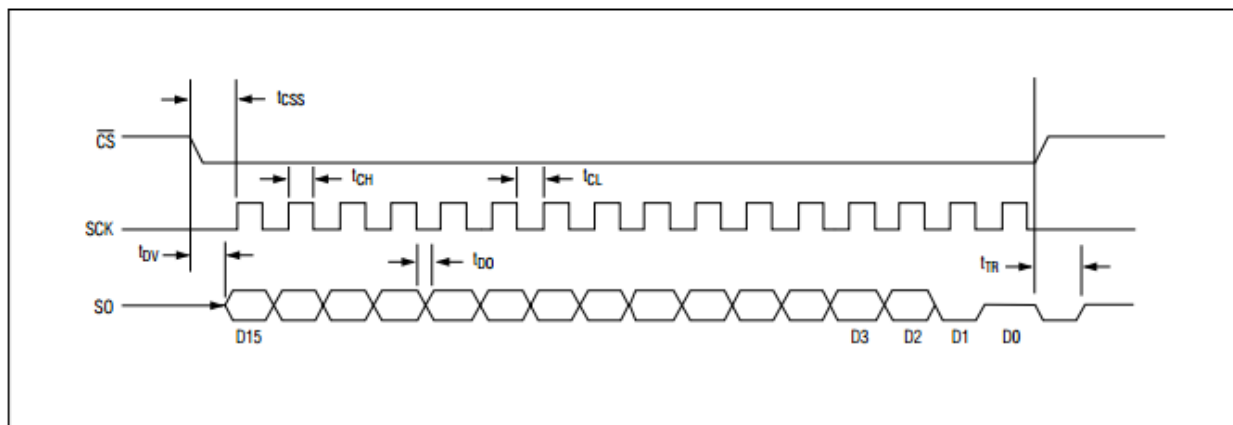


Fig. 4.17 Temporización de escritura del registro SO¹

TIMING				
Serial Clock Frequency	f_{SCL}		4.3	MHz
SCK Pulse High Width	t_{CH}		100	ns
SCK Pulse Low Width	t_{CL}		100	ns
CSB Fall to SCK Rise	t_{CSS}	$C_L = 10\text{pF}$	100	ns
CSB Fall to Output Enable	t_{DV}	$C_L = 10\text{pF}$	100	ns
CSB Rise to Output Disable	t_{TR}	$C_L = 10\text{pF}$	100	ns
SCK Fall to Output Data Valid	t_{DO}	$C_L = 10\text{pF}$	100	ns

Fig. 4.18 Tiempos de las patillas CS, SCK, y SO¹

Proceso

Para los colores de los terminales, una fuente inicial parece asumir que sólo existe el código ANSI y que los fabricantes a veces se equivocan³. En realidad, parece que hay distintos códigos estandarizados (Figura 4.19) y, en particular, el termopar de este proyecto sigue el estándar BS.

Referencias

1. <http://henrysbench.capnfatz.com/wp-content/uploads/2015/05/MAX6675-Datasheet.pdf>
2. https://en.wikipedia.org/wiki/Thermocouple#Type_K
3. <http://www.xcsource-pic.com/TE578-E-10-main1.jpg>
4. <http://henrysbench.capnfatz.com/henrys-bench/arduino-temperature-measurements/max6675-temp-module-arduino-manual-and-tutorial/>
5. <http://www.electronica2000.com/temas/capacitores-de-paso.htm>
6. <http://www.learningaboutelectronics.com/Articles/What-is-a-bypass-capacitor.html>
7. [https://en.wikipedia.org/wiki/Chopper_\(electronics\)#Chopper_amplifiers](https://en.wikipedia.org/wiki/Chopper_(electronics)#Chopper_amplifiers)
8. https://en.wikipedia.org/wiki/Instrumentation_amplifier
9. https://es.wikipedia.org/wiki/Amplificador_de_instrumentaci%C3%B3n
10. https://en.wikipedia.org/wiki/Common-mode_rejection_ratio
11. https://ocw.mit.edu/courses/media-arts-and-sciences/mas-836-sensor-technologies-for-interactive-environments-spring-2011/readings/MITMAS_836S11_read02_bias.pdf
12. https://en.wikipedia.org/wiki/Low-noise_amplifier#Noise_reduction
13. <http://www.analog.com/en/analog-dialogue/articles/common-problems-when-designing-amplifier-circuits.html>
14. <http://digital.ni.com/public.nsf/allkb/68F14E8E26B3D101862569350069E0B9>
15. https://es.wikipedia.org/wiki/Amplificador_operacional#Seguidor_de_voltaje_o_tensi.C3.B3n
16. https://es.wikipedia.org/wiki/Compensaci%C3%B3n_de_uni%C3%B3n_fr%C3%ADa
17. <https://es.wikipedia.org/wiki/Termopar#Tipos>
18. <https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/CompOrg/tristate.html>
19. <https://github.com/adafruit/MAX6675-library>
20. <https://www.arduino.cc/en/Tutorial/AnalogInputPins>
21. <https://forum.arduino.cc/index.php?topic=28720.0>

5 Horno

Descripción

El horno consiste de un hilo de nicromo de aproximadamente 20 cm enrollado sobre un tubo de cristal (la carcasa del hornillo) que calienta el aire en su interior. El hilo se alimenta con una tensión continua de 13,0 V y puede consumir hasta 6,3 A. En todo momento, la corriente que atraviesa a la bobina se controla mediante un transistor Darlington, que recibe en su base la señal PWM de un pin digital del Arduino (Figura 5.0).

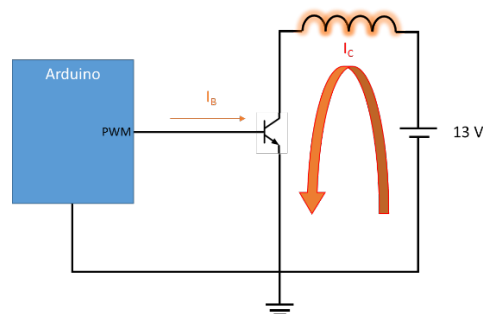


Fig. 5.0 Control PWM del horno

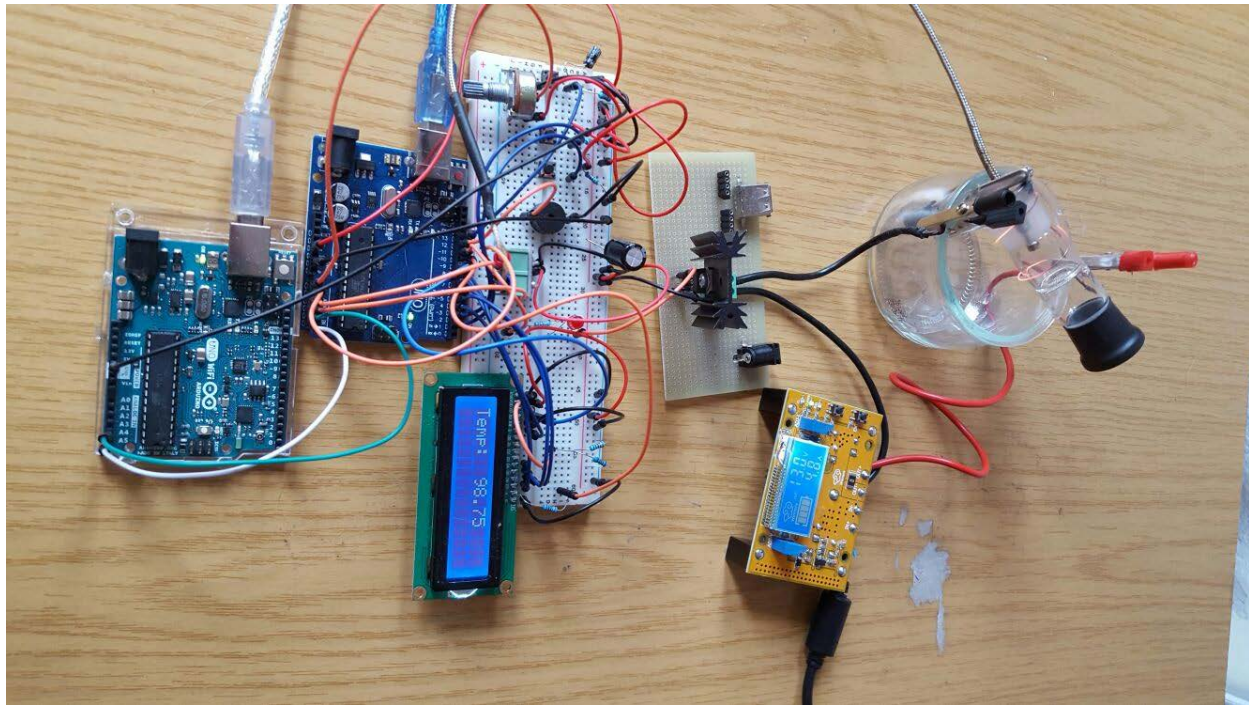


Fig. 5.1 Maqueta con horno encendido

El hilo de nicromo, una aleación de proporciones variables de níquel-cromo o níquel-cromo-hierro, actúa como una resistencia altamente capaz de disipar la potencia que le atraviesa. Con una resistividad de aproximadamente $\rho = 1,3 \cdot 10^{-6} \Omega m$ ¹, una longitud de $L = 0,2 m$, y un diámetro de $4,06 \cdot 10^{-4} m$, este hilo supone una resistencia de:

$$R = \rho \frac{L}{A} = (1,3 \cdot 10^{-6} \Omega m) \frac{0,2 m}{\pi \left(\frac{4,06 \cdot 10^{-4} m}{2} \right)^2} \cong 2 \Omega$$

Se pueden encontrar valores similares en otras fuentes, pero al no conocerse las proporciones exactas de esta aleación, el valor real puede diferir un poco del calculado². Al conducir un máximo de 6,3 A, este hilo disipa una potencia de aproximadamente $P = RI^2 = (2 \Omega) \cdot (6,3 A)^2 = 79,38 W$, protegiendo al transistor conectado aguas abajo (Figura 5.0).

Este transistor se trata de un Darlington, es decir, de dos transistores bipolares NPN integrados (Figura 5.2) en un encapsulado TO-220. Este modelo BDX33C puede soportar una tensión V_{CB} y V_{CE} de al menos 100 V, una corriente en su colector de hasta 10 A, y una corriente en su base de hasta 250 mA³. No se supera ninguno de estos límites en esta aplicación, aunque sí se requiere una resistencia de base $R_B = 330 \Omega$ y un disipador de calor considerable, no pudiendo superarse una temperatura en su unión de 150 °C y una disipación en su colector de 70 W³. El transistor se opera en su región activa, amplificando la corriente que le llega desde el Arduino a su base de forma que circula una corriente mucho mayor ($\beta = 250$) en su colector, esto es, a través de la bobina.

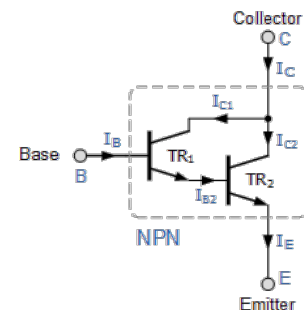


Fig. 5.2 Transistor Darlington⁴

```

6 void calentarHorno(int valorPWM)
7 {
8   analogWrite(HORNO, valorPWM);
9 }

```

Fig. 5.3 Función calentarHorno()

La función `calentarHorno()` dentro del `loop()` se encarga de escribir un valor PWM en la patilla digital del transistor. Este valor es la salida de la función PID (Sección 6), que se pasa como un parámetro a `calentarHorno()` (Figura 5.3). De esta forma, se puede controlar desde el Arduino a la corriente que pasa por la bobina, es decir, a la temperatura del hornillo.

Proceso

Para la bobina del horno existen distintas opciones, como módulos prefabricados⁵ o escaleras de resistencias⁵, aunque su capacidad de soportar la suficiente corriente para llegar a 200 – 300 °C es dudosa, recomendándose en su lugar el hilo de nicromo^{6,7}.

Una calculadora online⁸ permite familiarizarse con la relación entre la temperatura máxima deseada y la longitud, el diámetro, y la tensión y corriente de alimentación (Figura 5.3). Principalmente, el objetivo ha sido alcanzar una temperatura de al menos 220 °C y la suficiente longitud para dar 2 o 3 vueltas al horno de cristal. De todas formas, los valores de esta calculadora no se aplicaron directamente.

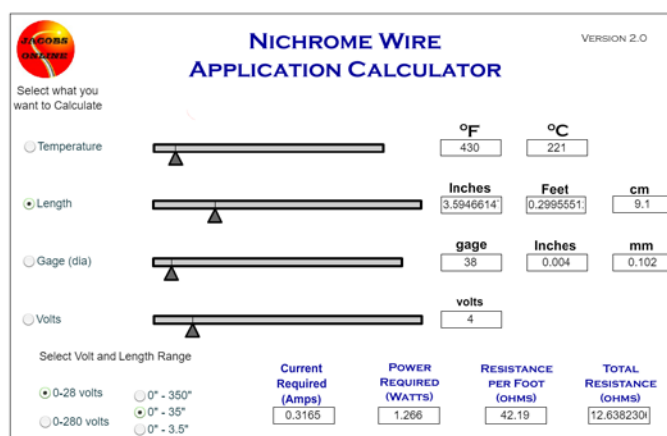


Fig. 5.3 Calculadora de aplicación⁸

Por simplicidad y elegancia, se preferiría utilizar una sola fuente para el circuito de la bobina y el Arduino mediante su entrada V_{in} , pero se recomienda separar los circuitos para evitar el ruido.

Para la fuente externa, los módulos de alimentación como el MB102 están limitados a una salida de 700 mA⁹ y los cargadores USB suelen estar limitados a 2.0 – 2.5 A. Por ello se acabó conectando un cargador de portátil antiguo a un convertidor “buck” cuya pantalla muestra las tensiones y corrientes de entrada y de salida (Figura 5.4). Al bajar la tensión de los 19 V del cargador a 13 V, el convertidor puede producir una corriente de salida mayor a los 3,31 A del cargador. Sus salidas están conectadas al transistor mediante una regleta de dos contactos, que puede soportar hasta 8 A¹⁰, mayor que los 6,3 A máximos que circulan por el colector y emisor.

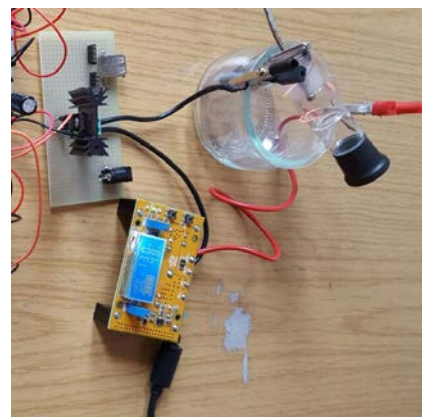


Fig. 5.4 Convertidor, transistor, y horno



Las corrientes del transistor y su resistencia de base se han determinado principalmente por prueba y error, basándose en parte en ciertas pautas básicas¹¹. Se conectó al horno directamente al convertidor buck y se iba aumentando la tensión y corriente de salida del convertidor hasta conseguir una temperatura máxima deseada de aproximadamente 300 °C. Dividiendo esta corriente del colector máxima entre $\beta = 250$ se halla la corriente de base máxima y consecuentemente la resistencia de base necesaria. A continuación, introduciendo el transistor y asignándole un valor PWM de 255 se intentó llegar a esta máxima temperatura. En algunos intentos iniciales el transistor explotó ya que le faltaba un disipador de calor. El cálculo de la resistencia de base necesaria produjo un valor de aproximadamente 195 Ω . Sin embargo, al no dar buenos resultados con respecto a la corriente del colector, se optó por ir probando resistencias desde 5.1 K Ω hasta 220 Ω hasta dar con un valor aceptable. Con 330 Ω todavía funciona el circuito, pero al bajar a 220 Ω explotó el transistor. Con un transistor nuevo y volviendo a probar una resistencia de 330 Ω , el circuito volvió a funcionar por lo que se ha utilizado este valor en el circuito final. Ajustando la salida del convertidor un poco más permitió conseguir la misma temperatura máxima de alrededor de 300 °C.

Para el transistor, en lugar de un BJT se podría haber utilizado un MOSFET, cuya menor disipación de energía y su control por tensión habría sido útil¹². Pero los transistores BJT son aptos para mandarles señales PWM y con comparar distintas hojas de especificaciones^{3,13} se puede determinar que el BD333C puede soportar las tensiones y corrientes previstas.

La mayoría de las fuentes recomiendan un diodo flyback¹⁴ en paralelo con la bobina para disipar la corriente que permanece en el circuito de la bobina tras un corte de alimentación. Sin embargo, por limitaciones constructivas y de prioridades y, al ver que el circuito de momento no ha sufrido por faltarle el diodo, se ha ignorado ese componente. No obstante, se trata definitivamente de una adición futura al prototipo.

Referencias

1. <http://hypertextbook.com/facts/2007/HarveyKwan.shtml>
2. <https://en.wikipedia.org/wiki/Nichrome>
3. http://www.braude.ac.il/files/departments/electrical_electronic_engineering/labs/data_pages/p5.pdf
4. <http://www.electronics-tutorials.ws/transistor/darlington-transistor.html>
5. <http://alexnlid.com/product/5v-12v-zvs-induction-heating-power-supply-module-with-coil/>
6. <http://www.alanzucconi.com/2016/08/02/arduino-heater-1/>
7. https://www.reddit.com/r/arduino/comments/3s08uf/most_efficient_way_to_create_a_heating_element/
8. <http://www.jacobs-online.biz/nichrome/NichromeCalc.html>
9. <https://soloarduino.blogspot.com.es/2016/04/mb102-alimentacion-para-breadboards.html>
10. <http://www.electronicaembajadores.com/Productos/Detalle/11/ctna2502/regleta-clema-pcb-paso-2-54mm-2-contactos>
11. <https://art511.wordpress.com/works-in-progress/how-to-use-a-transistor-as-a-switch/>
12. <https://www.luisllamas.es/arduino-transistor-mosfet/>
13. <https://art511.files.wordpress.com/2010/03/tip120.pdf>
14. <https://www.luisllamas.es/salidas-mayor-potencia-arduino-transistor-bjt/>

6 Control PID de Temperatura

Descripción

El sistema del horno, sensor, y Arduino conforman un bucle cerrado con control PID (Figura 6.0). El setpoint (SP) se fija con los pulsadores de temperatura y se compara con la salida del sensor de temperatura. Este error entra en el controlador PID $\left(G_c(s) = 3.2 + \frac{0.095}{s} + 4.5s\right)$, que es una función del Arduino. Esta función entrega un valor PWM al horno, el cual está integrado con el sensor de temperatura para formar la planta $\left(G_p(s) = \frac{0.0001403}{s^2 + 0.02139s + 0.0001157}\right)$ y genera la variable del proceso (PV) detectada por el sensor.

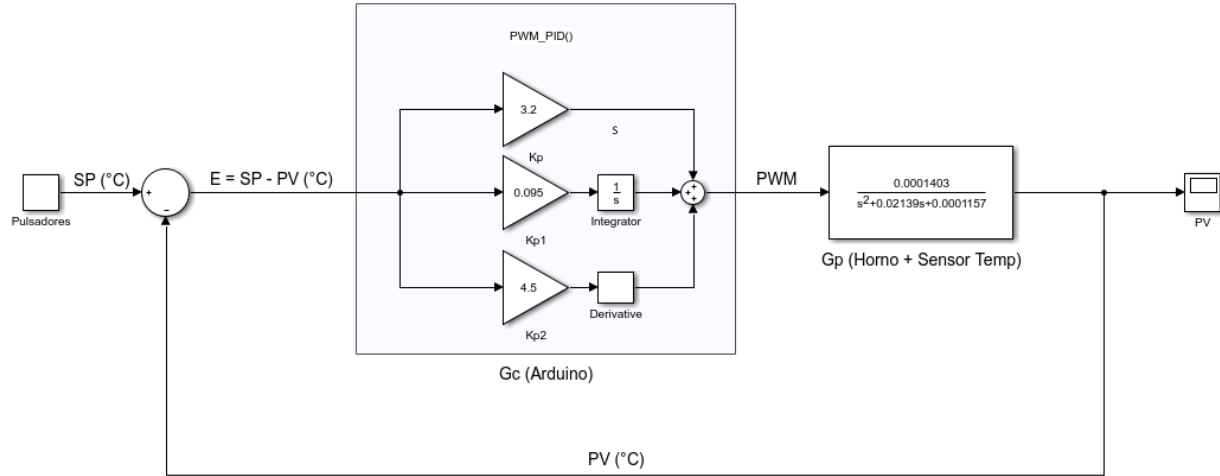


Fig. 6.0 Diagrama de bloques

Esta $G_p(s) = \frac{0.0001403}{s^2 + 0.02139s + 0.0001157}$ tiene una ganancia estática $K_e = 1,2126188$, una frecuencia natural $\omega_n = 0,01075766$, y un coeficiente de amortiguamiento de $\xi = 0,9942184$. Se trata por lo tanto de una función subamortiguada, aunque casi críticamente amortiguada. Sus polos conjugados $p = -0,0106955 \pm j0,00115123$ prácticamente carecen de una parte imaginaria, lo cual es consistente con la naturaleza casi críticamente amortiguada de esta planta.

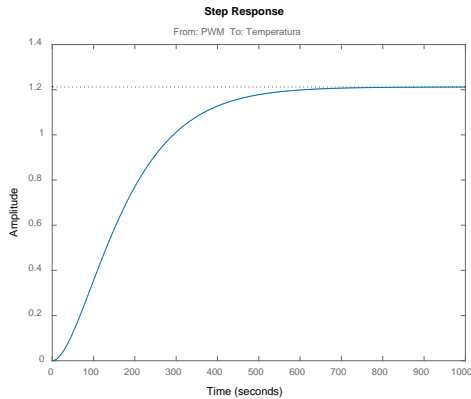


Fig. 6.1 Respuesta al escalón del lazo abierto

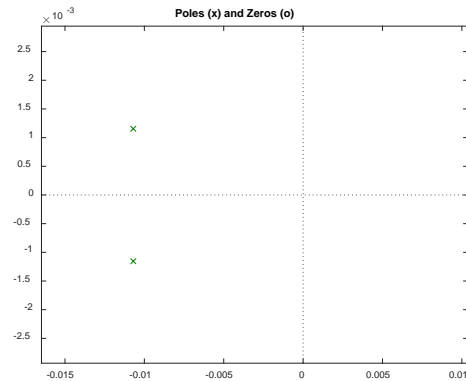


Fig. 6.2 Polos del sistema

En el siguiente apartado (*Proceso*) se detallan los ensayos empíricos que resultaron en una $G_c(s) = 3.2 + \frac{0.095}{s} + 4.5s$, que se introduce en un bucle cerrado para obtener la función $G(s) = \frac{0.0006313s^2 + 0.000449s + 0.00001333}{s^3 + 0.02202s^2 + 0.0005647s + 0.00001333}$.

```
Gc =
    4.5 s^2 + 3.2 s + 0.095
    -----
    s
Continuous-time transfer function.
```

Fig. 6.3 FDT del

```
>> Gp = tf([0.0001403],[1 0.02139 0.0001157])
Gp =
    0.0001403
    -----
    s^2 + 0.02139 s + 0.0001157
Continuous-time transfer function.
```

Fig. 6.4 FDT de la planta

```
>> G = feedback(Gc*Gp,1)
G =
    0.0006313 s^2 + 0.000449 s + 1.333e-05
    -----
    s^3 + 0.02202 s^2 + 0.0005647 s + 1.333e-05
Continuous-time transfer function.
```

Fig. 6.5 FDT del bucle cerrado

El bucle cerrado muestra una reducción del tiempo de establecimiento de casi 67%, siendo éste la característica dinámica primaria a la hora de diseñar el controlador. La sobreoscilación generada no es importante, siendo menor que un 2%.

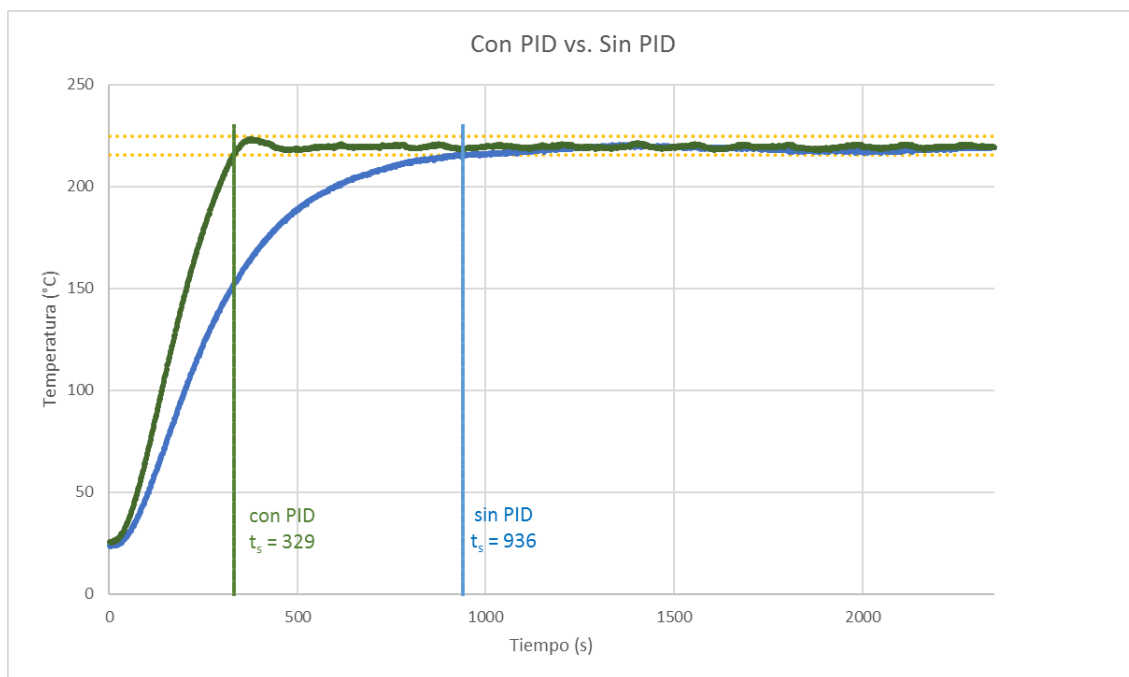


Fig. 6.6 Lazo abierto sin control PID (azul) y lazo cerrado con control PID (verde)

La función `PWM_PID()` corre en el `loop()` cada 5 segundos, ya que no es necesario vigilar la temperatura actual constantemente para este sistema relativamente lento. Esta función calcula y devuelve el `Output` (Figura 6.7), un valor PWM que se pasa como parámetro en la función `calentarHorno()` de la sección anterior.

```
7 double PWM_PID()
8 {
9     Input = temp();           // vigilar temperatura actual
10    Setpoint = SPactual;      // seguir el setpoint actual
11
12    PIDhorno.Compute();       // calcular salida PWM
13
14    return Output;            // devolver salida PWM
15 }
```

Fig. 6.7 Función `PWM_PID()`

Proceso

La función de transferencia (FDT) se halló fijando un valor de PWM = 0 en el horno e imprimiendo en el monitor serie el valor de `millis()` seguido por la temperatura actual en cada momento (Figura 6.8).

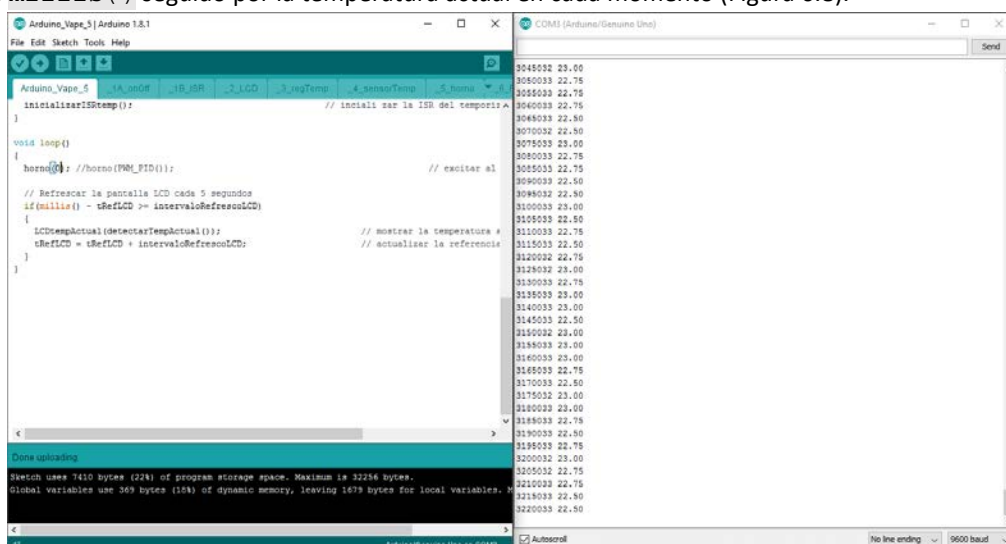


Fig. 6.8 Tiempos (ms) y temperaturas (°C) iniciales

Tras observar una temperatura del horno estable de 22 – 23 °C (la temperatura ambiental), se fijó un valor PWM = 255 y se esperó a que se establezca la temperatura, ahora llegando a 308 – 309 °C (Figura 6.9).

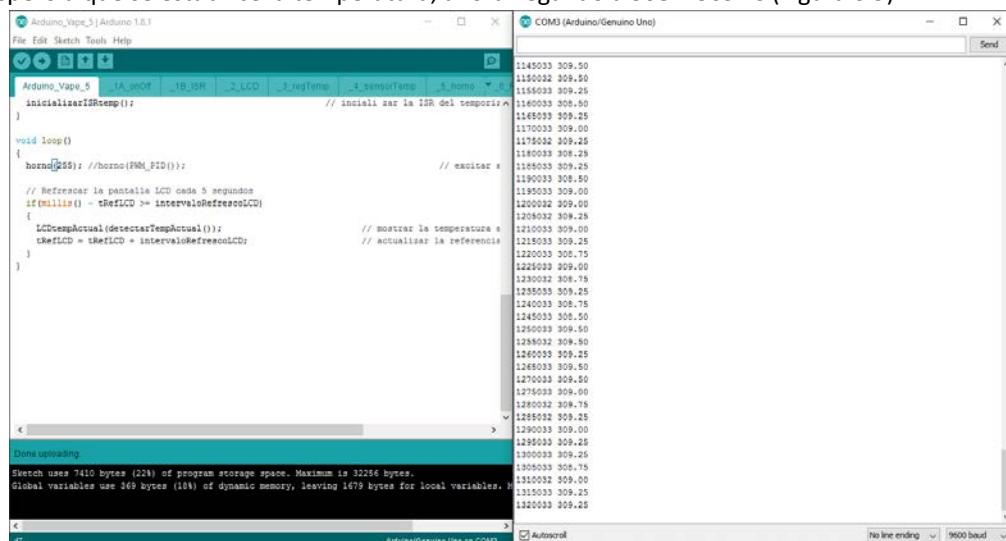


Fig. 6.9 Tiempos (ms) y temperaturas (°C) finales

Se importaron estos datos en Matlab (Figura 6.10) para observar su gráfica (Figura 6.11).

```
>> %Datos = import('datos.xlsx')
%Datos =
1.0000e+02 2.2000e+01
2.0000e+02 2.3000e+01
3.0000e+02 2.4000e+01
4.0000e+02 2.5000e+01
5.0000e+02 2.6000e+01
6.0000e+02 2.7000e+01
7.0000e+02 2.8000e+01
8.0000e+02 2.9000e+01
9.0000e+02 3.0000e+01
1.0000e+03 3.1000e+01
1.1000e+03 3.2000e+01
1.2000e+03 3.3000e+01
1.3000e+03 3.4000e+01
1.4000e+03 3.5000e+01
1.5000e+03 3.6000e+01
1.6000e+03 3.7000e+01
1.7000e+03 3.8000e+01
1.8000e+03 3.9000e+01
1.9000e+03 4.0000e+01
2.0000e+03 4.1000e+01
2.1000e+03 4.2000e+01
2.2000e+03 4.3000e+01
2.3000e+03 4.4000e+01
2.4000e+03 4.5000e+01
2.5000e+03 4.6000e+01
2.6000e+03 4.7000e+01
2.7000e+03 4.8000e+01
2.8000e+03 4.9000e+01
2.9000e+03 5.0000e+01
3.0000e+03 5.1000e+01
3.1000e+03 5.2000e+01
3.2000e+03 5.3000e+01
3.3000e+03 5.4000e+01
3.4000e+03 5.5000e+01
3.5000e+03 5.6000e+01
3.6000e+03 5.7000e+01
3.7000e+03 5.8000e+01
3.8000e+03 5.9000e+01
3.9000e+03 6.0000e+01
4.0000e+03 6.1000e+01
4.1000e+03 6.2000e+01
4.2000e+03 6.3000e+01
4.3000e+03 6.4000e+01
4.4000e+03 6.5000e+01
4.5000e+03 6.6000e+01
4.6000e+03 6.7000e+01
4.7000e+03 6.8000e+01
4.8000e+03 6.9000e+01
4.9000e+03 7.0000e+01
5.0000e+03 7.1000e+01
5.1000e+03 7.2000e+01
5.2000e+03 7.3000e+01
5.3000e+03 7.4000e+01
5.4000e+03 7.5000e+01
5.5000e+03 7.6000e+01
5.6000e+03 7.7000e+01
5.7000e+03 7.8000e+01
5.8000e+03 7.9000e+01
5.9000e+03 8.0000e+01
6.0000e+03 8.1000e+01
6.1000e+03 8.2000e+01
6.2000e+03 8.3000e+01
6.3000e+03 8.4000e+01
6.4000e+03 8.5000e+01
6.5000e+03 8.6000e+01
6.6000e+03 8.7000e+01
6.7000e+03 8.8000e+01
6.8000e+03 8.9000e+01
6.9000e+03 9.0000e+01
7.0000e+03 9.1000e+01
7.1000e+03 9.2000e+01
7.2000e+03 9.3000e+01
7.3000e+03 9.4000e+01
7.4000e+03 9.5000e+01
7.5000e+03 9.6000e+01
7.6000e+03 9.7000e+01
7.7000e+03 9.8000e+01
7.8000e+03 9.9000e+01
7.9000e+03 1.0000e+02
8.0000e+03 1.0100e+02
8.1000e+03 1.0200e+02
8.2000e+03 1.0300e+02
8.3000e+03 1.0400e+02
8.4000e+03 1.0500e+02
8.5000e+03 1.0600e+02
8.6000e+03 1.0700e+02
8.7000e+03 1.0800e+02
8.8000e+03 1.0900e+02
8.9000e+03 1.1000e+02
9.0000e+03 1.1100e+02
9.1000e+03 1.1200e+02
9.2000e+03 1.1300e+02
9.3000e+03 1.1400e+02
9.4000e+03 1.1500e+02
9.5000e+03 1.1600e+02
9.6000e+03 1.1700e+02
9.7000e+03 1.1800e+02
9.8000e+03 1.1900e+02
9.9000e+03 1.2000e+02
1.0000e+04 1.2100e+02
1.0100e+04 1.2200e+02
1.0200e+04 1.2300e+02
1.0300e+04 1.2400e+02
1.0400e+04 1.2500e+02
1.0500e+04 1.2600e+02
1.0600e+04 1.2700e+02
1.0700e+04 1.2800e+02
1.0800e+04 1.2900e+02
1.0900e+04 1.3000e+02
1.1000e+04 1.3100e+02
1.1100e+04 1.3200e+02
1.1200e+04 1.3300e+02
1.1300e+04 1.3400e+02
1.1400e+04 1.3500e+02
1.1500e+04 1.3600e+02
1.1600e+04 1.3700e+02
1.1700e+04 1.3800e+02
1.1800e+04 1.3900e+02
1.1900e+04 1.4000e+02
1.2000e+04 1.4100e+02
1.2100e+04 1.4200e+02
1.2200e+04 1.4300e+02
1.2300e+04 1.4400e+02
1.2400e+04 1.4500e+02
1.2500e+04 1.4600e+02
1.2600e+04 1.4700e+02
1.2700e+04 1.4800e+02
1.2800e+04 1.4900e+02
1.2900e+04 1.5000e+02
1.3000e+04 1.5100e+02
1.3100e+04 1.5200e+02
1.3200e+04 1.5300e+02
1.3300e+04 1.5400e+02
1.3400e+04 1.5500e+02
1.3500e+04 1.5600e+02
1.3600e+04 1.5700e+02
1.3700e+04 1.5800e+02
1.3800e+04 1.5900e+02
1.3900e+04 1.6000e+02
1.4000e+04 1.6100e+02
1.4100e+04 1.6200e+02
1.4200e+04 1.6300e+02
1.4300e+04 1.6400e+02
1.4400e+04 1.6500e+02
1.4500e+04 1.6600e+02
1.4600e+04 1.6700e+02
1.4700e+04 1.6800e+02
1.4800e+04 1.6900e+02
1.4900e+04 1.7000e+02
1.5000e+04 1.7100e+02
1.5100e+04 1.7200e+02
1.5200e+04 1.7300e+02
1.5300e+04 1.7400e+02
1.5400e+04 1.7500e+02
1.5500e+04 1.7600e+02
1.5600e+04 1.7700e+02
1.5700e+04 1.7800e+02
1.5800e+04 1.7900e+02
1.5900e+04 1.8000e+02
1.6000e+04 1.8100e+02
1.6100e+04 1.8200e+02
1.6200e+04 1.8300e+02
1.6300e+04 1.8400e+02
1.6400e+04 1.8500e+02
1.6500e+04 1.8600e+02
1.6600e+04 1.8700e+02
1.6700e+04 1.8800e+02
1.6800e+04 1.8900e+02
1.6900e+04 1.9000e+02
1.7000e+04 1.9100e+02
1.7100e+04 1.9200e+02
1.7200e+04 1.9300e+02
1.7300e+04 1.9400e+02
1.7400e+04 1.9500e+02
1.7500e+04 1.9600e+02
1.7600e+04 1.9700e+02
1.7700e+04 1.9800e+02
1.7800e+04 1.9900e+02
1.7900e+04 2.0000e+02
1.8000e+04 2.0100e+02
1.8100e+04 2.0200e+02
1.8200e+04 2.0300e+02
1.8300e+04 2.0400e+02
1.8400e+04 2.0500e+02
1.8500e+04 2.0600e+02
1.8600e+04 2.0700e+02
1.8700e+04 2.0800e+02
1.8800e+04 2.0900e+02
1.8900e+04 2.1000e+02
1.9000e+04 2.1100e+02
1.9100e+04 2.1200e+02
1.9200e+04 2.1300e+02
1.9300e+04 2.1400e+02
1.9400e+04 2.1500e+02
1.9500e+04 2.1600e+02
1.9600e+04 2.1700e+02
1.9700e+04 2.1800e+02
1.9800e+04 2.1900e+02
1.9900e+04 2.2000e+02
2.0000e+04 2.2100e+02
2.0100e+04 2.2200e+02
2.0200e+04 2.2300e+02
2.0300e+04 2.2400e+02
2.0400e+04 2.2500e+02
2.0500e+04 2.2600e+02
2.0600e+04 2.2700e+02
2.0700e+04 2.2800e+02
2.0800e+04 2.2900e+02
2.0900e+04 2.3000e+02
2.1000e+04 2.3100e+02
2.1100e+04 2.3200e+02
2.1200e+04 2.3300e+02
2.1300e+04 2.3400e+02
2.1400e+04 2.3500e+02
2.1500e+04 2.3600e+02
2.1600e+04 2.3700e+02
2.1700e+04 2.3800e+02
2.1800e+04 2.3900e+02
2.1900e+04 2.4000e+02
2.2000e+04 2.4100e+02
2.2100e+04 2.4200e+02
2.2200e+04 2.4300e+02
2.2300e+04 2.4400e+02
2.2400e+04 2.4500e+02
2.2500e+04 2.4600e+02
2.2600e+04 2.4700e+02
2.2700e+04 2.4800e+02
2.2800e+04 2.4900e+02
2.2900e+04 2.5000e+02
2.3000e+04 2.5100e+02
2.3100e+04 2.5200e+02
2.3200e+04 2.5300e+02
2.3300e+04 2.5400e+02
2.3400e+04 2.5500e+02
2.3500e+04 2.5600e+02
2.3600e+04 2.5700e+02
2.3700e+04 2.5800e+02
2.3800e+04 2.5900e+02
2.3900e+04 2.6000e+02
2.4000e+04 2.6100e+02
2.4100e+04 2.6200e+02
2.4200e+04 2.6300e+02
2.4300e+04 2.6400e+02
2.4400e+04 2.6500e+02
2.4500e+04 2.6600e+02
2.4600e+04 2.6700e+02
2.4700e+04 2.6800e+02
2.4800e+04 2.6900e+02
2.4900e+04 2.7000e+02
2.5000e+04 2.7100e+02
2.5100e+04 2.7200e+02
2.5200e+04 2.7300e+02
2.5300e+04 2.7400e+02
2.5400e+04 2.7500e+02
2.5500e+04 2.7600e+02
2.5600e+04 2.7700e+02
2.5700e+04 2.7800e+02
2.5800e+04 2.7900e+02
2.5900e+04 2.8000e+02
2.6000e+04 2.8100e+02
2.6100e+04 2.8200e+02
2.6200e+04 2.8300e+02
2.6300e+04 2.8400e+02
2.6400e+04 2.8500e+02
2.6500e+04 2.8600e+02
2.6600e+04 2.8700e+02
2.6700e+04 2.8800e+02
2.6800e+04 2.8900e+02
2.6900e+04 2.9000e+02
2.7000e+04 2.9100e+02
2.7100e+04 2.9200e+02
2.7200e+04 2.9300e+02
2.7300e+04 2.9400e+02
2.7400e+04 2.9500e+02
2.7500e+04 2.9600e+02
2.7600e+04 2.9700e+02
2.7700e+04 2.9800e+02
2.7800e+04 2.9900e+02
2.7900e+04 3.0000e+02
2.8000e+04 3.0100e+02
2.8100e+04 3.0200e+02
2.8200e+04 3.0300e+02
2.8300e+04 3.0400e+02
2.8400e+04 3.0500e+02
2.8500e+04 3.0600e+02
2.8600e+04 3.0700e+02
2.8700e+04 3.0800e+02
2.8800e+04 3.0900e+02
2.8900e+04 3.1000e+02
2.9000e+04 3.1100e+02
2.9100e+04 3.1200e+02
2.9200e+04 3.1300e+02
2.9300e+04 3.1400e+02
2.9400e+04 3.1500e+02
2.9500e+04 3.1600e+02
2.9600e+04 3.1700e+02
2.9700e+04 3.1800e+02
2.9800e+04 3.1900e+02
2.9900e+04 3.2000e+02
3.0000e+04 3.2100e+02
3.0100e+04 3.2200e+02
3.0200e+04 3.2300e+02
3.0300e+04 3.2400e+02
3.0400e+04 3.2500e+02
3.0500e+04 3.2600e+02
3.0600e+04 3.2700e+02
3.0700e+04 3.2800e+02
3.0800e+04 3.2900e+02
3.0900e+04 3.3000e+02
3.1000e+04 3.3100e+02
3.1100e+04 3.3200e+02
3.1200e+04 3.3300e+02
3.1300e+04 3.3400e+02
3.1400e+04 3.3500e+02
3.1500e+04 3.3600e+02
3.1600e+04 3.3700e+02
3.1700e+04 3.3800e+02
3.1800e+04 3.3900e+02
3.1900e+04 3.4000e+02
3.2000e+04 3.4100e+02
3.2100e+04 3.4200e+02
3.2200e+04 3.4300e+02
3.2300e+04 3.4400e+02
3.2400e+04 3.4500e+02
3.2500e+04 3.4600e+02
3.2600e+04 3.4700e+02
3.2700e+04 3.4800e+02
3.2800e+04 3.4900e+02
3.2900e+04 3.5000e+02
3.3000e+04 3.5100e+02
3.3100e+04 3.5200e+02
3.3200e+04 3.5300e+02
3.3300e+04 3.5400e+02
3.3400e+04 3.5500e+02
3.3500e+04 3.5600e+02
3.3600e+04 3.5700e+02
3.3700e+04 3.5800e+02
3.3800e+04 3.5900e+02
3.3900e+04 3.6000e+02
3.4000e+04 3.6100e+02
3.4100e+04 3.6200e+02
3.4200e+04 3.6300e+02
3.4300e+04 3.6400e+02
3.4400e+04 3.6500e+02
3.4500e+04 3.6600e+02
3.4600e+04 3.6700e+02
3.4700e+04 3.6800e+02
3.4800e+04 3.6900e+02
3.4900e+04 3.7000e+02
3.5000e+04 3.7100e+02
3.5100e+04 3.7200e+02
3.5200e+04 3.7300e+02
3.5300e+04 3.7400e+02
3.5400e+04 3.7500e+02
3.5500e+04 3.7600e+02
3.5600e+04 3.7700e+02
3.5700e+04 3.7800e+02
3.5800e+04 3.7900e+02
3.5900e+04 3.8000e+02
3.6000e+04 3.8100e+02
3.6100e+04 3.8200e+02
3.6200e+04 3.8300e+02
3.6300e+04 3.8400e+02
3.6400e+04 3.8500e+02
3.6500e+04 3.8600e+02
3.6600e+04 3.8700e+02
3.6700e+04 3.8800e+02
3.6800e+04 3.8900e+02
3.6900e+04 3.9000e+02
3.7000e+04 3.9100e+02
3.7100e+04 3.9200e+02
3.7200e+04 3.9300e+02
3.7300e+04 3.9400e+02
3.7400e+04 3.9500e+02
3.7500e+04 3.9600e+02
3.7600e+04 3.9700e+02
3.7700e+04 3.9800e+02
3.7800e+04 3.9900e+02
3.7900e+04 4.0000e+02
3.8000e+04 4.0100e+02
3.8100e+04 4.0200e+02
3.8200e+04 4.0300e+02
3.8300e+04 4.0400e+02
3.8400e+04 4.0500e+02
3.8500e+04 4.0600e+02
3.8600e+04 4.0700e+02
3.8700e+04 4.0800e+02
3.8800e+04 4.0900e+02
3.8900e+04 4.1000e+02
3.9000e+04 4.1100e+02
3.9100e+04 4.1200e+02
3.9200e+04 4.1300e+02
3.9300e+04 4.1400e+02
3.9400e+04 4.1500e+02
3.9500e+04 4.1600e+02
3.9600e+04 4.1700e+02
3.9700e+04 4.1800e+02
3.9800e+04 4.1900e+02
3.9900e+04 4.2000e+02
4.0000e+04 4.2100e+02
4.0100e+04 4.2200e+02
4.0200e+04 4.2300e+02
4.0300e+04 4.2400e+02
4.0400e+04 4.2500e+02
4.0500e+04 4.2600e+02
4.0600e+04 4.2700e+02
4.0700e+04 4.2800e+02
4.0800e+04 4.2900e+02
4.0900e+04 4.3000e+02
4.1000e+04 4.3100e+02
4.1100e+04 4.3200e+02
4.1200e+04 4.3300e+02
4.1300e+04 4.3400e+02
4.1400e+04 4.3500e+02
4.1500e+04 4.3600e+02
4.1600e+04 4.3700e+02
4.1700e+04 4.3800e+02
4.1800e+04 4.3900e+02
4.1900e+04 4.4000e+02
4.2000e+04 4.4100e+02
4.2100e+04 4.4200e+02
4.2200e+04 4.4300e+02
4.2300e+04 4.4400e+02
4.2400e+04 4.4500e+02
4.2500e+04 4.4600e+02
4.2600e+04 4.4700e+02
4.2700e+04 4.4800e+02
4.2800e+04 4.4900e+02
4.2900e+04 4.5000e+02
4.3000e+04 4.5100e+02
4.3100e+04 4.5200e+02
4.3200e+04 4.5300e+02
4.3300e+04 4.5400e+02
4.3400e+04 4.5500e+02
4.3500e+04 4.5600e+02
4.3600e+04 4.5700e+02
4.3700e+04 4.5800e+02
4.3800e+04 4.5900e+02
4.3900e+04 4.6000e+02
4.4000e+04 4.6100e+02
4.4100e+04 4.6200e+02
4.4200e+04 4.6300e+02
4.4300e+04 4.6400e+02
4.4400e+04 4.6500e+02
4.4500e+04 4.6600e+02
4.4600e+04 4.6700e+02
4.4700e+04 4.6800e+02
4.4800e+04 4.6900e+02
4.4900e+04 4.7000e+02
4.5000e+04 4.7100e+02
4.5100e+04 4.7200e+02
4.5200e+04 4.7300e+02
4.5300e+04 4.7400e+02
4.5400e+04 4.7500e+02
4.5500e+04 4.7600e+02
4.5600e+04 4.7700e+02
4.5700e+04 4.7800e+02
4.5800e+04 4.7900e+02
4.5900e+04 4.8000e+02
4.6000e+04 4.8100e+02
4.6100e+04 4.8200e+02
4.6200e+04 4.8300e+02
4.6300e+04 4.8400e+02
4.6400e+04 4.8500e+02
4.6500e+04 4.8600e+02
4.6600e+04 4.8700e+02
4.6700e+04 4.8800e+02
4.6800e+04 4.8900e+02
4.6900e+04 4.9000e+02
4.7000e+04 4.9100e+02
4.7100e+04 4.9200e+02
4.7200e+04 4.9300e+02
4.7300e+04 4.9400e+02
4.7400e+04 4.9500e+02
4.7500e+04 4.9600e+02
4.7600e+04 4.9700e+02
4.7700e+04 4.9800e+02
4.7800e+04 4.9900e+02
4.7900e+04 5.0000e+02
4.8000e+04 5.0100e+02
4.8100e+04 5.0200e+02
4.8200e+04 5.0300e+02
4.8300e+04 5.0400e+02
4.8400e+04 5.0500e+02
4.8500e+04 5.0600e+02
4.8600e+04 5.0700e+02
4.8700e+04 5.0800e+02
4.8800e+04 5.0900e+02
4.8900e+04 5.1000e+02
4.9000e+04 5.1100e+02
4.9100e+04 5.1200e+02
4.9200e+04 5.1300e+02
4.9300e+04 5.1400e+02
4.9400e+04 5.1500e+02
4.9500e+04 5.1600e+02
4.9600e+04 5.1700e+02
4.9700e+04 5.1800e+02
4.9800e+04 5.1900e+02
4.9900e+04 5.2000e+02
5.0000e+04 5.2100e+02
5.0100e+04 5.2200e+02
5.0200e+04 5.2300e+02
5.0300e+04 5.2400e+02
5.0400e+04 5.2500e+02
5.0500e+04 5.2600e+02
5.0600e+04 5.2700e+02
5.0700e+04 5.2800e+02
5.0800e+04 5.2900e+02
5.0900e+04 5.3000e+02
5.1000e+04 5.3100e+02
5.1100e+04 5.3200e+02
5.1200e+04 5.3300e+02
5.1300e+04 5.3400e+02
5.1400e+04 5.3500e+02
5.1500e+04 5.3600e+02
5.1600e+04 5.3700e+02
5.1700e+04 5.3800e+02
5.1800e+04 5.3900e+02
5.1900e+04 5.4000e+02
5.2000e+04 5.4100e+02
5.2100e+04 5.4200e+02
5.2200e+04 5.4300e+02
5.2300e+04 5.4400e+02
5.2400e+04 5.4500e+02
5.2500e+04 5.4600e+02
5.2600e+04 5.4700e+02
5.2700e+04 5.4800e+02
5.2800e+0
```

Se estimó una FDT mediante la aplicación de Matlab de identificación de sistemas (Figura 6.12). Se exigió una FDT con 2 polos porque los sistemas térmicos suelen ser de segundo orden^{1,2}.

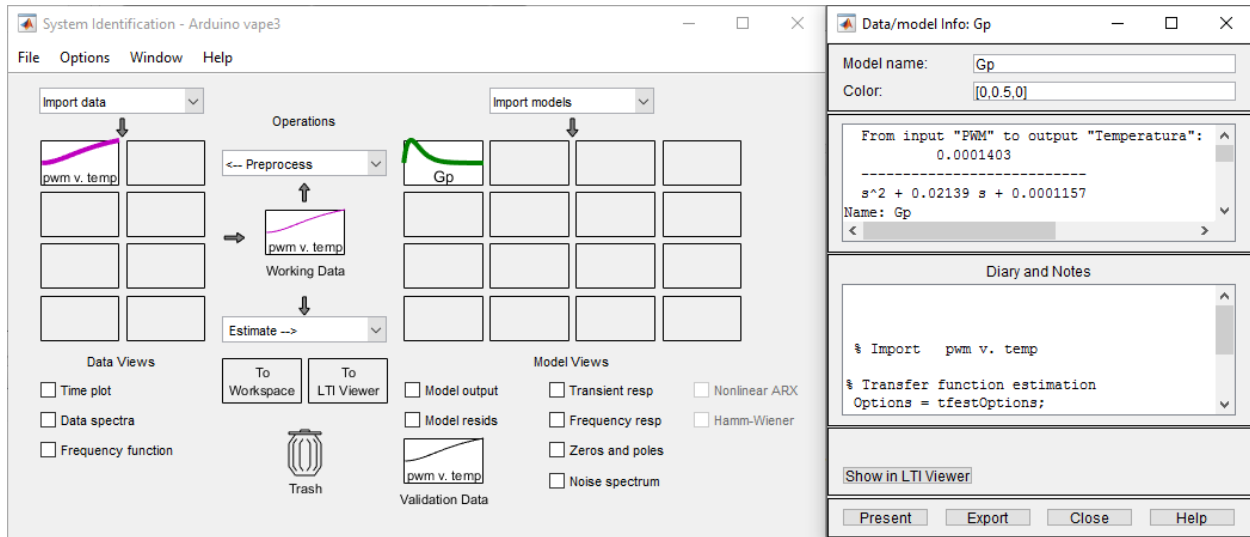


Fig. 6.12 Identificación del sistema Matlab

Este método proporciona una FDT que se compara con su forma genérica para hallar sus parámetros:

$$G_p(s) = \frac{0,0001403}{s^2 + 0,02139s + 0,0001157} = \frac{K_e \omega_n^2}{s^2 + 2\xi \omega_n s + \omega_n^2}$$

De la igualdad se halla la frecuencia natural ω_n , la ganancia estática K_e , y el coeficiente de amortiguamiento ξ :

$$K_e \omega_n^2 = 0,0001403 \Rightarrow K_e = \frac{0,0001403}{\omega_n^2} = \frac{0,0001403}{0,0001157} \Rightarrow K_e = 1,2126188$$

$$\omega_n^2 = 0,0001157 \Rightarrow \omega_n = 0,0107564 \rightarrow$$

$$2\xi \omega_n = 0,02139 \Rightarrow \xi = \frac{0,02139}{2\omega_n} = \frac{0,02139}{2(0,0107564)} \Rightarrow \xi = 0,99429$$

El valor de $K_e = 1,2126188$ se confirma con su respuesta al escalón (Figura 6.18) mientras que los resultados explícitos de la función `damp()` (Figura 6.13) confirman cierto error de redondeo de la $G_p(s)$. Para no arrastrar este error al cálculo de las características dinámicas, se corrigió ω_n y ξ .

```
>> [Wn,zeta,p] = damp(Gp)
Wn =
    1.075766031572346e-02
    1.075766031572346e-02
zeta =
    9.942183916314276e-01
    9.942183916314276e-01
p =
   -1.069546373681581e-02 + 1.155123769536967e-03i
   -1.069546373681581e-02 - 1.155123769536967e-03i
```

$$\omega_n = 0,01075766$$

$$\xi = 0,9942184$$

Fig. 6.13 Función damp() en Matlab

El coeficiente de amortiguamiento $\xi = 0,9942184 \approx 1$ indica que es un sistema prácticamente críticamente amortiguado. Por si acaso, se probó a exigir una FDT críticamente amortiguada, la cual se aproxima a los datos peor (Best Fit = 99.36%) que la FDT subamortiguada (Best Fit = 99.45%) (Figura 6.14).

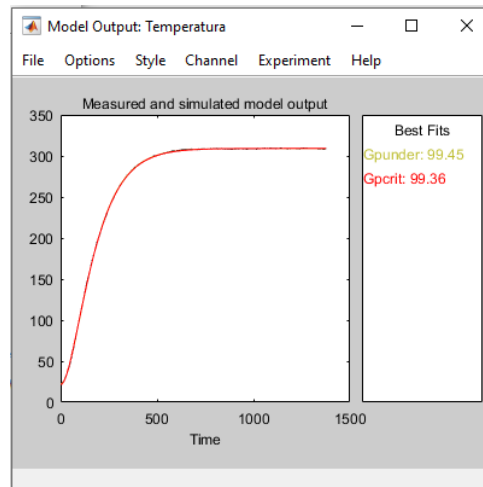


Fig. 6.14 Comparación de sistemas en Matlab

Sus polos conjugados, efectivamente, tienen una parte real negativa ($\sigma = -0,01069 < 0$) y una parte imaginaria muy pequeña ($\omega_d = 0,00115 \approx 0$) los cuales confirman, respectivamente, que es un sistema estable y casi críticamente amortiguado (Figura 6.15):

$$p = \sigma \pm j\omega_d = -\xi\omega_n \pm j\omega_n\sqrt{1-\xi^2} \Rightarrow$$

$$\Rightarrow p = -0,9942184 \cdot 0,01075766 \pm j0,01075766\sqrt{1-0,9942184^2} \Rightarrow$$

$$\Rightarrow p = -0,0106955 \pm j0,00115123$$

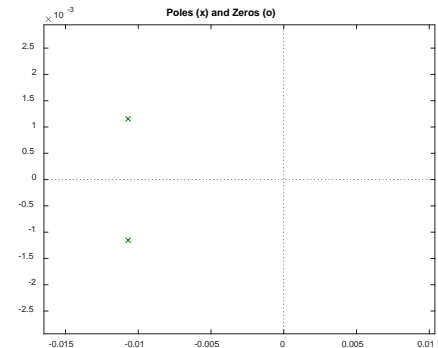


Fig. 6.15 Polos del sistema

Aún con la corrección anterior de ω_n y ξ , las características dinámicas salen con mucho error (Figura 6.16) y se observa que los resultados de `stepinfo()` (Figura 6.17) concuerdan más con la respuesta al escalón (Figura 6.18).

```
>> wn = wn(1)
zeta = zeta(1)
tr = (pi-acos(zeta))/(wn*sqrt(1-zeta^2))
ts = 4/(wn*zeta)
M = exp((-zeta*pi)/sqrt(1-zeta^2))
tp = pi/(wn*sqrt(1-zeta^2))
wn =
    1.075766031572346e-02
zeta =
    9.942183916314276e-01
tr =
    2.626565618645504e+03
ts =
    3.739903288373784e+02
M =
    2.328280443583892e-13
tp =
    2.719702196803655e+03
```

Fig. 6.16 Cálculos de las variables dinámicas

```
>> stepinfo(Gp)
ans =
    struct with fields:
        RiseTime: 3.095248330614304e+02
        SettlingTime: 5.362166527016186e+02
        SettlingMin: 1.093775433275331e+00
        SettlingMax: 1.212008839126195e+00
        Overshoot: 0
        Undershoot: 0
        Peak: 1.212008839126195e+00
        PeakTime: 1.158239427937029e+03
```

Fig. 6.17 Generación directa de las variables dinámicas

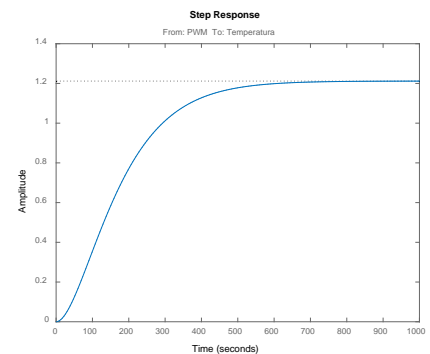


Fig. 6.18 Respuesta al escalón en lazo abierto

Inicialmente se intentó hallar los parámetros del controlador mediante la asignación de polos:

$$G_p(s) = \frac{0,0001403}{s^2 + 0,02139s + 0,0001157}$$

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_p s + K_i + K_d s^2}{s}$$

$$G(s) = \frac{G_c(s) \cdot G_p(s)}{1 + G_c(s) \cdot G_p(s)} = \frac{\left(\frac{K_p s + K_i + K_d s^2}{s}\right) \cdot \left(\frac{0,0001403}{s^2 + 0,02139s + 0,0001157}\right)}{1 + \left(\frac{K_p s + K_i + K_d s^2}{s}\right) \cdot \left(\frac{0,0001403}{s^2 + 0,02139s + 0,0001157}\right)} =$$

$$= \frac{(0,0001403) \cdot (K_p s + K_i + K_d s^2)}{s^3 + (0,02139 + 0,0001403K_d)s^2 + (0,0001157 + 0,0001403K_p)s + 0,0001403K_i}$$

Al añadir un polo en $s = -\alpha\omega_n$, el denominador adquiere la forma $(s + \alpha\omega_n) \cdot (s^2 + 2\xi\omega_n s + \omega_n^2) = s^3 + 2\xi\omega_n(1 + \alpha)s^2 + \omega_n^2(1 + 2\alpha\xi)s + \alpha\omega_n^3$, que al comparar con el denominador anterior produce unos parámetros en función de ω_n y ξ :

$$s^3 + (0,02139 + 0,0001403K_d)s^2 + (0,0001157 + 0,0001403K_p)s + 0,0001403K_i = s^3 + 2\xi\omega_n(1 + \alpha)s^2 + \omega_n^2(1 + 2\alpha\xi)s + \alpha\omega_n^3 \Rightarrow$$

$$0,02139 + 0,0001403K_d = 2\xi\omega_n(1 + \alpha) \Rightarrow K_d = \frac{2\xi\omega_n(1 + \alpha) - 0,02139}{0,0001403}$$

$$0,0001157 + 0,0001403K_p = \omega_n^2(1 + 2\alpha\xi) \Rightarrow K_p = \frac{\omega_n^2(1 + 2\alpha\xi) - 0,0001157}{0,0001403}$$

$$0,0001403K_i = \alpha\omega_n^3 \Rightarrow K_i = \frac{\alpha\omega_n^3}{0,0001403}$$

Se busca una reducción de en el tiempo de establecimiento $t_s = 536$ s y de subida $t_r = 310$ s a un 25% de su valor, por lo que se busca una $t_s = 94,15$ s y $t_r = 59,76$ s:

$$t_s = \frac{4}{\omega_n \xi} \Rightarrow 94,15 = \frac{4}{\omega_n \xi} \Rightarrow \xi = \frac{4}{94,15\omega_n}$$

$$t_r = \frac{\pi - \cos^{-1}(\xi)}{\omega_n \sqrt{1 - \xi^2}} \Rightarrow 59,76 = \frac{\pi - \cos^{-1}(\xi)}{\omega_n \sqrt{1 - \xi^2}} \Rightarrow \omega_n = \frac{\pi - \cos^{-1}(\xi)}{59,76 \sqrt{1 - \xi^2}}$$

```
>> syms z w
eqn1 = z == 4/(94.15*w);
eqn2 = w == (pi-acos(z))/(59.76*sqrt(1-z^2));
sol = solve([eqn1, eqn2], [z, w]);
zSol = sol.z
wSol = sol.w
zSol =
0.72856597488413467633790847470862
wSol =
0.058313724645188769544370882747737
```

Fig. 6.19 Determinación de los nuevos parámetros ξ y ω_n

Con una $\omega_n = 0,05314$ y $\xi = 0,72857$, y eligiendo una $\alpha = 10$, se hallan los parámetros del controlador:

$$K_p = \frac{(0,05314^2)(1 + 2(10)(0,72857)) - 0,0001261}{4,378 \cdot 10^{-5}} = 1001,49$$

$$K_i = \frac{(10)(0,05314^3)}{4,378 \cdot 10^{-5}} = 34,28$$

$$K_d = \frac{2(0,72857)(0,05314)(1 + 10) - 0,01918}{4,378 \cdot 10^{-5}} = 19017,28$$

Estos valores tan altos no producían resultados satisfactorios. La K_p resultaba sobreoscilaciones muy exageradas y la K_d producía valores de PWM erráticos y muy altos, aún cuando se había sobrepasado el SP, y no permitía bajar la temperatura. Además, la K_d causaba errores al leer la variable **Input** (la PV) de la función PID.

Por ello se recurrió a la aplicación de PID de Matlab, con la que se buscaba un tiempo de establecimiento rápido, pero sin grandes sobreoscilaciones para reducir el desperdicio de energía (Figura 6.20).

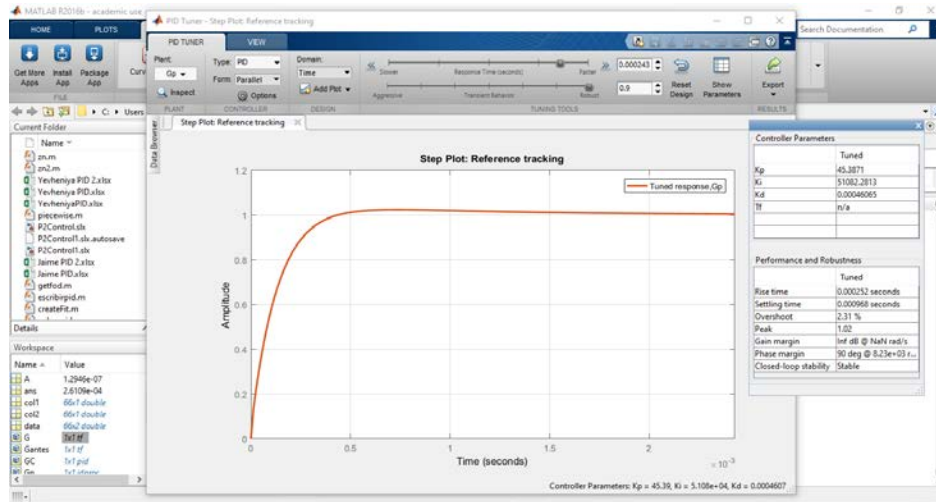


Fig. 6.20 PID Tuner de Matlab

El controlador obtiene la forma $G_c(s) = K_p + \frac{K_i}{s} + K_d s = 45.4 + \frac{51082}{s} + 0.000461s$ y junto con la $G_p(s)$ se halla el lazo cerrado $G(s) = \frac{G_c(s)G_p(s)}{1+G_c(s)G_p(s)} = 17134 \frac{(s+9.739 \cdot 10^4)(s+1139)}{(s+2.136 \cdot 10^5)(s+7395)(s+1203)}$:

```
>> Gc
Gc =
      1
  Kp + Ki * ---- + Kd * s
      s

with Kp = 45.4, Ki = 5.11e+04, Kd = 0.000461
Continuous-time PID controller in parallel form.
```

Fig. 6.21 Controlador en Matlab

```
>> G = feedback(Gc*Gp, 1)
G =
From input "t" to output:
  17134 (s+9.739e04) (s+1139)
-----
(s+2.136e05) (s+7395) (s+1203)
Continuous-time zero/pole/gain model.
```

Fig. 6.22 Lazo cerrado en Matlab

Las nuevas características dinámicas implican una mejora:

Lazo abierto $G_p(s)$	Lazo cerrado $G(s)$
$t_r = 2,8492 \cdot 10^{-3} s$	$t_r = 2,5183 \cdot 10^{-4} s$
$t_s = 5,0782 \cdot 10^{-3} s$	$t_s = 9,6829 \cdot 10^{-4} s$
$M = 0\%$	$M = 2\%$
$t_p = 1,367 \cdot 10^{-3} s$	$t_p = 7,3481 \cdot 10^{-4} s$

```
>> stepinfo(G)
ans =
struct with fields:
    RiseTime: 2.5183e-04
    SettlingTime: 9.6829e-04
    SettlingMin: 9.0101e-01
    SettlingMax: 1.0231e+00
    Overshoot: 2.3075e+00
    Undershoot: 0
    Peak: 1.0231e+00
    PeakTime: 7.3481e-04
```

Fig. 6.23 Características dinámicas del lazo cerrado

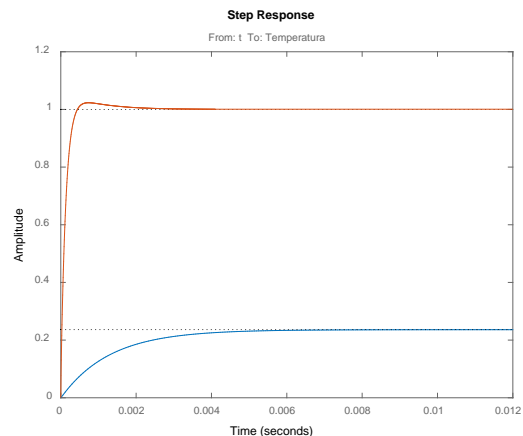


Fig. 6.24 Lazo abierto (azul) y cerrado (naranja)

Desafortunadamente, estos valores producían efectos similarmente erráticos o exagerados, posiblemente porque los cálculos teóricos no representan bien al sistema real, cuya inercia térmica parecía muy difícil de controlar. Por ello se acabó dependiendo del método de prueba y error. Para ellos, unas pautas básicas para sistemas térmicos³ recomiendan empezar por un aumento de K_p . Efectivamente, aumentaba las oscilaciones, pero reducía el error (Figura 6.25). Fijando un valor de K_p , en principio satisfactorio, se aumentó gradualmente la K_i , la cual reducía el error en el régimen permanente y aumentaba la sobreoscilación (Figura 6.26). Fijado los dos anteriores parámetros y aumentando la K_d , se reduce la sobreoscilación considerablemente (Figura 6.27).

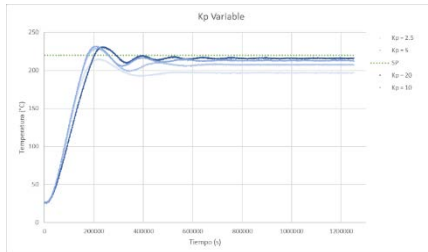


Fig. 6.25 Aumento de K_p

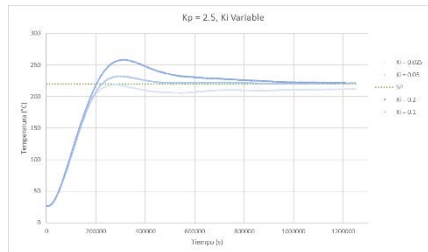


Fig. 6.26 Aumento de K_i

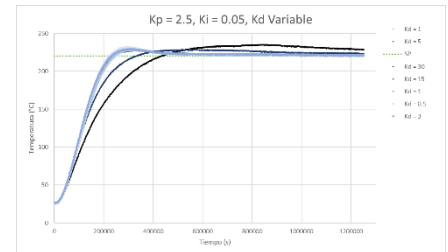


Fig. 6.27 Aumento de K_d

Estos perfiles permitían formar una idea de los efectos de cada parámetro sobre el sistema. De las distintas combinaciones (Figura 6.28), se optó por $K_p = 3.2$, $K_i = 0.095$, y $K_d = 4.5$.

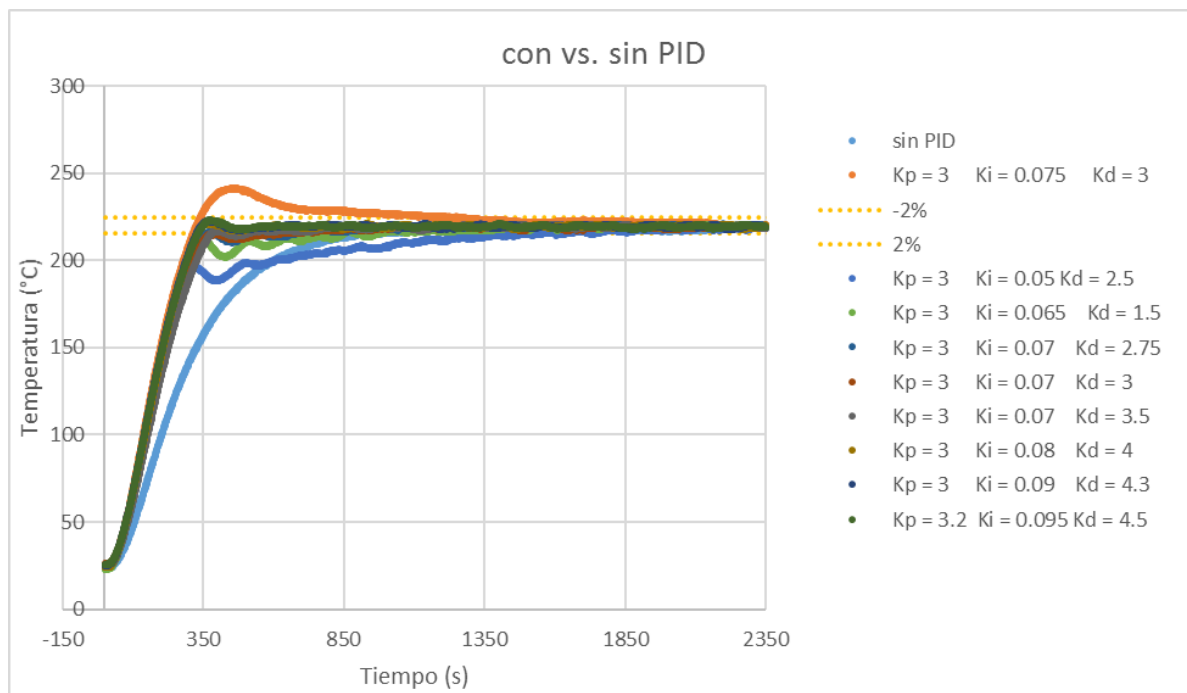


Fig. 6.28 Combinaciones de parámetros PID

Referencias

1. https://www.ncnr.nist.gov/staff/dimeo/riddle/temp_control_presentation.pdf
2. <https://ocw.mit.edu/courses/mechanical-engineering/2-003-modeling-dynamics-and-control-i-spring-2005/readings/notesinstalment2.pdf>
3. <http://forum.seemecnc.com/viewtopic.php?t=11440>

7 Comunicaciones

Descripción

El vaporizador, basado en un Arduino Uno, se comunica por el bus I²C con un Arduino Uno Wifi. El Uno envía datos de temperatura al Uno Wifi, el cual publica estos datos por MQTT a una Raspberry Pi para su monitoreo y almacenaje. A su vez, el Uno Wifi se suscribe para recibir unos comandos de encendido/apagado y de regulación de la temperatura para enviarlos al Uno (Figura 7.0).

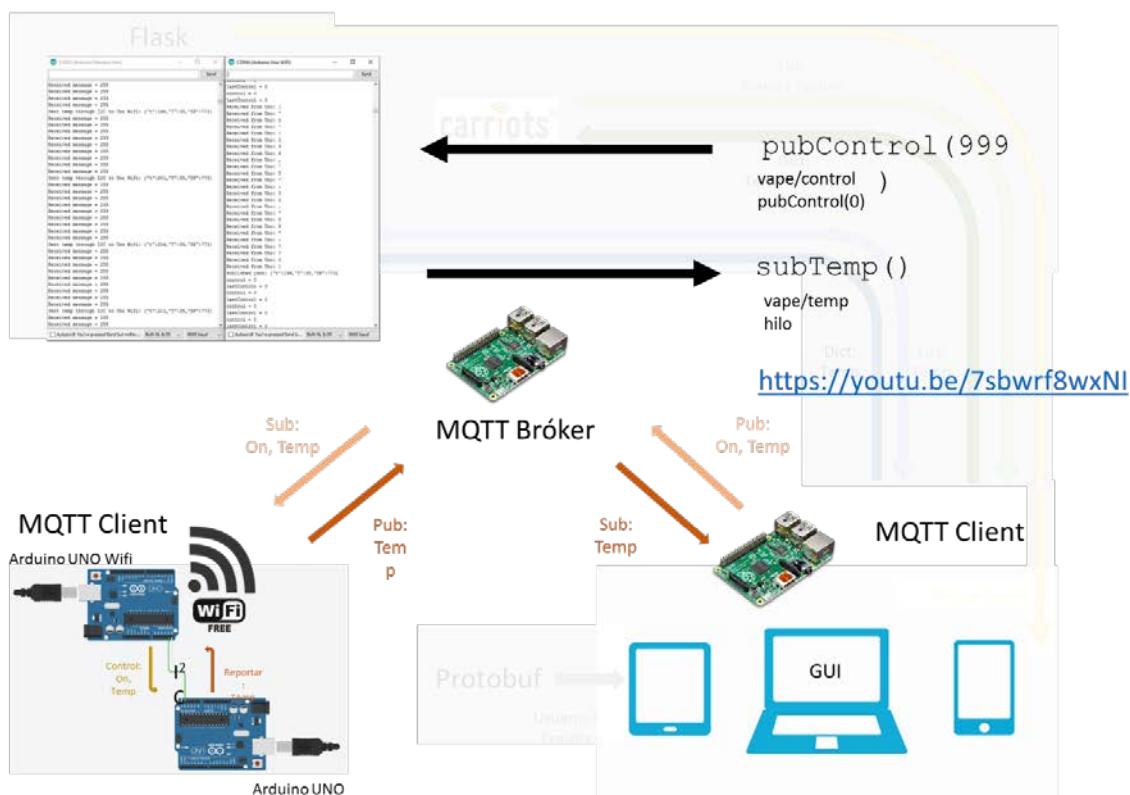


Fig. 7.0 Comunicación por I²C y MQTT

El Uno calcula el tiempo en segundos de la medida de temperatura que va a mandar, asegurándose de comparar con el mismo tiempo de referencia. A continuación, con la librería `ArduinoJson.h`¹ se reservan 256 bytes para este array de caracteres. Se crea el objeto `json = {"t": tiempo, "T": temperatura, "SP": SPactual}` y se rellena el `jsonArray` con estos contenidos (Figura 7.1).

```

9 void enviarTemp()
10 {
11     if (tRefTemp == 0)
12     {
13         tRefTemp = millis();
14     }
15
16     int tiempo = int((millis() - tRefTemp)/1000);    // calcular intervalo en segundos
17     int temperatura = temp();
18
19     StaticJsonBuffer<256> jsonBuffer;                // reservar memoria para el JSON (array de caracteres)
20
21     JsonObject json = jsonBuffer.createObject();      // crear objeto JSON = {"t": tiempo, "T": temperatura, "SP": SPactual}
22     json["t"] = tiempo;
23     json["T"] = temperatura;
24     json["SP"] = SPactual;
25
26     json.printTo(jsonArray, sizeof(jsonArray));      // rellenar array de caracteres
27
28
29     Wire.beginTransmission(1);                      // transmitir a dispositivo #1
30     Wire.write(jsonArray);                          // enviar JSON por I2C
31     Wire.endTransmission();                         // finalizar transmisión
32
33     Serial.print("Temperatura enviada por I2C a Uno Wifi: ");
34     Serial.println(jsonArray);
35 }

```

Fig. 7.1 Enviar temperatura por I²C

Para enviar el `jsonArray` por I²C se conectan las patillas A4 (SDA: los datos) y A5 (SCL: el reloj) entre los Arduinos, así como la tierra común². La Figura 7.2 muestra los bits de la línea SDA, que empiezan a leerse con un flanco negativo de mientras SCL = 1. Dada esta condición de Start, los primeros 7 bits es la dirección del esclavo, seguido por la acción escribir (R/W = 0) o leer (R/W = 1) del esclavo, y terminando el primer byte con un bit de Acknowledge, usado por el esclavo para indicar la recepción exitosa de la secuencia anterior. Los siguientes 8 bits suelen son para los registros internos del esclavo, por ejemplo, cuando se quiere leer el eje X de un acelerómetro de 3 ejes, terminando también con un bit ACK. Los siguientes grupos de 8 bits constituyen los datos del mensaje, que se envía hasta que se recibe la señal de terminación Stop, que ocurre con un flanco positivo de SDA = 1 mientras SCL = 1³.

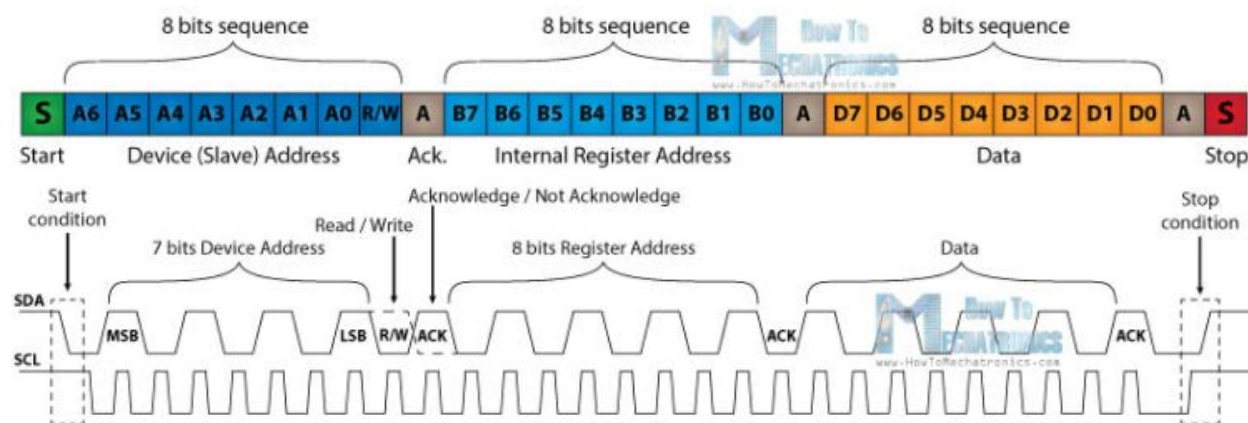


Fig. 7.2 Líneas SDA y SCL del bus I²C³

Con la librería `Wire.h` se envía el array del json mediante la comunicación I²C⁴, habiéndose asignado en el `setup()` como maestro ya que su función `Wire.begin()` carece de parámetros, es decir, de una dirección. En la función `enviarTemp()` se inicia la transmisión, se envía el array, y se cierra la transición (Figura 7.1).

En el otro extremo, el Arduino Uno Wifi se establece en el `setup()` como un esclavo en la dirección 1 del bus I²C y prepara las funciones a ejecutar en caso de recepción o petición de datos por este bus (Figura 7.3). Al recibir datos, se llama a la función `receiveEvent()`, la cual recibe el array por I²C de char en char y los va almacenando en `json[]` (Figura 7.4). Va avanzando por el array ya que se mantiene en el bucle de `Wire.available()` mientras siga detectando un flujo de datos por el bus I²C, aumentando el índice en cada vuelta. Con la marca booleana `publicado`, el Arduino Uno Wifi evita la doble publicación de un mismo json.

```
17 void setup()
18 {
19   Serial.begin(9600);
20
21   Wire.begin(1);           // establecerse como esclavo I2C en la dirección 1
22   Wire.onReceive(receiveEvent); // ejecutar función al recibir por I2C
23   Wire.onRequest(requestEvent); // ejetura
24
25   Ciao.begin();           // mqtt
26 }
```

Fig. 7.3 Función `setup()` del Arduino Uno Wifi

```
6 int receiveEvent(int bytes)
7 {
8   int i = 0;
9
10  while (Wire.available())
11  {
12    json[i] = Wire.read(); // leer los bytes en el I2C y almacenar en un array
13
14    Serial.print("Recibido del Uno: ");
15    Serial.println(json[i]);
16
17    i++;
18    publicado = false; // indica que hay que publicar estos datos
19  }
20 }
```

Fig. 7.4 Recibir temperatura por I²C

La Figura 7.5 muestra como el Arduino Uno (COM5) envía el array del json para que el Arduino Uno Wifi (COM4) lo recibe de char en char para formar el json a publicar.

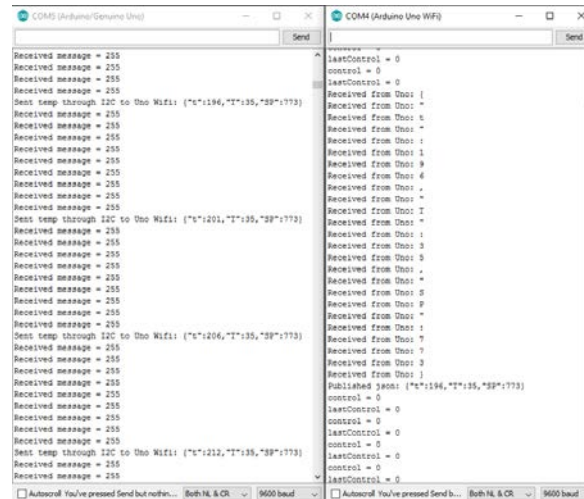


Fig. 7.5 Traspaso de JSON de Arduino Uno a Uno Wifi

En el `loop()` se llama a la función `mqttPub()` sólo si la marca `publicado = false` indica que este json todavía no se ha publicado (Figura 7.6). Esta función de la librería `Ciao`⁵ utiliza el connector “mqtt” para publicar el array al topic “vape/temp” e indicar que ha publicado algo con `publicado = true` (Figura 7.7).

```
29 void loop()
30 {
31   delay(500);
32
33   // Publicar json sólo si no se ha publicado ya
34   if (publicado == false)
35   {
36     mqttPub(json);
37   }
38
39   // Suscribirse a comandos remotos
40   control = mqttSub();
41
42   Serial.print("control = ");
43   Serial.println(control);
44   Serial.print("lastControl = ");
45   Serial.println(lastControl);
46 }
```

Fig. 7.6 Recibir temperatura por I²C

```
6 void mqttPub(char jsonArray[])
7 {
8   Ciao.write(CONNECTOR, TOPIC1, jsonArray); // #define CONNECTOR "mqtt"
9                                             // #define TOPIC1 "vape/temp"
10
11   Serial.print("Publicado json: ");
12   Serial.println(jsonArray);
13
14   publicado = true; // indica que ya se han publicado los datos
15 }
```

Fig. 7.7 Publicar temperatura por MQTT

A la misma vez, la función `mqttSub()` está en el `loop()` constantemente subscribiéndose a el topic "vape/control" a la espera de comandos remotos (Figura 7.6). Si se recibe un dato, se guarda en un char y posteriormente se convierte a un int (Figura 7.8).

```

7 int mqttSub()
8 {
9     CiaoData data = Ciao.read(CONNECTOR, TOPIC2);           // #define CONNECTOR "mqtt"
10                                     // #define TOPIC2 "vape/control"
11     if (!data.isEmpty())
12     {
13         const char* value = data.get(2);                   // recibir comando en formato char
14
15         int valueInt = atoi(value);                         // convertir char a int
16
17         Serial.print("Subscrito y recibido valor int = ");
18         Serial.println(valueInt);
19
20         return valueInt;
21     }
22 }

```

Fig. 7.8 Suscribirse por MQTT

Este valor entero se devuelve para guardarlo en la variable `control` (Figura 7.6). El Arduino Uno Wifi envía esta variable por I²C como esclavo⁶ mediante la función `requestEvent()` (Figura 7.9). Esta función comprueba que el mensaje actual y último mensaje enviado son distintos para evitar duplicados y además comprueba que el último valor recibido es un cero. Se crea un array de 2 bytes llamado `myArray[]` y en el primero se introduce el segundo byte de `control` mediante un traslado de 8 bits (operador Bitshift⁷) y una operación AND bit a bit y el segundo byte `myArray[]` se llena con el primer byte de `control`⁸.

```

7 void requestEvent()
8 {
9     if (control != lastControl && lastControl == 0)
10     {
11         byte myArray[2];
12
13         myArray[0] = (control >> 8) & 0xFF;
14         myArray[1] = control & 0xFF;
15
16         Wire.write(myArray, 2);
17
18         Serial.print("Enviado por I2C a Uno: ");
19         Serial.print(myArray[0]);
20         Serial.println(myArray[1]);
21     }
22
23     lastControl = control;
24 }

```

Fig. 7.9 Enviar control por I²C

$(control \gg 8) \& 0xFF$	$=$	00000000	00000011	
		00000000	11111111	
		00000000	00000011	
				myArray[0]
$control \& 0xFF$	$=$	00000011	11100111	
		00000000	11111111	
		00000000	11100111	
				myArray[1]
				myArray[] = 00000011 11100111 = 999

Fig. 7.10 Ejemplo de operaciones con bits para enviar control

La Figura 7.10 muestra un ejemplo de esta manipulación de bits para el caso en que `control` = 999, es decir, los bytes 00000011 11100111. La función `requestEvent()` envía los 2 bytes del array por el bus I²C y almacena el valor del último mensaje (Figura 7.9).

Por último, la función `recibirControl()` permite al Arduino Uno recibir el comando proveniente del Arduino Uno Wifi por el bus I²C (Figura 7.11). Esto consiste en exigir 2 bytes al maestro del esclavo en la dirección 1 con `Wire.requestFrom(1, 2)` y esperar a que lleguen los dos, los cuales se van guardando cada uno en el byte `a` o `b`. La variable `message` es un entero que almacena en su segundo byte el byte `a` y traslada al byte `b` 8 bits a su izquierda para almacenarlo en su primer byte con una operación OR bit a bit⁸.



```

39 void recibirControl()
40 {
41     byte a,b;
42
43     Wire.requestFrom(1, 2);    // exigir 2 bytes del esclavo #1
44
45     while (Wire.available())
46     {
47         a = Wire.read();        // leer el primer byte
48         b = Wire.read();        // leer el segundo byte
49
50         message = a;            // almacenar el primer byte leído en el segundo byte de message
51         message = message << 8 | b; // almacenar el segundo byte leído en el primer byte de message
52
53         Serial.print("Recibido del Uno Wifi el message = ");
54         Serial.println(message);
55
56         switch (message)
57         {
58             case 999:           // comando ON
59             {
60                 onOff();        // apagar o encender el vaporizador
61                 break;
62             }
63             case 255:           // ignorar valor 255
64             {
65                 break;
66             }
67             case 300:           // convertir valor 300 en 255
68             {
69                 message = 255;
70                 enterSP(message); // asignar valor a SPactual
71                 break;
72             }
73             default:
74                 enterSP(message); // asignar valores distintos a 999, 255, y 300 a SPactual
75                 break;
76         }
77     }
78
79     delay(500);
80 }

```

Fig. 7.11 Recibir control por I²C

Siguiendo el ejemplo anterior donde `myArray[] = 00000011 11100111`, la Figura 7.10 muestra su almacenaje en la variable `message`. En el caso de que esta variable valga 999, el sistema se apagará o encenderá con la función `onOff()`. Un valor de 255 se descarta, por tratarse de un valor que por defecto aparece cuando el maestro no recibe nada². En su lugar, se utiliza el valor de 300, que nunca puede asignarse como SP por ser excesivamente alto, para representar al valor de 255. De esta forma, los valores distintos al 999 se introducen en la función `enterSP()`, que asignará el valor a la `SPactual` (Sección 3).

```

a = 00000011
b = 11100111

```

```

message = a = 00000000 00000011
message = message << 8 = 00000011 00000000
message = message << 8 | b = 00000011 00000000
                             00000000 11100111
                             00000011 11100111
                             -----
                             message = 999

```

Fig. 7.10 Ejemplo de operaciones con bits para recibir control

Proceso

La idea original era conectar un módulo ESP8266 (en lugar del Arduino Uno Wifi) al Arduino Uno, pero no fue posible establecer la comunicación pese a distintos intentos de cableado y de actualización del firmware ESP. Para este proyecto el Arduino Uno Wifi sirve como una representación simbólica del módulo ESP8266. Es más, el Uno Wifi lleva integrado este mismo módulo.

En el `loop()`, se incluye un `delay()` de 500 milisegundos ya que con un `millis()` la comunicación por MQTT no parecía funcionar. Se sospecha que el procesador debe dormirse, aunque sea medio segundo para reducir el ruido o la confusión entre las funciones de publicar y suscribir.



El Arduino Uno parecía recibir mensajes I²C por duplicado por lo que hay que vigilar el último valor recibido. Además, la comunicación por MQTT se beneficia del envío de una señal de 0 entre mensajes.

Por último, el Arduino Uno recibe un valor de 255 constantemente siempre que no reciba un mensaje I²C del Arduino Uno Wifi. Por ello se debe burlar este efecto asignándole ningún comando al mensaje de 255 y en su lugar utilizando un mensaje de 300 como un comando de 255.

Referencias

1. <https://bblanchon.github.io/ArduinoJson/doc/encoding/>
2. <http://www.instructables.com/id/I2C-between-Arduinos/>
3. <http://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/>
4. <https://www.arduino.cc/en/Tutorial/MasterWriter>
5. <http://www.arduino.org/learning/reference/ciao-write>
6. <https://www.arduino.cc/en/Tutorial/MasterReader>
7. <https://www.arduino.cc/en/Reference/Bitshift>
8. <https://thewanderingengineer.com/2015/05/06/sending-16-bit-and-32-bit-numbers-with-arduino-i2c/>
9. <http://forum.arduino.cc/index.php?topic=139286.0>