SENSORES Y ACTUADORES

GRADO EN INGENIERIA DE ROBOTICA SOFTWARE
(FUENLABRADA)

Colección de ejercicios prácticos y problemas
de apoyo al material de lectura de la asignatura

Material docente en abierto de la Universidad Rey Juan Carlos

ÍNDICE

Prácticas de programación

Problemas de apoyo al material de lectura

Práctica 1

Introducción a la Raspberry Pi y al mecanismo PWM para modular una señal digital (LED)

Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Introducción

El mundo se está automatizando, con grandes cantidades de datos producidos y procesados para fines analíticos, de control y de conexión. La Raspberry Pi puede proporcionar una amplia gama de automatización y procesamientos de datos si se explota todo su potencial. Esta pequeña placa ofrece amplias funcionalidades y oportunidades para cambiar el mundo que nos rodea. Esta práctica es el primer paso para hacerlo...

Dada la gran capacidad de esta placa para interactuar con el mundo exterior, las principales aplicaciones que se pueden desarrollar usando Raspberry Pi son aquellas en las que se usan los diferentes sensores disponibles para esta y se convierten los datos vertidos por estos en datos útiles para analizar y controlar los dispositivos que vamos a experimentar durante el curso.

2. Descripción de la placa

Lo primero de todo es identificar los diferentes conectores y elementos de la placa. Para ello, observemos la imagen 1.

- Puertos GPIO: GPIO significa Entrada/Salida de Propósito General. Los vamos a usar mucho durante el curso, así que ya los veremos en detalle en la siguiente práctica. Lo mejor de estos puertos es que se les puede asignar una tarea específica según el GPIO específico del programa. Podremos leer valores de cualquier periférico, como sensores o actuadores, así como enviar los valores calculados en nuestros programas. También podremos conectar LEDs o pantallas LCD. Estos puertos, sin duda, marcan la gran diferencia entre la Raspberry Pi y cualquier otra placa microcontroladora, al dar a los desarrolladores la absoluta libertad de crear.
- Conector de salida de audio de 3,5 mm: si no se usa la conexión HDMI (que describimos a continuación), el audio se puede reproducir a través de altavoces o auriculares utilizando este conector estándar de 3,5 mm.
- USB: este es el conector más común, ampliamente utilizado en cualquier ordenador, y por lo tanto llamado Universal Serial Bus. Puedes conectar un pendrive, teclado, ratón e incluso un concentrador USB con alimentación externa para poder conectar más periféricos USB.
- Ethernet: esta es una conexión muy importante, además, claro está, de la conexión por Wi-Fi que ofrece este modelo, pues nos es de utilidad para tener un control remoto sobre la Raspberry Pi, así como para proporcionar conexión a Internet por cable. No siempre podemos conectar la placa a una pantalla dedicada, por lo que usamos el inicio de sesión remoto y así vemos todo el Escritorio y el interfaz de línea de comandos de la Pi en la pantalla de nuestro equipo.

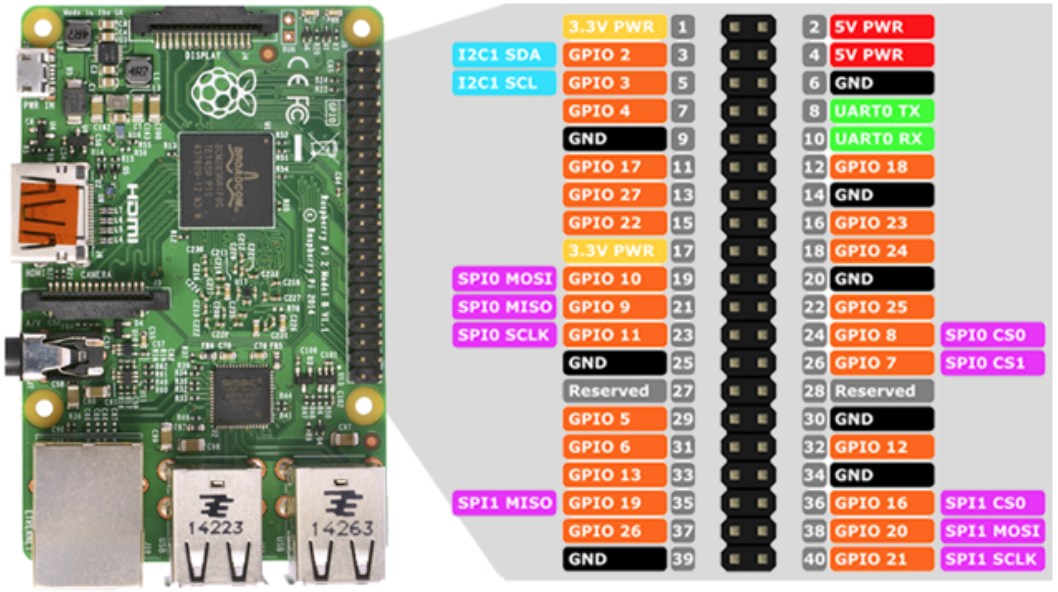

Figura 1: Elementos e interfaz GPIO en Raspberry Pi

- Conector de cámara CSI: la placa no incluye la cámara, pero sí incorpora un conector CSI para esta, la PiCamera, que se puede adquirir por separado. La cámara incluye un cable flexible de 15 cm (cuanto más largo es, hay más posibilidad de perder calidad en la imagen). La cámara tiene 8 megapíxeles y se puede usar, gracias a este puerto, para grabar vídeos de alta definición, así como fotografías fijas y, como no, desarrollar algoritmos de visión computacional.
- Conector HDMI: la interfaz multimedia de alta definición (HDMI) es un interfaz de audio/vídeo compacta utilizada para transferir datos multimedia sin comprimir. Gracias a este, puedes conectar una pantalla que disponga de esta moderna conexión para ver la imagen en alta definición. Si se usa este conector, ya no es necesario conectar los altavoces a la toma de audio.
- Micro USB: a través de este puerto, la placa recibe la energía necesaria para funcionar. Necesita de un voltaje de entrada de 5V y un mínimo de intensidad de 2'5A, aunque este último valor depende de los dispositivos que estén conectados a la Raspberry. Cabe añadir que, como se puede apreciar, la placa no tiene botón de encendido; la placa se enciende simplemente enchufando el adaptador de corriente.
- Ranura para microSD: la microSD es importante porque es donde la Raspberry tiene instalado su sistema operativo. También es donde almacenará todos nuestros documentos, programas, etc. Es el disco duro de la placa. En cuanto a la RAM, viene integrada en la placa.

3. Instalación del Sistema Operativo

Para instalar un sistema operativo necesitamos una tarjeta microSD donde instalarlo. Una vez que tenemos la microSD, vamos a proceder a descargar e instalar una imagen de sistema operativo válido para Raspberry Pi. Existen numerosas distribuciones. Nosotros vamos a usar la más estandarizada para esta placa: `Raspberry Pi OS`. Para ello, seguimos estos pasos:

1. Descargar del siguiente enlace la imagen que incluye la parte de escritorio (*Raspberry Pi OS with desktop and recommended software*):
   https://www.raspberrypi.org/software/operating-systems/ (3G).
2. Mientras, vamos formateando la microSD. Para ello, usamos la herramienta `Discos` de Linux. Le damos formato *Fat32*.

3. Una vez se haya descargado la imagen, volvemos a usar `Discos` con la opción de *Restaurar imagen*. Seleccionamos la que acabamos de descargar y empezamos su restauración en la microSD.

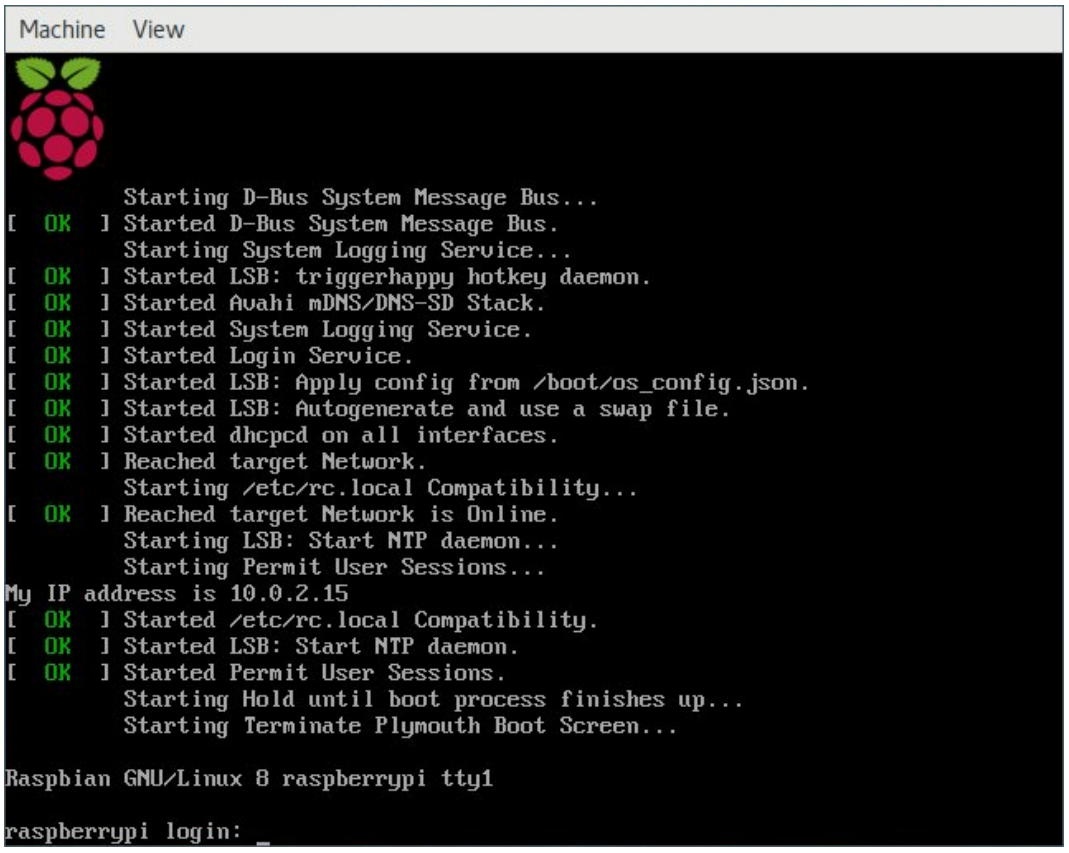

Figura 2: Arranque de Raspberry Pi OS

Cuando acabe la restauración, podemos insertar la microSD en la Raspberry para que pueda arrancar. Para ello, conectamos el cargador y debería aparecer la Figura 2.

Y, si todo ha ido bien, y dado que hemos instalado la versión con escritorio, nos debería aparecer directamente la pantalla de la Figura 3 con nuestro sistema ya funcionando.

Por último, comentar que, por defecto, Raspberry Pi OS establece los siguientes datos de acceso:

- Usuario: pi (que es *root*, por cierto).
- Contraseña: raspberry.

4. Introducción a la técnica PWM

Para empezar a trastear con la placa vamos a aprender a conectar y controlar un diodo emisor de luz (LED) usando la Raspberry Pi. Lo primero que hemos de tener en cuenta es que todos los pines GPIO de la Raspberry Pi son digitales. Por ello, a priori, la conexión de un LED a cualquier pin GPIO, solo nos va a permitir encenderlo o apagarlo. Ya veremos más en profundidad la electrónica que hay detrás de los puertos GPIO.

De modo que, para poder controlar el brillo del LED como si de una señal analógica se tratara, hemos de emplear algún artificio. Y ese artificio se llama PWM, cuyas siglas provienen de *Pulse Width Modulation*

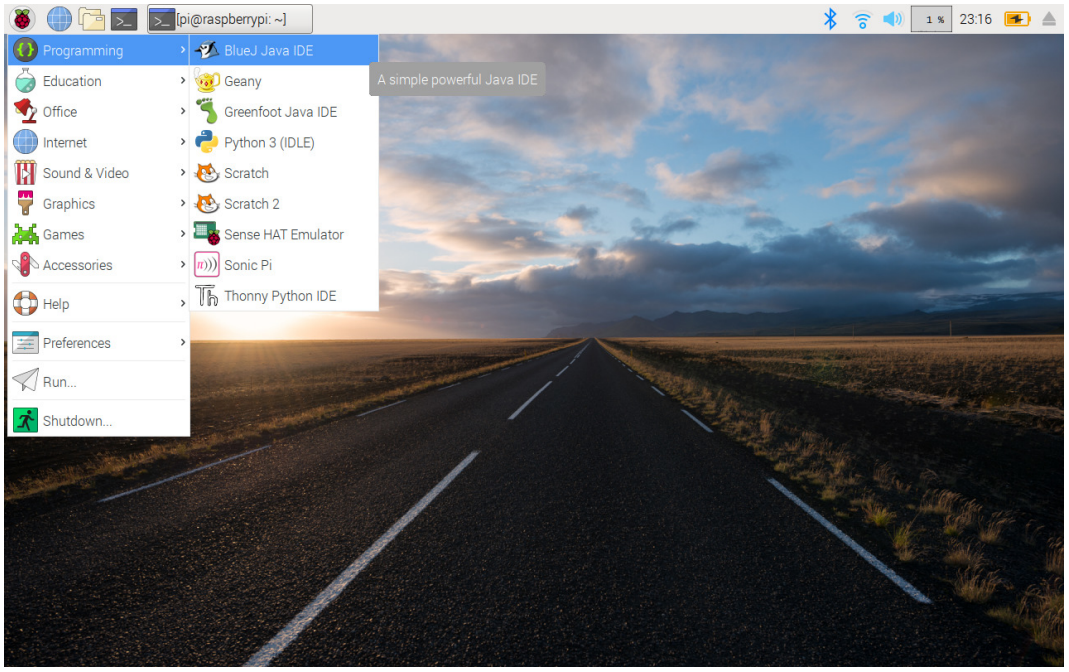

Figura 3: Aspecto del interfaz de escritorio de Raspberry Pi OS

o, en castellano, modulación de ancho de pulso. Si no empleamos este mecanismo, solo podríamos hacer dos cosas: encender (estado del pin HIGH) o apagar (LOW) el LED. Gracias al PWM podremos simular voltajes analógicos a través de los pines digitales. De modo que, considerando los pines GPIO de la Raspberry —que son de $3{,}3V$— si por ejemplo queremos inyectar una señal de $1{,}65V$, hemos de encender y apagar el pin de forma que este esté encendido la mitad del tiempo, y la otra mitad apagado. Asimismo, si queremos simular una salida analógica de $1{,}1V$, deberíamos tomar tiempos para que la señal de $3{,}3V$ estuviera encendida el 33 % del tiempo. Como vemos, este enfoque es muy práctico para muchas aplicaciones como la que pretendemos: controlar el brillo de un LED.

Para hacer uso de este concepto, en Raspberry hemos de considerar la tensión como una señal con una frecuencia y un ciclo de trabajo. Así por ejemplo si consideramos una señal con una frecuencia de 100 Hz, tendría un período de 10 milisegundos o, en otras palabras, la señal se repite cada 10 milisegundos. Si la señal tuviera un ciclo de trabajo del 100 %, sería *Alta* el 100 % del tiempo y *Baja* el 0 % del tiempo. Si tuviera un ciclo de trabajo del 50 %, sería *Alta* el 50 % del tiempo (0.5x10 milisegundos = 5 milisegundos) y *Baja* el 50 % del tiempo (0.5x10 milisegundos = 5 milisegundos). Por lo que obtenemos que, con una señal de 100 Hz de frecuencia, durante un periodo total de 10 milisegundos, la señal sería *Alta* 5ms y *Baja* otros tantos.

5. Instalación del LED

Pasamos a instalar y usar el LED. Lo primero que debemos tener en cuenta es la polaridad de un LED. Si nos fijamos con el en la mano, vemos que hay un pin más corto que otro (Figura 4), y que además, en ese mismo lado el encapsulado del LED presenta una muesca o marca plana. Pues bien, este lado es el negativo (cátodo); por contra, el pin más largo (y sin muesca) es el ánodo o polo positivo.

Recordemos que el ánodo y cátodo de un dispositivo depende de si el dispositivo en cuestión consume o aporta energía. Siendo el ánodo el electrodo por el cual entra la energía al dispositivo; y el cátodo, por el que sale. Así, en una batería, en modo *aportar energía* (no en modo cargando), la energía sale del positivo y entra en el negativo; es por ello que el ánodo es el negativo y, el cátodo, el positivo. Por contra, en un diodo LED emisor de luz, que consume energía, la corriente entra por el positivo y sale por el negativo; por tanto, el ánodo es el positivo y, el cátodo, el negativo.

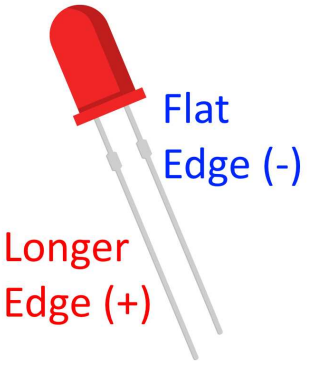

Figura 4: Conexión de un LED con la placa Raspberry Pi

En la Figura 5 vemos cómo quedaría la conexión del LED con la placa de Raspberry Pi. En este caso, hemos usado una resistencia, si bien se ha testeado durante largos periodos la conexión de un LED sin esta y no se han detectado anomalías de funcionamiento.

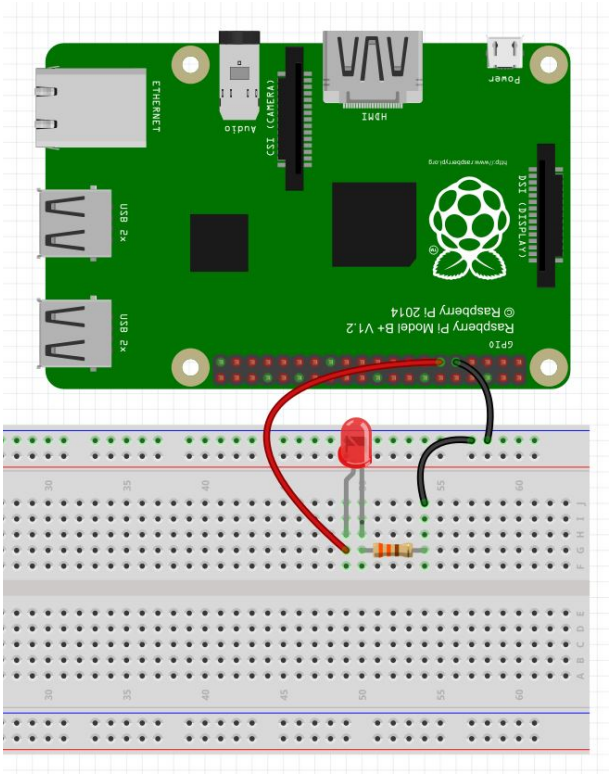

Figura 5: Conexión de un LED con la placa Raspberry Pi

Como se puede apreciar, estamos utilizando el pin físico 9 como toma de tierra (GND) y el pin físico 11 (o GPIO 17) como el pin de manejo (y alimentación $3{,}3V$). Recuerda el diagrama de puertos GPIO que ya vimos en la Figura 1, y que más cómodamente puedes visualizar en cualquier momento en el Terminal de Raspberry Pi OS mediante el siguiente comando:

```
pinout
```

6. Manejo básico del LED mediante el Terminal

Podemos controlar los puertos GPIO sencillamente mediante comandos en el Terminal, concretamente, usando los ficheros de sistema existentes en la carpeta `/sys/class/gpio/` para tal efecto.

Lo primero que hemos de hacer es inicializar el fichero que manejará su correspondiente puerto GPIO. Por ejemplo, para manejar el pin físico 11, creamos el fichero correspondiente al GPIO 17 mediante el siguiente comando:

```
echo 17 > /sys/class/gpio/export
```

A continuación, especificamos si lo vamos a usar de salida o de entrada. En este caso, especificamos que va a ser de salida (out):

```
echo out > /sys/class/gpio/gpio17/direction
```

Y ya podemos usar el pin GPIO. Para enviar un pulso eléctrico y que, de este modo, se encienda el LED, lanzamos el siguiente comando:

```
echo 1 > /sys/class/gpio/gpio17/value
```

De forma similar, si queremos poner su estado a bajo o LOW, enviamos un 0:

```
echo 0 > /sys/class/gpio/gpio17/value
```

Y, por último, para resetear el comportamiento que hemos otorgado a este pin, borramos el fichero creado a tal efecto mediante el siguiente comando:

```
echo 17 > /sys/class/gpio/unexport
```

7. Programación del LED mediante PWM

Lo primero que debemos hacer siempre para poder usar los puertos GPIO de la placa Raspberry es importar la biblioteca RPi:

```
import RPi.GPIO as GPIO
```

A continuación lo que debemos hacer para evitar confusiones (muy importante) es indicar qué esquema de numeración de pin vamos a usar: numeración de pin físico (BOARD) o numeración según *Broadcom SOC channel* (BCM). En nuestro caso, por mera preferencia memorística, vamos a usar la primera opción:

```
GPIO.setmode (GPIO.BOARD)
```

En otro caso, deberíamos reemplazar BOARD por BCM en el comando anterior. El siguiente paso es indicarle que el pin que vamos a usar, el 11, va a ser de salida, pues no vamos a percibir nada del LED, sólo emitir. Para ello escribimos la siguiente instrucción:

```
GPIO.setup (11, GPIO.OUT)
```

En este punto ya podríamos escribir la señal en dicho pin en estado *Alto* o *Bajo*, pero nuestro objetivo es usar PWM para controlar el brillo del LED a nuestro antojo, para lo que necesitamos hacer una cosa más: crear un objeto PWM con el pin a usar y la frecuencia de trabajo como parámetros:

```
pwm = GPIO.PWM (11,100)
```

La frecuencia de 100 Hz es muy cómoda para trabajar con ella, pues el periodo resultante es de 10 ms, un número bastante redondo para hacer cálculos de forma sencilla.

El siguiente paso es establecer el ciclo de trabajo o *DutyCyle*. Este, por definición, es el porcentaje del período en que la señal está en estado *Alto*. Así, por ejemplo, si queremos aproximar una señal de 1.6 V, que es la mitad de los 3.3 V que aporta el pin de la Raspberry, necesitamos establecer un ciclo de trabajo del 50 %. Esto se hace mediante el siguiente comando:

```
pwm.start (50)
```

Ya con este comando se debería encender el LED con la mitad de brillo. Cambiando el ciclo de trabajo podemos cambiar este; por ejemplo, si queremos que la luz sea muy tenue, estableceremos un ciclo de trabajo del 1 % con el siguiente comando:

```
pwm.ChangeDutyCycle (1)
```

Por contra, si quiero que luzca en todo su esplendor, al brillo máximo, debería establecer un ciclo de trabajo del 100 %: `pwm.ChangeDutyCycle (100)`

También se puede cambiar la frecuencia de la señal PWM con `ChangeFrequency`. Así, por ejemplo, si quiero establecer una frecuencia de 1000 Hz, lo haré mediante la instrucción:

```
pwm.ChangeFrequency (1000)
```

Aunque en este caso no apreciaré un cambio en el brillo de la luz, sino simplemente va más rápido, pero esto no afecta el tiempo fraccional en que la señal está encendida y apagada, por lo tanto, el brillo del LED no cambia.

Para que estas instrucciones comandadas tengan efecto, hemos de darles tiempo para que se ejecuten. Esto lo conseguimos añadiendo cualquier instrucción que fuerce a parar la ejecución del programa; por ejemplo, la siguiente:

```
input("Ejecutando hasta que pulse una tecla...")
```

Por último, antes de acabar la ejecución del programa, hemos de desactivar el PWM con:

```
pwm.stop ()
```

Y también hacemos lo correspondiente con los puertos GPIO:

```
GPIO.cleanup ()
```

Práctica 2

Introducción a la electrónica GPIO con el LED RGB

Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Introducción

Una de las primeras prácticas fue sobre cómo manejar un LED (normal) mediante la técnica de PWM o modulación del ancho de pulso. Se puede considerar esta como el *Hola mundo* de la electrónica. Es fácil de instalar, se recuerdan conceptos de electricidad que ya tenías aprendido de antes y, además, el resultado es vistoso.

En esta ocasión, vamos a usar un LED más avanzado y, por tanto, más complejo de instalar y programar: el LED RGB. Evidentemente, las siglas RGB ya te resultarán familiares; son las siglas de *Red*, *Green*, *Blue*. Este *macro*LED contiene en una sola pieza esos tres LEDs[1] de colores. Además, y como ya sabrás, estos son los colores primarios de la luz, lo que significa que —modulándolos convenientemente— podrás conseguir cualquier color de luz.

Para manejar apropiadamente este LED, vamos a aprender en detalle cómo funcionan los puertos o pines GPIO, ya que son clave para conectar los sensores y actuadores que veremos durante el curso.

2. Los pines GPIO

Como ya hemos visto en la Práctica 1, además del interfaz para conexión de dispositivos estándar (HDMI, Ethernet, USB, Jack), la Raspberry Pi ofrece una serie de pines denominados GPIO (*General Purpose Input Output*). Estos pines se emplean comúnmente para conectar dispositivos electrónicos, como sensores y actuadores.

Ya hemos aprendido que estos pines son digitales (si queremos usarlos como analógicos hemos de acoplar un DAC, *Digital to Analog Converter*), y que pueden ser configurados de entrada o de salida. Al configurarlos de salida, como hicimos en la Práctica 1, pueden ser activados o desactivados por código; esto es, pueden ponerse a 1 (*HIGH*, o nivel alto de voltaje) o a 0 (*LOW*, o nivel bajo de voltaje). Por contra, si los configuramos de entrada, pueden leer datos binarios; es decir, unos (hay señal de voltaje) o ceros (no hay voltaje, o este es cercano a cero).

Es muy importante tener en cuenta que los puertos GPIO funcionan sobre niveles lógicos de $3{,}3V$: $0V$ significa el 0 lógico o *LOW* y $3{,}3V$ es la tensión equivalente al 1 lógico o *HIGH*. Por ello, hay que tener cuidado de no conectar nunca los pines de alimentación de $3{,}3V$ y $5V$ directamente entre sí, o un pin de $5V$

[1] Realmente, deberíamos hablar de *ledes* (igual que el plural de *red* es *redes*), aunque coloquialmente está más extendido el uso —incorrecto— de *LEDs*.

directamente a un pin GPIO.

Respecto al uso de un pin GPIO como pin de entrada, esta puede ser configurada de dos formas: como evento/interrupción para que, al producirse esta, genere alguna acción sobre el sistema; o, lo más común y el modo que emplearemos nosotros, que la acción sea realizada cuando uno o más pines sean activados por algún sensor.

Por último, y según las especificaciones del bus de puertos GPIO de la Raspberry Pi 3B+, los pines de $3{,}3V$ son capaces de proporcionar un máximo de $50mA$ al mismo tiempo. Aunque no existe un limitador, nos sirve para hacernos una idea de qué números estamos manejando. Dicho de otro modo, no ocurrirá nada porque nos pasemos ligeramente de ese valor en un momento dado. Por ejemplo, si pensamos en el caso de mayor demanda de intensidad, que sería cuando están encendidos los tres LEDs de color al mismo tiempo, la placa estará perfectamente protegida si nos mantenemos cercanos a esos $50mA$. Podemos establecer la intensidad máxima de cada LED del siguiente modo: $50mA/3 \approx 17mA \implies I_{max_R} = I_{max_G} = I_{max_B} = 17mA$.

3. El LED RGB

Antes de proceder con la instalación de este LED tan particular hemos de echar números. Veremos que cada color requiere de un valor diferente de tensión e intensidad: el rojo es el que menos tensión requiere, mientras que el azul es el que más tensión necesita.

Figura 1: LED RGB

Por otro lado, como vemos, este LED tiene cuatro pines en lugar de dos, como puedes comprobar en la Figura 1. Esto es debido a que, como ya hemos mencionado previamente, un LED RGB incluye en realidad un circuito de tres LEDs diferentes. Comúnmente, el pin más largo (n.º 2) es el cátodo, mientras que los otros tres corresponden a los tres colores: rojo (1), verde (3) y azul (4).

Para salir de dudas y, como siempre, recuerda consultar la hoja de especificaciones de tu modelo concreto. Se adjunta la hoja de especificaciones de un LED RGB común. No obstante, el LED RGB que se incluye en el kit tiene la polaridad cambiada respecto a la norma común. Hay que tener esto en cuenta para la conexión del circuito (Sección 3.1).

3.1. Esquema de conexión

Para conectar el LED RGB y ajustarnos a las especificaciones de los pines GPIO de la placa, vamos a necesitar resistencias. El uso de estas es fundamental para conectar estos y otros dispositivos delicados a la placa, para proteger no solo a estos, sino —y lo que es más importante— a la placa. Ten en cuenta que, típicamente, estos LEDs soportan hasta $30mA$ de intensidad, pero nosotros, por seguridad, los vamos

a mantener por debajo de $17mA$, como habíamos calculado.

Las resistencias que tendremos que usar dependerán de las especificaciones concretas del LED RGB. En cualquier caso, obviamente necesitarás tres resistencias; una por cada pin de color. Para saber leer el código de colores de una resistencia, recuerda tener en cuenta lo siguiente:

1. Nosotros usaremos resistencias de cuatro bandas, aunque también las hay de cinco.
2. Las resistencias son simétricas; para leerla correctamente fíjate en las dos bandas situadas en los extremos, verás que hay una que está más pegada al extremo que la otra; pues bien, esa debe quedar a la izquierda.
3. Por si no estás totalmente seguro con el paso anterior, otra forma para asegurarnos de que estamos leyendo la resistencia correctamente es observar que existe un extremo en el que existe mayor separación entre la última banda y la penúltima; pues bien, este extremo ha de quedar a la derecha.
4. Ya teniendo bien situada la resistencia, observamos la primera y segunda banda. Estas nos dan el valor. Cada una va de 0 a 9, obteniendo por tanto un valor total comprendido entre 00 y 99.
5. La tercera banda es el multiplicador; esto es, por cuánto hemos de multiplicar el valor obtenido en el anterior paso. Así, por ejemplo, si el valor obtenido de las bandas 1 y 2 es 10 y el multiplicador es 100, el valor de la resistencia será de 1000Ω.
6. La cuarta banda es la tolerancia; esto es, el margen de error entre el valor dado según el código de colores y el valor real de la resistencia. Un margen de error o tolerancia habitual es el 5 %, representado por el color dorado.

También existen calculadoras *on-line*[2] que nos vierten el valor de una resistencia en función de sus bandas de colores.

O también puedes hacer uso de alguna de las calculadoras *on-line* de resistencia específicamente diseñadas para cuando usemos LEDs[3][4], que seguro te resultarán de mucha ayuda. Estas emplean la famosa *Ley de Ohm*.

Para usar una de estas herramientas, implemente introduce la tensión de la fuente, $V_S = 3{,}3V$, el voltaje que queremos que llegue al LED, e.g. $V_{LED} = 1{,}5V$ (lógicamente, $V_{LED} <= V_S$), y la intensidad (I_{LED}) que deseemos aplicar a este (recuerda, $I_{LED} < 17mA$). Con estos datos, obtendremos la resistencia (R) que hemos de colocar, así como la potencia que tendrá el LED (P). Otra forma de trabajar, y siguiendo con la Ley de Ohm, según las necesidades, es *despejar* la V_{LED}; esto es, indicar la resistencia que vamos a aplicar, y en función de esta nos indicará qué tensión V_{LED} le llegará al LED.

Sea como sea, junto con las tres resistencias que finalmente insertemos y el cableado, resulta de especial importancia el uso de la *protoboard*. Con ayuda de esta, el esquema de conexión de un LED RGB común quedaría como vemos en la Figura 2.

Nótese que el esquema mostrado en la Figura 2 es el esquema de conexión general de un LED RGB común, en el que, como vemos, necesitamos conectar el cátodo común del LED RGB a un pin de toma de tierra (*ground*), y cada uno de los tres ánodos a un pin programable diferente, usando en total los pines: 9 (GND), 11, 13 y 15.

No obstante, y como ya se ha mencionado previamente, el LED RGB que se incluye en el kit tiene la polaridad cambiada. Es por ello que, siguiendo el esquema de conexión de la Figura 2, no funcionará. Simplemente, habría que conectar los cables al revés de lo mostrado en la Figura 2; esto es, considerando la patilla larga como ánodo. Considérese esto una excepción, pues la mayoría de estos LEDs RGB suelen estar configurados como se ha explicado de forma general en los párrafos anteriores.

[2] `https://www.digikey.es/es/resources/conversion-calculators/conversion-calculator-resistor-color-code`
[3] `https://ohmslawcalculator.com/led-resistor-calculator`
[4] `https://ledcalculator.net`

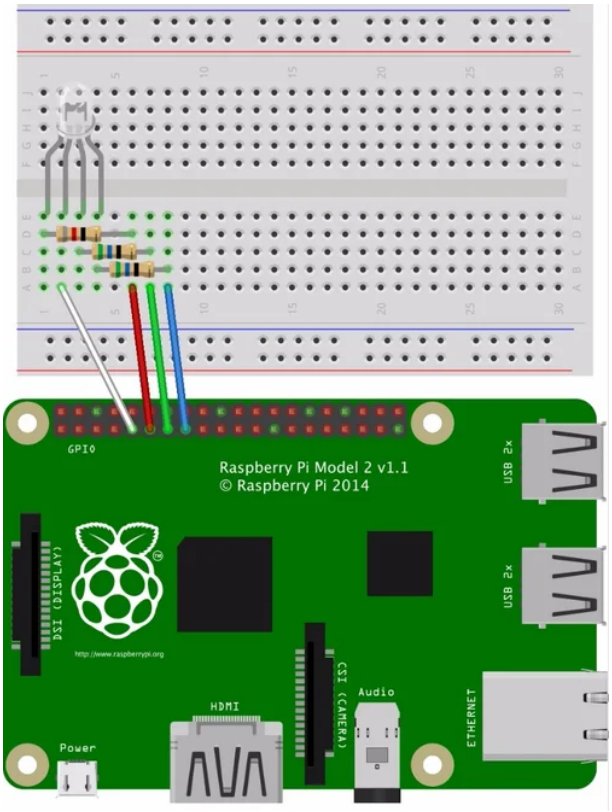

Figura 2: Esquema de conexión de un LED RGB común con protoboard

4. Programación del LED RGB

Una vez tenemos lo anterior, ya la programación es sencilla; basta con crearnos tres variables para guardar los pines correspondientes a cada uno de los tres colores y operar con ellos a nuestro antojo, como ya hicimos para el LED básico.

En el código que se facilita (`ledRGB.py`) hemos usado el modo `GPIO.BOARD`, por lo que hemos creado las siguientes tres variables:

```
rojoPin = 11
verdePin = 15
azulPin = 13
```

Se facilitan dos funciones para encender y apagar un pin determinado, así como una función concreta para encender el color rojo. Faltan, entre otras cosas, las instrucciones que ya hemos visto de *limpieza* de puertos GPIO (`GPIO.cleanup()`) de antes de finalizar la ejecución del programa.

5. Ejercicios

Ejercicio 1. Uso de colores

1. En primer lugar, modifica convenientemente el programa para que pueda gestionar el encendido y apagado de los tres colores primarios.

2. En segundo lugar, añade la funcionalidad necesaria para gestionar el encendido y apagado de, al menos, otros cinco colores, p. ej.: magenta, amarillo, cyan, blanco, etc.

Ejercicio 2. Interfaz de usuario

Partiendo de la funcionalidad implementada para el Ejercicio 1, añade nueva funcionalidad para que el programa pida por teclado al usuario las órdenes de qué quiere hacer. Por ejemplo:

```
encender rojo
apagar rojo
encender blanco
encender verde
salir
```

Práctica 3

Uso de interrupciones para leer valores de un sensor

Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Introducción

Las interrupciones suponen un mecanismo muy empleado en programación y un tema muy interesante en cómo las manejan los sistemas operativos. Por definición, una interrupción es un evento que ocurre de forma asíncrona (en cualquier momento) respecto a la ejecución del programa. A continuación, se expone un breve repaso al funcionamiento de un programa cuando este se ejecuta.

Durante la ejecución de un programa se van sucediendo saltos entre las distintas líneas de código. Por ejemplo, cuando se llama a una función, se produce un salto de ejecución de la línea actual del programa a dicha función. La dirección actual se almacena en la pila de ejecución del programa; dicho más apropiadamente, del proceso asociado al programa, aunque esto es otra historia sobre la relación programa-proceso que establece un sistema operativo. En UNIX, un proceso corresponde a un programa en ejecución. Almacenada esa dirección en la pila del proceso, se salta a la función y se ejecuta la rutina de instrucciones que engloba esta. Una vez termina de ejecutarse la función (puede haber más llamadas anidadas a funciones), se salta a la dirección que indique el puntero de pila, que será la última *apilada* y que corresponderá con la dirección de la instrucción que llamó a la función. El procesador seguirá ejecutando el proceso a partir de esta dirección; esto es, el programa sigue su curso...

1.1. Interrupciones o eventos

Hasta aquí todo normal, el programa sigue estrictamente las instrucciones según se haya diseñado este. Pero esta armonía del programa se verá interrumpida si introducimos el manejo de eventos en este. La principal novedad que se presenta al tratar eventos es que estos se dan de forma asíncrona; esto es, mientras que el programa está haciendo *sus cosas*, ocurre un evento, y hay que tratarlo. Cuando el SO, según su planificación, decida que es momento de tratar el evento, entonces para la ejecución *normal* del programa para ejecutar la rutina de tratamiento de dicho evento, también denominado *callback*. Sintácticamente, un *callback* es una función especial, pues es una función que se pasa a otra como argumento. Ejemplo de *callback*:

```
def botonPresionado ():
    print("El boton ha sido presionado")

GPIO.add_event_detect(BUTTON_GPIO, GPIO.FALLING,
  callback=botonPresionado, bouncetime=100)
```

1.2. Programación concurrente

En este punto adquiere especial relevancia la programación concurrente; esto es, la gestión de varios hilos de ejecución en un programa. Mientras que en programación secuencial solo tratamos con un hilo de ejecución, el hilo principal o *main*, y las instrucciones se van sucediendo una tras otra, en programación concurrente lanzamos conscientemente diferentes hilos, o *threads*, que nacen siempre a partir del hilo principal el programa; se produce lo que se denomina un *fork*, cuyo nombre, tenedor, se debe a su forma: el mango sería el hilo principal y luego se bifurca en varios hilos...

Si juntamos el tratamiento de interrupciones con la programación concurrente, tendremos programas muy eficientemente diseñados. Una buena práctica es, por ejemplo, asociar un hilo diferente a cada evento, para que el programa se pueda ejecutar a la vez que se puedan estar tratando los eventos. En realidad, esto es posible si el ordenador dispone de varios procesadores o núcleos, algo que en la actualidad es casi imposible no encontrar. Concretamente, en la Raspberry Pi, se pasó de uno a cuatro núcleos, siempre de la compañía *Broadcom*, a partir de su modelo 2B (2014).

2. Comportamiento de un pin GPIO de entrada

Hasta ahora hemos trabajado con los GPIO en su configuración de salida (OUT), para manejar convenientemente un LED. Es una configuración sencilla, porque somos nosotros los que variamos a nuestro antojo la señal de salida del pin. Pero al leer de un pin GPIO hemos de tener en cuenta cómo es su señal de entrada, y es que esta oscilará entre 0 y 1 si no está conectado a un voltaje, por ejemplo cuando leemos de un sensor; típicamente, este dará una señal cuando *sense* algo. Este hecho, como cualquier evento de los que hemos hablado en la sección anterior, se produce de forma asíncrona, en cualquier momento. Dicho de otro modo, si no leemos en el momento oportuno, no podremos distinguir la señal dada por el sensor de la propia señal oscilante entre 0-1 del pin; nunca podremos saber cuándo ha ocurrido el evento externo que se está sensando.

La solución a lo anterior es fijar un valor al pin y, de este modo, podremos detectar cuándo ocurre un cambio de valor, por ejemplo cuando el sensor envía señal porque se ha producido un evento. Para fijar un determinado valor a un pin GPIO, existen las denominadas resistencias *pull-up* y resistencias *pull-down*. Estas son unas resistencias específicas de la librería `rPi.GPIO` que suministran, vía software, un voltaje fijo para que el pin GPIO tenga un valor estático hasta que sea anulado por una señal más fuerte.

Se establece una resistencia *pull-down* (en 0) para un determinado pin GPIO cuando se espera que la señal conectada al mismo lo eleve a 1 cuando ocurra un evento. Y viceversa, se debe establecer un *pull-up* (en 1) si se espera que la señal conectada lo baje a 0 cuando ocurra el fenómeno que está siendo sensado.

3. Ejercicios prácticos de manejo de eventos

A continuación veremos diferentes ejemplos prácticos sobre cómo manejar eventos. En todos ellos usaremos un botón o pulsador como sensor o dispositivo de entrada, y el objetivo será siempre saber cuándo se ha presionado el pulsador. El esquema de conexión seguido para estos ejemplos es el que aparece en la Figura 1.

Nótese que el cable rojo está conectado al pin de lectura GPIO 16 (modo BCM), y el cable negro al sexto pin (GND-BOARD o toma de tierra). Por otro lado, téngase en cuenta que el pulsador ha de colocarse en la zona central de la protoboard, donde —recordemos— el circuito está discontinuado; es el pulsador el que une (o no) esas filas para precisamente cerrar el circuito (o no).

3.1. Método 0. Pidiendo el valor continuamente

En este primer ejemplo, cuyo código tenemos en `sinInterrupciones.py` (y cuyo *snippet* vemos a continuación), comprobamos continuamente el estado del pin de lectura del LED. Esto, como ya hemos adelantado, es poco eficiente. Además, y lo que es peor, siguiendo esta dinámica de *comprobación exhaustiva* se puede

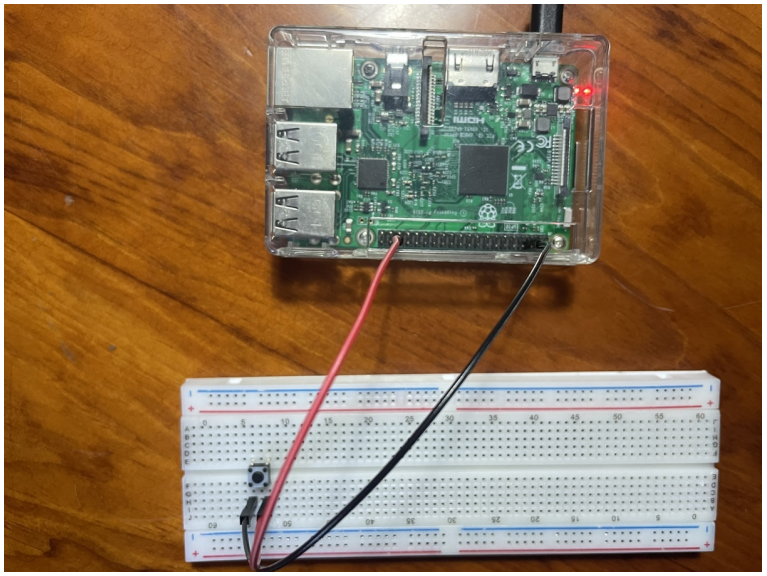

Figura 1: Esquema de conexión de pulsador empleado para estos ejemplos

llegar a saturar el procesador.

```
# Activamos resistencia pull_up_down en modo HIGH, esto es:
# - HIGH: estado por defecto del GPIO (no se ha pulsado).
# - LOW: estado del GPIO cuando se ha pulsado el boton.
GPIO.setup(pulsadorGPIO, GPIO.IN, pull_up_down=GPIO.PUD_UP)

pulsado = False

while True:
    if not GPIO.input(pulsadorGPIO): # la lectura siempre da 1 (HIGH/True) excepto al pulsar,
                                     # y solo en ese instante, que da 0 (LOW/False)
        if not pulsado: # con esta variable evitamos considerar mas de una vez una pulsacion;
                        # puede que se lea +1 vez el estado del pin antes de cambiar su estado
                        # Este fenomeno se conoce como rebote (o bounce). Algunas funciones
                        # permiten establecer un parametro denominado "bouncetime"
            print("El boton se ha pulsado")
            pulsado = True
    else:
        pulsado = False

    time.sleep(0.1) # si pulsamos rapida/ veremos que algunas se escapan...
```

Como observamos en el código, además de establecer el pin en modo lectura, fijamos un valor al mismo —como ya hemos comentado previamente—. Establecemos *HIGH* como valor por defecto (`pull_up_down=GPIO.PUD_UP`), y así cuando se pulse el botón, el estado cambiará a *LOW*. A continuación, comprobaremos continuamente el estado del pin en un bucle infinito; cuando se pulse el botón, se muestra un mensaje por pantalla.

Para no saturar al procesador, establecemos una determinada frecuencia con la instrucción `time.sleep(segundos)`. A menor frecuencia, mayor será el alivio para el procesador, pero más comprobaciones nos perdemos. Por ejemplo, con un *sleep* de $0{,}1s$, obtenemos una frecuencia de comprobación de $f = 1/t \implies f = 1/0{,}1 = 10Hz$; dicho de otro modo, todas las pulsaciones que se hagan dentro de ese intervalo contarán como una. Así pues, hemos de llegar a un valor que equilibre ambos frentes. Además, otro factor negativo de emplear este mecanismo para tratar eventos es que no sabremos nunca cuándo exactamente se ha pulsado el botón; puede ser cualquier momento del intervalo establecido.

3.2. Método 1. Interrupciones con wait_for_edge

En Raspberry Pi no existen las interrupciones hardware (como sí las hay, por ejemplo, en Arduino), pero sí se pueden simular vía software gracias a la librería de manejo de GPIO (`RPi.GPIO`). Sea de una forma o de otra, las interrupciones se activan cuando el estado del pin que estamos leyendo cambia de valor. Concretamente, se pueden activar de dos modos: cuando el pulso cambia de 0 a 1 (*rising*) o cuando cambia de 1 a 0 (*falling*). En nuestro contexto, *rising* sería sería el instante en que el pulsador va hacia arriba (se deja de pulsar) y *falling* el instante en que va hacia abajo (es pulsado).

```
while True:
    GPIO.wait_for_edge(pulsadorGPIO, GPIO.RISING)
    print("El boton se ha pulsado")
```

En este *snippet* del codigo `interrupcionEdge.py` también facilitado, vemos la aplicación de la función `wait_for_edge`. Se configure en modo *rising* o *falling*, recibiremos una notificación cuando el pin cambia de estado. Ya no tenemos que estar comprobando una y otra vez cuál es el estado del pin. También se puede configurar para recibir la notificación en cualquier caso, estableciendo `BOTH` en su lugar.

Al ejecutar este código te percatarás de dos cosas: la primera, es que el comportamiento del programa no es del todo correcto, pues hay ocasiones en que se muestra más de una vez el mensaje de *pulsado* cuando en realidad solo se ha pulsado una vez el botón; la segunda, es que no hay forma de parar el programa de la forma habitual, pulsando la combinación *CTRL+C*.

La explicación a estas dos cuestiones es sencilla. La razón de la primera es que no hemos implementado ningún mecanismo antirebote, como sí hicimos en el primer ejercicio, y así vemos los efectos. La razón de la segunda es que, en esta ocasión, como podemos observar, no damos un respiro al procesador en el bucle infinito, y el resultado es un procesador dedicado exclusivamente a dar vueltas, es por ello que ignora la la combinación *CTRL+C*.

La solución a ambos problemas también es sencilla. Para el primer problema, hemos de implementar un sistema antirebote, similar al primer ejercicio. Para el segundo problema, la solución es establecer una frecuencia de iteración en el bucle, como también hacíamos en el primer ejercicio. De momento, para salir del programa, en lugar de emplear la combinación *CTRL+C*, que se trata de una interrupción de teclado (y por ello está siendo ignorada, porque el procesador no da para más), emplea la combinación *CTRL+Z*, que lo que hace es detener, que no terminar, el proceso `python3 interrupcionEdge.py`. Es por ello que este proceso sigue ejecutándose en *background*. Puedes comprobar que está entre los procesos en ejecución con el comando `ps ax`. También puedes —y debes— terminarlo definitivamente mediante la orden `killall -9 python3` (o bien con la orden `kill -9 PID`, PID es el ID del proceso).

Otra solución más digna para el segundo problema es emplear un *thread* aparte dedicado a tal evento, pero esto es otro cantar... Veamos la solución plausible, sin necesidad de más hilos de ejecución en el siguiente apartado.

3.3. Método 2. Interrupciones con add_event_detect

La principal y radical diferencia entre este y el anterior método es el uso de *callbacks*. Esta será la función que se invoque cuando ocurra el evento; esto es, cuando se haya pulsado el botón. En realidad, un *callback* tiene asociado un hilo de ejecución aparte, pero no lo hemos de implementar nosotros, por eso el código resulta más claro, más sencillo y —sobre todo, si no hemos trabajado con *threads*— más entendible.

```
GPIO.add_event_detect(pulsadorGPIO, GPIO.FALLING,
  callback=callbackBotonPulsado, bouncetime=500) # expresado en ms.

signal.signal(signal.SIGINT, callbackSalir) # callback para CTRL+C
signal.pause() # esperamos por hilo/callback CTRL+C antes de acabar
```

Como vemos en este *snippet* perteneciente al código facilitado en `interrupcionEvent.py`, la función `add_event_detect` admite ese tercer parámetro de *callback*, además de un cuarto donde expresar el tiempo de rebote (*bouncetime*) que se ajuste a nuestras necesidades. El valor establecido ($500ms$) ha demostrado funcionar convenientemente. Con estos dos parámetros, estas dos mejoras respecto al mecanismo anterior, ya tenemos cazados de una forma muy sencilla los dos problemas que se planteaban en el anterior ejemplo.

Por otro lado, estamos usando la librería signal, que nos permite tratar con interrupciones de teclado, como es el caso de la combinación $CTRL+C$ para terminar la ejecución. En este caso hemos de usar este mecanismo porque, de lo contrario, al tratar con varios hilos de ejecución (aunque no los veamos), si pulsamos $CTRL+C$ solo afectaría al hilo principal, que en este caso de hecho no está haciendo nada después de invocar el *callback*. Y ya que tratamos mediante otro *callback* la terminación del programa (función `callbackSalir`), aprovechamos a limpiar los recursos empleados de los puertos GPIO en dicha función mediante la instrucción `GPIO.cleanup()`.

Ejercicios

Ejercicio 1

Mejora la implementación del código facilitado en `interrupcionEdge.py` para que no ocurran los problemas descritos del mismo. Al fichero resultante denomínalo `interrupcionEdgeBueno.py`

Ejercicio 2

Rediseña el esquema de conexión, así como la implementación de los tres códigos facilitados para dar soporte a dos botones que a su vez operan sobre dos LEDs de forma independiente. Así, al pulsar uno de los botones, se encenderá un LED rojo (y se apagará al soltar dicho botón); y, al pulsar el otro botón, lo mismo pero con el otro LED, que será verde.

Recuerda que deberás ofrecer esa funcionalidad bajo las tres implementaciones que se han expuesto aquí. Los códigos resultantes estarán en ficheros con estas denominaciones:

1. `sinInterrupcionesMejorado.py`
2. `interrupcionEdgeMejorado.py`
3. `interrupcionEventMejorado.py`

Práctica 4

Construcción de un encoder incremental usando un sensor óptico

Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Introducción

Como ya hemos visto, un encoder es un sensor que convierte un movimiento rotacional o lineal en una señal digital equivalente. También se denominan tacómetros o codificadores de posición.

Los encoders son los sensores más usados fundamentalmente para la medición de sistemas rotacionales. Actualmente la mayoría son ópticos o magnéticos, frente a los que había hace años, que eran mecánicos. En la siguiente Figura 1 podemos ver estos sensores acoplados a los ratones mecánicos antiguos y a los ratones usados en la actualidad.

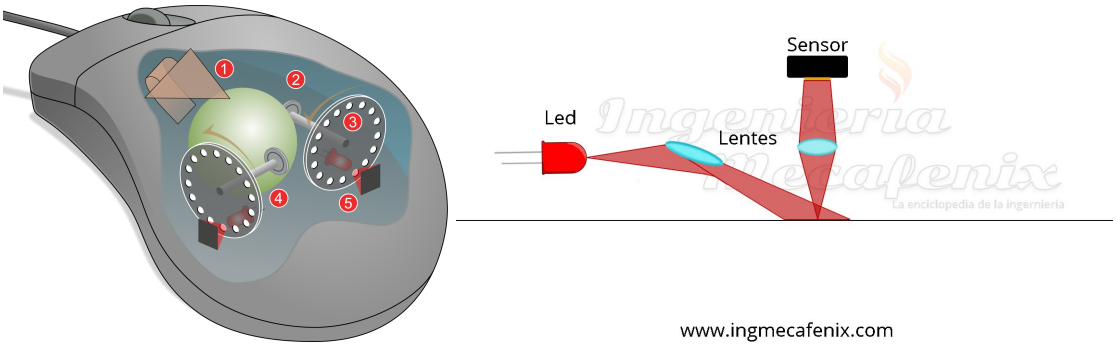

Figura 1: Ratones con encoders mecánico y óptico. Imágenes extraídas de Wikipedia y www.ingmecafenix.com

2. El encoder óptico

En esta práctica nos centraremos en el encoder de tipo óptico. Este tipo de encoders está compuesto por dos componentes optoelectrónicos (aquellos cuya tecnología combina la óptica y la electrónica): un emisor de luz y un receptor de luz. Normalmente, el emisor es un fotodiodo y el receptor es un fototransistor.

El mecanismo es sencillo; cuando el disco gira, se genera una señal alternante cuya frecuencia es directamente proporcional a la velocidad de rotación del eje. Calcular la posición en la que se encuentra una vez haya girado es algo delicado, y es que, saber el sentido de giro, para empezar, no es algo sencillo. La fórmula general que se aplica para saber la resolución de un encoder es la siguiente:

$$res = \frac{\pi D}{2a_r} \tag{1}$$

donde:

res : resolución del encoder
D : diámetro del disco
a_r : ancho de cada ranura

3. El encoder incremental

Una solución para conocer el sentido de giro de un encoder es el denominado modelo incremental (Figura 2). Este tipo de encoders se basa en emplear al menos dos pares emisor-receptor con un desfase de +1/4. Así, la señal estará desfasada 1/4 o 3/4 entre pares según el sentido de giro.

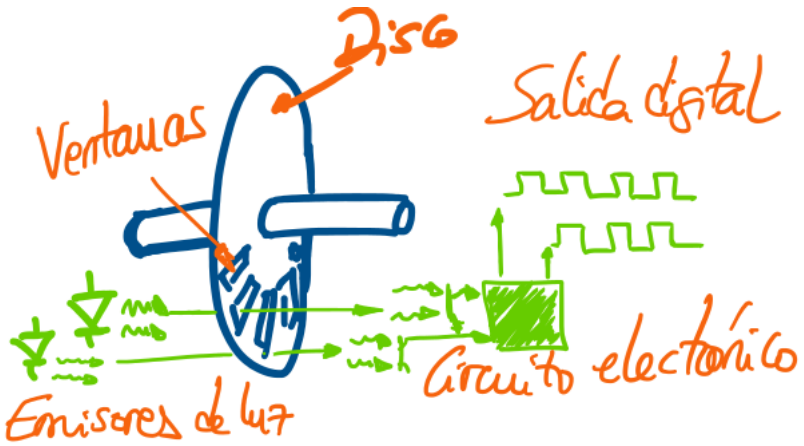

Figura 2: Mecanismo de un encoder incremental

Además de lo anterior, suelen incluir una muesca adicional para indicar una vuelta completa, lo que permite también conocer la posición inicial del encoder en caso de un fallo electrónico puntual. Por otro lado, esta muesca permite obtener el número de pulsos que se producen en una vuelta completa, y esto a su vez permite obtener la posición, sentido de giro y velocidad angular del encoder.

4. Descripción de la práctica

Los componentes que vamos a necesitar para realizar esta práctica son los siguientes:

1. Un optointerruptor ITR8102.
2. Una resistencia de 330Ω.
3. Una resistencia de $10k\Omega$.
4. Una resistencia de $20k\Omega$.
5. Disco con muescas casero.

Figura 3: Optointerruptor ITR8102. Imagen extraída del datasheet adjunto de Everlight

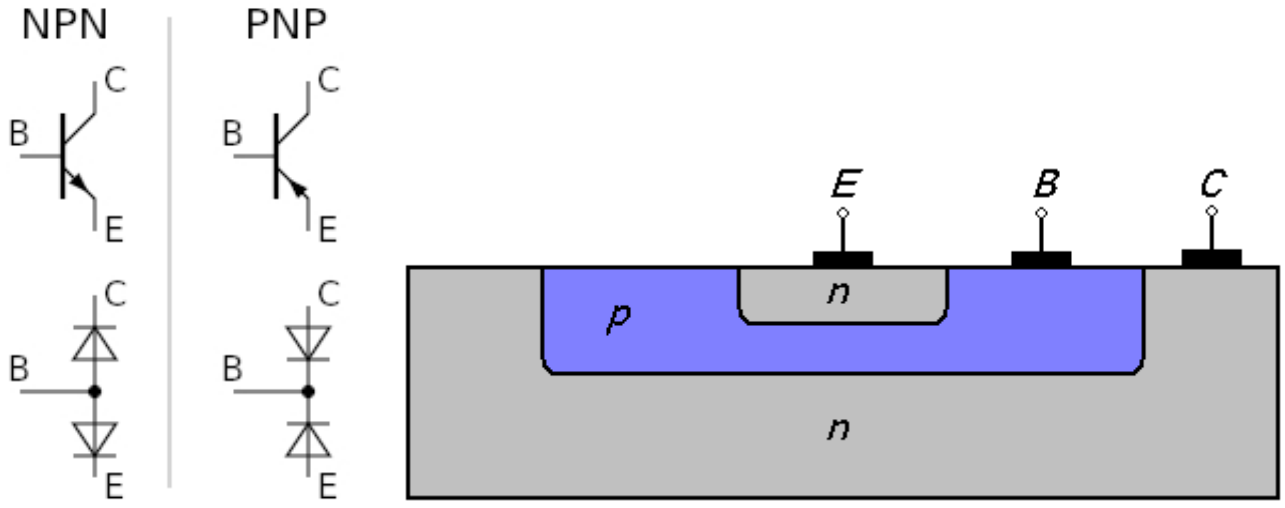

Figura 4: Transistores bipolares e interior de trans. NPN. Imágenes extraídas de Wikimedia Commons

4.1. Optointerruptor ITR8102

El modelo de optointerruptor que vamos a usar es el ITR8102 como el que podemos ver en la Figura 3.

Este modelo incorpora un LED infrarrojo y un fototransistor NPN. El fototransistor recibirá la luz infrarroja emitida por el LED, a menos que se interponga un objeto que impida el paso de la luz. Ya veremos en el Tema 4 más en profundidad los transistores y qué significa que el modelo sea NPN. No obstante, veamos algunos conceptos preliminares. Un transistor es un semiconductor que se controla según la I que se le aplique y del que se obtiene una I amplificada.

El semiconductor puede ser de dos tipos:

- Tipo P, que significa que tiene impurezas aceptoras o, dicho de otro modo, que generan una gran cantidad de portadores huecos (+).
- Tipo N, que posee impurezas donoras; esto es, que genera gran cantidad de portadores electrones (-).

Siguiendo con lo anterior, si el modelo es NPN o PNP significa que tiene dos uniones PN muy cercanas; es decir, que es un transistor bipolar (en inglés BJT (*Bipolar Junction Transistor*). En la parte izquierda de

la Figura 4 podemos ver el símbolo de ambos transistores bipolares; a la derecha, cómo es el interior de un transistor NPN, como el que vamos a usar nosotros.

En dicha figura, además, observamos los tres terminales de un transistor NPN, cuyas letras E, B y C se corresponden con Emisor (trabaja en directa, y hace que R disminuya), base y colector (trabaja en inversa, y hace que R aumente). Veamos más detalles:

- El emisor está fuertemente dopado; esto es, se comporta de forma similar a un metal, emite portadores de carga.
- Si la base es alimentada por una fuente de CC ocurre que $I_C = \beta I_E$, donde β o *Beta del transistor* se corresponde con el factor de amplificación o ganancia entre I_C e I_B.

4.2. Divisor de voltaje para el transistor

Según la hoja de especificaciones (adjunta), este modelo de optointerruptor tiene una señal de salida de $5V$, que será la señal a leer en un pin GPIO. Es por ello que necesitamos bajar ese voltaje a $3{,}3V$, que recordemos es la tensión con la que trabajan todos los pines GPIO.

Para rebajar esa tensión podríamos optar por diferentes opciones. Las más adecuadas y seguras serían acoplar: *(a)* un regulador de tensión o *(b)* un convertidor a nivel lógico (que baja de $5V$ a $3{,}3V$). Pero la forma más inmediata y barata es emplear un circuito divisor de tensión. Lo único que tenemos que tener en cuenta a la hora de aplicar este tipo de circuitos es estar seguros de que la tensión de salida (de entrada al GPIO) del dispositivo es efectivamente $5V$ y no más, pues eso se traduciría en más de $3{,}3V$ a la salida del circuito divisor.

En nuestro caso, dado que la tensión que alimenta al dispositivo sale del pin de la placa con $V_{CC} = 5V$, podemos estar seguros de que el dispositivo recibe una tensión de $5V$ estable y, por ende, su salida es estable a $5V$, con lo que no supone un problema el aplicar un circuito divisor de voltaje para rebajar esa tensión a $3{,}3V$ de forma estable.

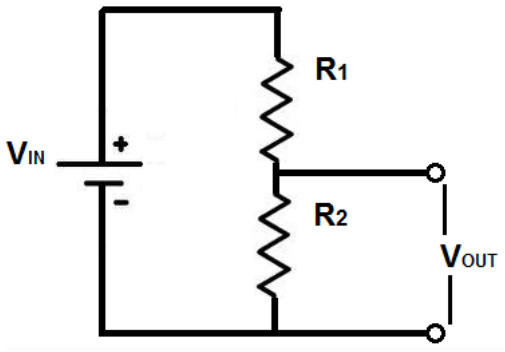

Figura 5: Circuito divisor de voltaje

Un circuito divisor como el mostrado en la Figura 5, con dos resistencias, R_1 y R_2, es el adecuado en este caso. Recordemos la fórmula general de un circuito divisor de voltaje:

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2} \tag{2}$$

Siguiendo la Fórmula 2, podemos obtener el voltaje de salida deseado de $3{,}3V$ partiendo de un voltaje de entrada de $5V$, si R_2 es el doble de R_1, como vemos en la Fórmula 3:

$$Si\ R_2 = 2 \cdot R_1 \implies V_{out} = \frac{2}{3}V_{in} = \frac{2}{3} \cdot 5V = 3{,}3V \tag{3}$$

Así, para nuestro objetivo, podemos emplear, por ejemplo, dos resistencias de valores $R_1 = 10k\Omega$ y $R_2 = 20k\Omega$.

Usa tu multímetro para asegurarte de que el esquema que has planteado es correcto antes de continuar.

5. Ejercicio

El objetivo de esta práctica es obtener la velocidad angular (en rpm) del encoder incremental que construiremos mediante un optointerruptor y un disco con muescas que fabricaremos de forma casera (un simple cartón con aspas es suficiente).

Ya conocemos las interrupciones software (las únicas que nos ofrece la placa Raspberry Pi). Deberás leer el tren de pulsos emitido por el sensor óptico y, haciendo uso de interrupciones, podrás sensar cuándo se produce un cambio en el pin al que conectemos el optointerruptor. Sabiendo esto, y llevando la cuenta de los cambios que se produzcan en un minuto, ya tendremos calculada la velocidad angular. También podemos hacer una estimación considerando un tiempo inferior (e.g. 10s).

En este caso no se facilita el esquema de conexión. Ya empezamos a jugar en la liga de mayores. Siguiendo el esquema de terminales del interruptor expuesto en la hoja de especificaciones adjunta, deberás ser capaz de identificar el lado del diodo emisor y el lado del transistor, así como de diseñar el circuito. Solo una pista: el LED infrarrojo ha de ir conectado a un pin de alimentación (¿5V?, ¿3V?) con una resistencia de 330Ω.

Práctica 5

Uso del sensor de ultrasonidos para medir distancias

Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Introducción

En el Tema 5 estudiaremos en profundidad los sensores de nivel y proximidad, entre los cuales destaca el sensor de ultrasonidos. En el mercado existen numerosos modelos de estos sensores, así como utilidades prácticas.

Es un sensor muy empleado en robótica, pues aporta información muy precisa sobre distancia a objetos y su programación es bastante sencilla.

Un sonido se califica como ultrasonido cuando la frecuencia es mayor que la frecuencia audible por el oído humano, que aproximadamente es de $20KHz$. El funcionamiento se basa en el ya comentado efecto Doppler, pues la onda emitida por el dispositivo es atenuada por objeto cuya distancia se pretende medir.

Pues bien, la medición de esa atenuación por parte del receptor es la clave de este tipo de sensores. El núcleo del sensor es un material piezoeléctrico. Recordemos que la piezoelectricidad se da cuando se produce una tensión de salida V_{out} debida a una presión ejercida sobre el material, o viceversa; esto es, debido a una excitación eléctrica, el material se deforma y emite una onda mecánica o, como sucede en este caso, emite una onda ultrasónica.

La onda se emite de forma cíclica a alta frecuencia y de corta duración. Esta onda viaja por el medio hasta que es reflejada en algún objeto y, entonces, regresa su *eco*. Mediante un circuito de acondicionamiento se determina el periodo de tiempo transcurrido entre la emisión de onda y la recepción de su eco. La fórmula empleada para ello se muestra en la Ecuación 1.

$$d = \frac{1}{2} v_s t \tag{1}$$

donde:

d : distancia del emisor-receptor al objeto [m]
v_s : velocidad del sonido
t : tiempo transcurrido [s]

2. Sensor de ultrasonidos HC-SR04

Dentro del kit que se ha facilitado en la asignatura, encontramos el sensor de ultrasonidos modelo HC-SR04, como el que vemos en la Figura 1, y cuya hoja de especificaciones, extraída de Sparkfun Electronics[1],

[1] `https://sparkfun.com`

se adjunta.

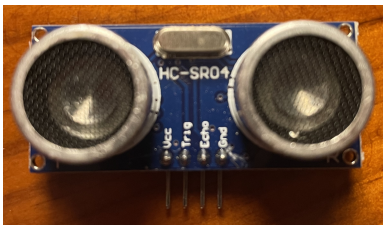

Figura 1: Sensor de ultrasonidos HC-SR04 incluido en el kit

Este modelo es un sensor de ultrasonidos cuya mayor utilidad es medir la distancia de este a cualquier objeto en el que rebota el ultrasonido en modo *reflexión directa*. Para ello, se contabiliza el tiempo que tarda la señal ultrasónica de $40kHz$ del transmisor en rebotar sobre una superficie y ser devuelta al receptor.

Teniendo en cuenta este tiempo y la velocidad a la que viaja el sonido en el aire, se puede obtener la distancia al objeto con una precisión bastante elevada.

2.1. Descripción de los pines del sensor

Este modelo cuenta con cuatro terminales o pines. Cada uno de ellos tiene una función que se detalla a continuación:

- VCC. Pin de alimentación del sensor (5V).
- TRIG. Pin de disparo (salida) que habilita la medición del sensor.
- ECHO. Señal de entrada al sistema.
- GND. Pin negativo de alimentación o toma de tierra.

2.2. Diagrama de tiempos

En la Figura 2, extraída de la hoja de especificaciones, podemos ver el funcionamiento de este modelo.

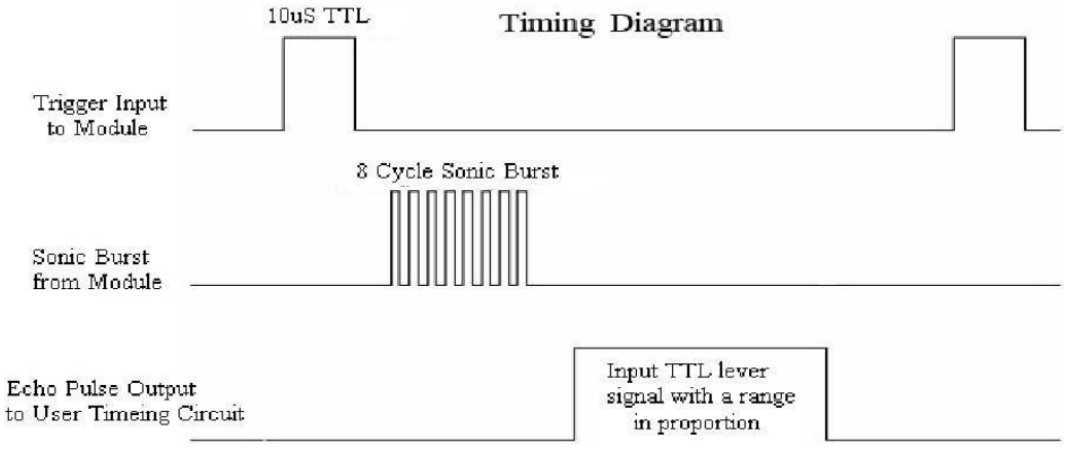

Figura 2: Diagrama de tiempos del modelo HC-SR04

Como vemos, para comenzar con la medición, es necesario aplicar un pulso, una señal en alto, a través del pin *Trig*, con una duración de —al menos— $10\mu s$. A continuación, el sensor envía una serie de 8 pulsos

de $40kHz$. Y, tras esto, pone el pin de *Echo* en alto, y permanecerá en ese estado hasta que se reciba el eco de los pulsos de $40kHz$. Es decir, este tiempo es el que se toma en cuenta para estimar la distancia a la que se encuentra el objeto.

2.3. Cálculo de la distancia al objeto

Como todos sabemos, la fórmula que relaciona la distancia con la velocidad es como se refleja en la Ecuación 2.

$$v = \frac{s}{t} \iff d = v \cdot t \tag{2}$$

Considerando que la velocidad del sonido en el aire es de $343\frac{m}{s}$, y sustituyendo en la Ecuación 2, nos queda la Ecuación 3.

$$d = v \cdot t \implies d = 343\frac{m}{s} \cdot t \tag{3}$$

En nuestro caso, como estamos usando el sensor en modo *reflexión directa*, la distancia d corresponde a la distancia recorrida por el tren de 8 pulsos desde que sale del módulo, hasta que vuelve al ser rebotada en el objeto. Dicho de otro modo, d es —por tanto— el doble de la distancia real d_r entre el sensor y el objeto. Así, esta d_r nos queda expresada apropiadamente en la Ecuación 4.

$$d = 343\frac{m}{s} \cdot t \implies d_r = 171{,}5\frac{m}{s} \cdot t \tag{4}$$

Por último, transformamos la Ecuación 4 a la Ecuación 5, donde hemos adaptado las unidades a las que estamos manejando en este problema en particular (cm y μs), para que nos resulte más cómodo trabajar con ella.

$$d_r = 171{,}5\frac{m}{s} \cdot t = 17150\frac{cm}{s} \cdot t(\mu s) \cdot 10^{-6}\frac{s}{\mu s} \tag{5}$$

3. Práctica

Una vez —y solo entonces— que hemos leído el *datasheet* y hemos interiorizado la teoría, podemos ponernos manos a la obra y montar el circuito, así como proceder a su programación.

3.1. Conexión del dispositivo

El esquema de conexión se puede observar en la Figura 3, cuyo funcionamiento se resume a continuación:

- VCC a pin de 5V directamente.
- TRIG a un pin GPIO.
- Divisor de voltaje para recibir ECHO (como ya hemos aprendido, e.g. $2{,}2K\Omega - 1K\Omega$, o también con $470\Omega - 220\Omega$:
  - ECHO a resistencia de $1K\Omega$.
  - La otra resistencia de $2{,}2K\Omega$ conecta en la misma línea de la otra resistencia hacia *Ground*.
  - Cable desde esa línea (entre las dos resistencias) hacia otro pin GPIO.
- GND a *Ground*.

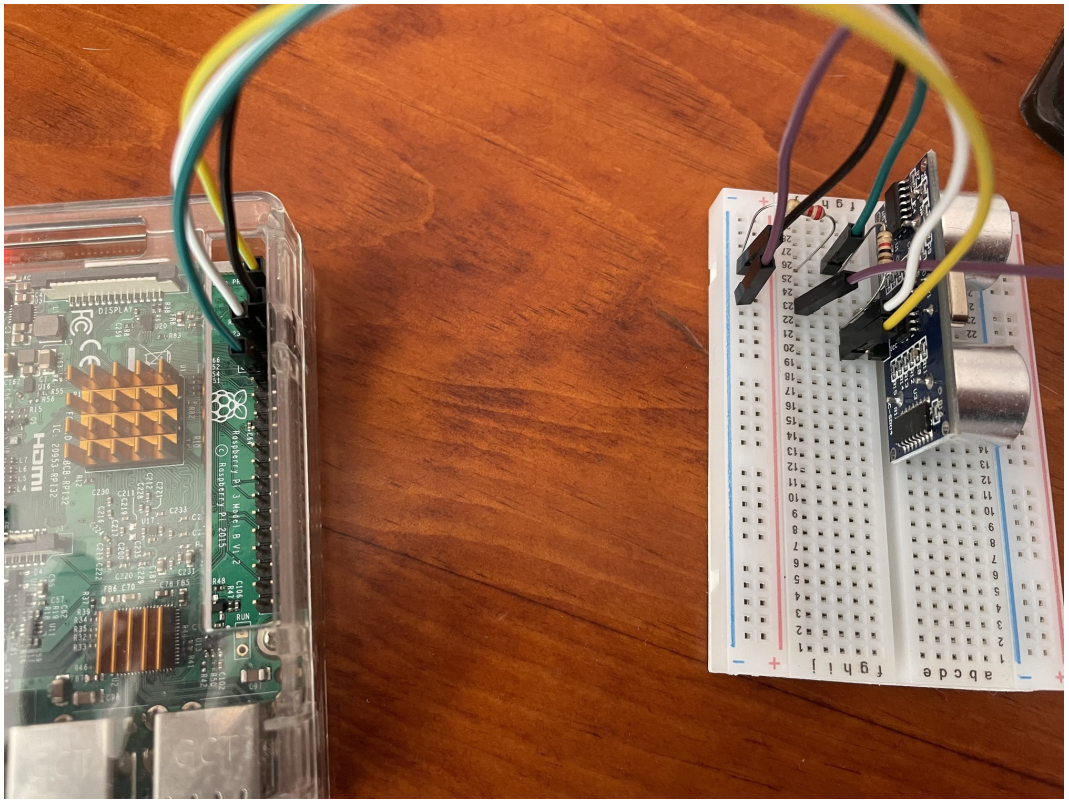

Figura 3: Esquema de conexión del US modelo HC-SR04

3.2. Programación del ultrasonidos

Una vez tengas el circuito conectado, prueba el funcionamiento del ultrasonidos con el código que se facilita en `usInstantaneo.py` y cuyo snippet se muestra a continuación. En este código se toma una única medición del US.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

try:
      GPIO.setmode(GPIO.BCM)

      PIN_TRIGGER = 4
      PIN_ECHO = 17

      GPIO.setup(PIN_TRIGGER, GPIO.OUT)
      GPIO.setup(PIN_ECHO, GPIO.IN)

      GPIO.output(PIN_TRIGGER, GPIO.LOW)

      print "Esperando a que se estabilice el US"
      time.sleep(2)

      GPIO.output(PIN_TRIGGER, GPIO.HIGH)
      time.sleep(0.00001)
      GPIO.output(PIN_TRIGGER, GPIO.LOW)
```

```
        while GPIO.input(PIN_ECHO)==0:
                inicioPulso = time.time()
        while GPIO.input(PIN_ECHO)==1:
                finPulso = time.time()

        duracionPulso = finPulso - inicioPulso
        distancia = round(duracionPulso * 17150, 2)
        print "Distancia: ", distancia, " cm"

finally:
        GPIO.cleanup()
```

3.3. Ejercicio

Adapta el código anterior para que el interfaz sea más amigable. Para ello, haz uso de tres LEDs que indiquen el acercamiento del objeto según tu criterio:

1. Luz verde. Inicialmente, será el LED que estará encendido, indicando que existe una distancia de seguridad razonable al objeto.
2. Luz amarilla. Si el objeto entra en un tramo intermedio de distancia, que consideremos prudencial, aunque no crítica.
3. Luz roja. Si el objeto entra en una zona peligrosa de distancia; esto es, está excesivamente cerca.

Es muy interesante hacer pruebas experimentales para, por ejemplo, establecer el rango del sensor; e.g. distancia mínima/máxima de detención de objetos.

Práctica 6

Uso de un *reed switch* como detector de proximidad

Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Introducción

En esta práctica vamos a construir una alarma de *puerta abierta*, usando para ello un interruptor magnético *reed switch*. En nuestro kit disponemos de estos interruptores (Figura 1), también denominados interruptores de lengüeta.

Figura 1: Interruptor de lengüeta o reed switch incluido en el kit. Imagen de *robotshop.com*

El mecanismo de un *reed switch* es sencillo. Este dispositivo incorpora dos láminas ferromagnéticas (que se pueden apreciar a simple vista) que, en presencia de un campo magnético se atraen (si es normalmente abierto, NA), o se separan (si es normalmente cerrado, NC).

En nuestro caso, para la fabricación de una alarma, lo normal es emplear uno de tipo NC. Así, en caso de que se produzca un corte de conexión, salta la alarma. En alarmas más sofisticadas, para evitar puentear este dispositivo (y que siempre parezca que está cerrado, indicando que todo está bien), se suele colocar una resistencia en serie con el interruptor.

2. Diseño del circuito

El circuito necesario para construir esta alarma es muy sencillo. Solo hemos de conectar uno de los terminales del *reed switch* a un pin de V_{CC} y, el otro, a GND. Cuando se acerque un imán, las láminas del interruptor se separarán, y con ello se cortará el circuito.

Pero si queremos controlar el estado del interruptor vía software, habrá que comprobar el estado del interruptor conectándolo a un pin GPIO de lectura con una resistencia *pull_up*.

3. Ejercicio

Construye el circuito para que se pueda conocer el estado del interruptor en todo momento, tenga cerca un imán o no (lo que simulará puerta abierta/cerrada). Para ello, haz uso —como creas conveniente— de artificios que ya conocemos como las resistencias *pull_up* y el manejo de eventos.

Una vez te asegures que estás recibiendo el estado coherentemente, añade al circuito un LED que refleje dicho estado.

Práctica 7

Uso del sensor de humedad de suelo FC-28

Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Introducción

En esta práctica vamos a trabajar con el sensor de humedad de suelo, modelo FC-28. Como puedes comprobar en tu kit, este sensor viene acompañado de un módulo comparador, modelo LM393 (cuyo *datasheet* se adjunta), junto con un cable de dos terminales. Todo ello lo puedes ver, conectado, en la Figura 1.

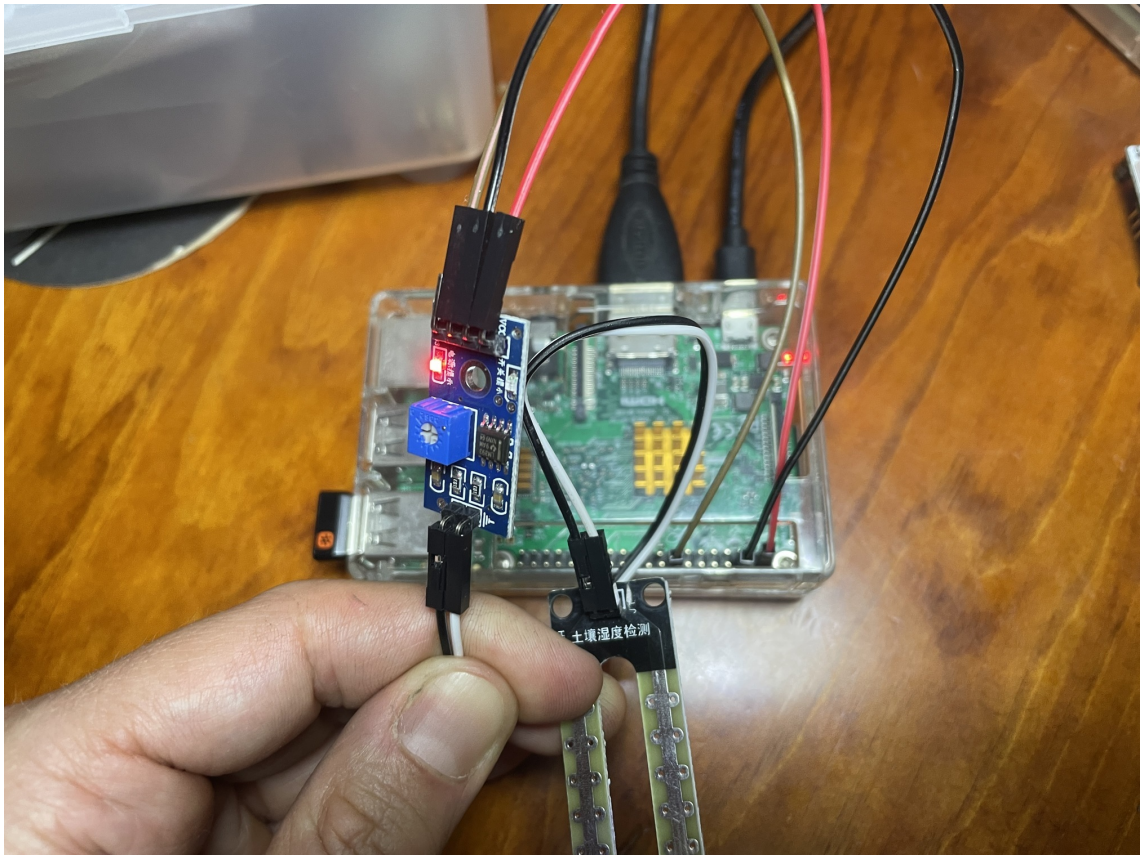

Figura 1: Esquema de conexión del circuito del sensor de humedad

La conexión de este dispositivo, gracias al módulo incluido, es sencilla. Simplemente hemos de conectar,

con el cable facilitado, los dos pines del sensor al lateral del módulo que tiene dos pines y, a continuación, fijarnos en el lateral del módulo que tiene cuatro pines.

2. Pines del módulo M393

En uno de los laterales del módulo M393 hay cuatro pines. En ellos podemos apreciar —aunque quizás necesitemos una lupa— las siguientes cuatro siglas:

- VCC. Es el pin al cual conectar uno de los pines 3V3 del raíl GPIO.
- GND. Directamente a un pin GPIO de toma de tierra.
- DO. Las siglas provienen de *Digital Output*.
- AO. Este pin es alternativo al anterior, cuyas siglas significan *Analog Output*.

3. DO vs. AO

De los cuatro pines mencionados anteriormente, ya estamos familiarizados con los dos primeros. Pero, como novedad, este módulo nos permite trabajar con la señal del sensor tanto en digital como en analógico. En nuestro caso, usaremos el pin *DO*. Este pin permite recibir la señal del sensor de forma digital; lo cual es muy útil para poder usarlo en la Raspberry Pi que, como ya sabemos, no tiene ningún pin que nos permita trabajar con una señal analógica.

En un sensor de temperatura/humedad, y como es de entender, una señal digital no tiene mucho sentido; lo suyo sería usar la señal analógica (en nuestro caso, mediante un *ADC*). La señal que nos vierte el pin *DO* es 0 o 1, en función de si la humedad sensada supera un umbral determinado. La parte positiva es que este umbral se puede adaptar/calibrar simplemente girando el tornillo que incorpora el módulo M393.

4. Para practicar

4.1. Programación

Desarrolla un programa para leer los valores del módulo FC-28. Se valorarán los siguientes ítems:

- La eficiencia de la implementación. Emplea los recursos que consideres más adecuado de entre los que hemos estudiado a lo largo del curso.
- Realiza una comparativa de eficiencia, simplemente tomando tiempos, para ver cuál de las posibles soluciones es la más adecuada.

4.2. Uso: detección de falta de agua en una planta

Como ya se ha mencionado, girando el tornillo incluido en el módulo LM393, se varía el umbral entre lo que se considera 0 (no humedad) y 1 (humedad). Ajústalo convenientemente para detectar cuándo una planta tiene la suficiente humedad y/o cuándo necesita ser regada.

En este apartado se valorará, sobre todo, la creatividad, para hacer un interfaz ameno e intuitivo para que una persona ajena al mundo de la programación pueda emplear tu programa para alertar de la falta de riego de una planta.

Como idea se plantea la siguiente. En primer lugar, un gran *feature* a ser implementado podría ser el autoarranque del programa nada más enchufar la Raspberry a la corriente. A esto se podría sumar el uso de algún LED para indicar que el programa ya está listo/arrancado, y otros para, por ejemplo, indicar la falta de humedad o humedad correcta.

Da rienda suelta a tu imaginación. Y no olvides, como siempre, añadir un vídeo del funcionamiento completo del sistema.

Práctica 8

Uso del sensor de presión junto con el de presencia

Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Introducción

En esta práctica vamos a usar el sensor de presión, modelo FSR, junto con el de presencia, tipo PIR. Ambos incluidos en el kit.

Figura 1: extraída de *digikey.es*. Sensor de fuerza piezorresistivo modelo FSR.

El sensor de fuerza o presión FSR, que se muestra en la Figura 1, es un sensor de fuerza, deformación o presión, cuyo principio de funcionamiento —como ya hemos visto— es la piezorresistividad; esto es, su resistencia eléctrica se modifica con la deformación. Como podemos sentir al tacto, está compuesto de un polímero flexible. Y en la zona circular incorpora una tinta sensible a la presión; es por ello que solo sea sensible en esta zona.

Respecto al sensor de presencia, vamos a usar un sensor tipo PIR, cuyas siglas provienen de *Passive Infrared Detection*. Es decir, que este sensor mide variaciones de luz infrarroja. En concreto, el modelo que usaremos es el HC-SR501, como el que vemos en la Figura 2.

2. El sensor de presencia HC-SR501

Ya hemos comentado que este sensor es de tipo PIR. Con esto, lo primero que se nos debería pasar por la cabeza es que ¿cómo es posible que nos detecte a nosotros, las personas? No olvidemos que la *P* de PIR proviene de *Passive*. Es decir, que este dispositivo no emite infrarrojos que reboten en el objeto a detectar,

Figura 2: extraída de *iberobotics.com*. Sensor de fuerza piezorresistivo modelo FSR.

sino que es sensible a la radiación infrarroja del objeto a detectar. Entonces, y volviendo a la pregunta, ¿cómo es posible que nos detecte a nosotros, las personas?

La respuesta parece poesía: porque todos y cada uno de nosotros brillamos con luz propia... Pero lejos de ser poesía, es una realidad. De hecho, casi todo ser vivo emite luz. No la vemos porque es una luz infrarroja muy débil; se supone que sobre 1000 veces menos intensa que el mínimo nivel visible por el ojo humano.

Siendo así, ahora ya cobra sentido el funcionamiento de este dispositivo: al pasar nosotros delante del sensor —simplemente por nuestra morfología— causamos una variación en la radiación percibida por el sensor. Y entonces, este verterá la salida `HIGH`.

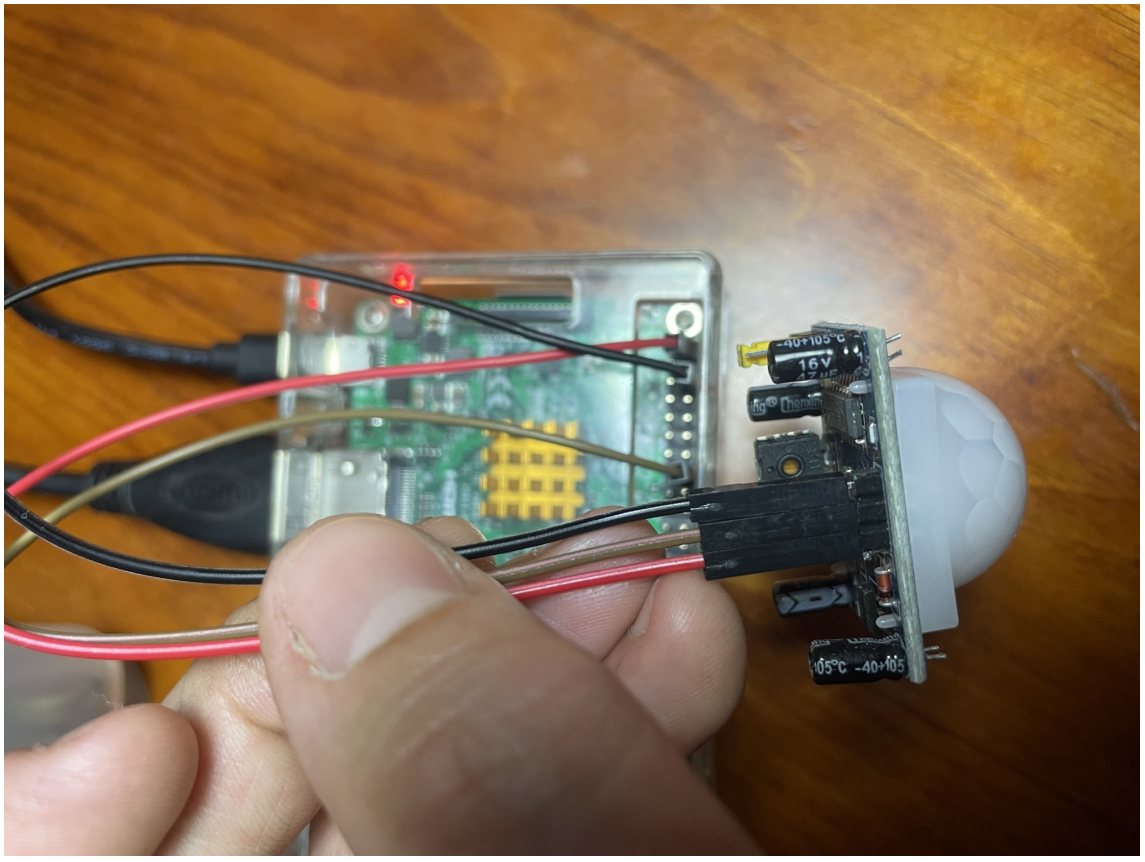

Figura 3: Esquema de conexión del circuito del sensor de presencia

Este sensor, como podemos ver en la Figura 3, tiene tres pines: GND, OUTPUT y VCC. Trabaja a $5V$, por lo que su entrada de energía la conectaremos directamente a un pin de $5V$ de la placa; GND, a tierra y, la salida, que se da a $3{,}3V$, directamente a cualquier pin GPIO. Además de esos tres pines, si nos fijamos en la Figura 4, el sensor incorpora, un *jumper* y dos tornillos de ajuste.

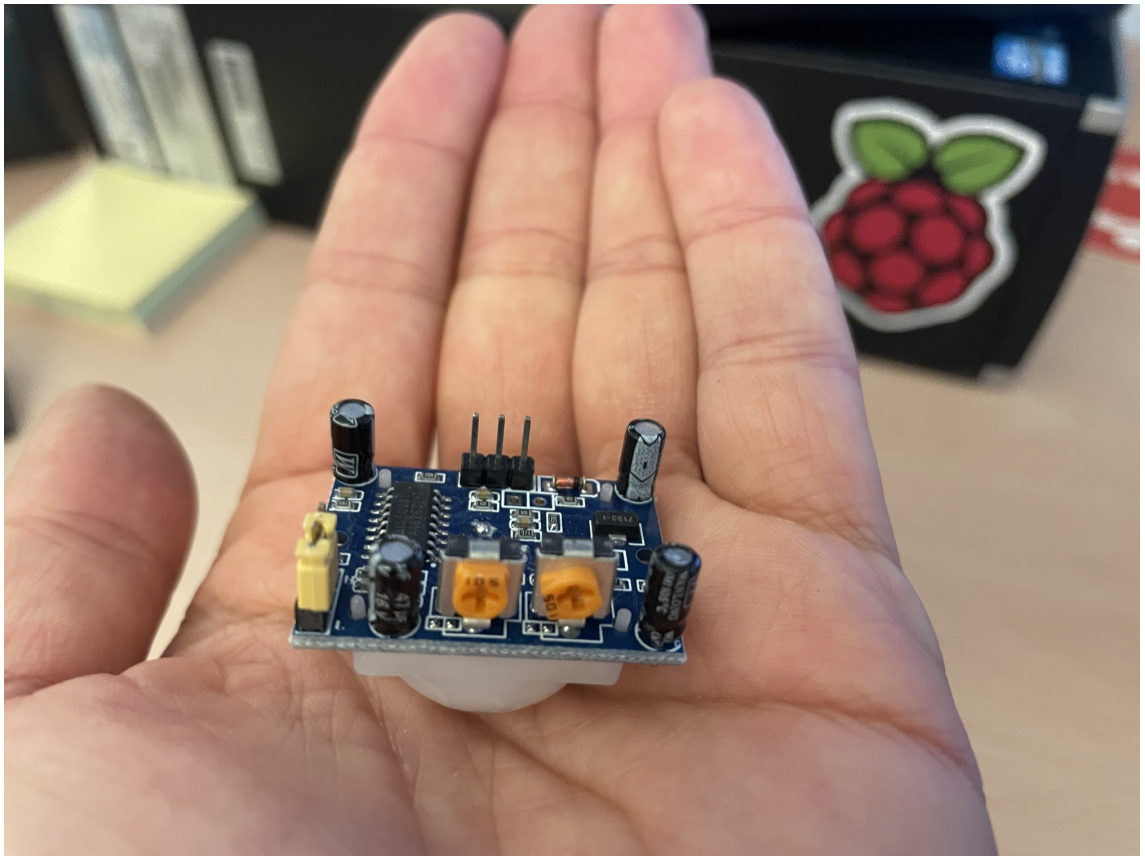

Figura 4: Elementos de ajuste que incorpora el sensor HC-SR501

Lo primero de todo es entender qué ajusta el tornillo de la derecha de la Figura 4. Como ya se ha comentado previamente, cuando se detecta movimiento, la señal de salida pasará de `LOW` a `HIGH`. Cuánto tiempo permanece la señal en estado `HIGH` es justo lo que se regula en este tornillo. Girando a la izquierda se reduce ese tiempo; a la derecha, se incrementa.

Entendido el mecanismo anterior, pasemos a analizar el del *jumper*, que está situado en la esquina inferior izquierda de la Figura 4. En la posición más esquinada estaremos en modo L, como podemos leer en la inscripción grabada —en miniatura— en la placa. En este modo, si el sensor está detectando presencia y, por tanto, la señal de salida está en alto (dentro del tiempo fijado), no volverá a dispararse un nuevo evento aunque se detectara otro movimiento. En posición H sí que se generaría un nuevo evento por cada detección de movimiento. Esto se consigue porque, bajo este modo, lo que se hace es resetear a 0 el tiempo (fijado con el primer tornillo) cada vez que se detecta movimiento.

Finalmente, el tornillo de la izquierda de la Figura 4 sirve para ajustar la sensitividad del sensor.

3. El sensor de presión FSR

El mecanismo de este sensor es mucho más sencillo que el anterior y, por tanto, mucho más fácil de conectar a la placa; al menos, tal cual lo vamos a emplear nosotros, que es en modo digital (`LOW`/`HIGH`). Para

obtener esta funcionalidad, simplemente hemos de colocar una resistencia de $1M\Omega$ en el terminal que sirve tanto para cerrar el circuito como para leer el estado.

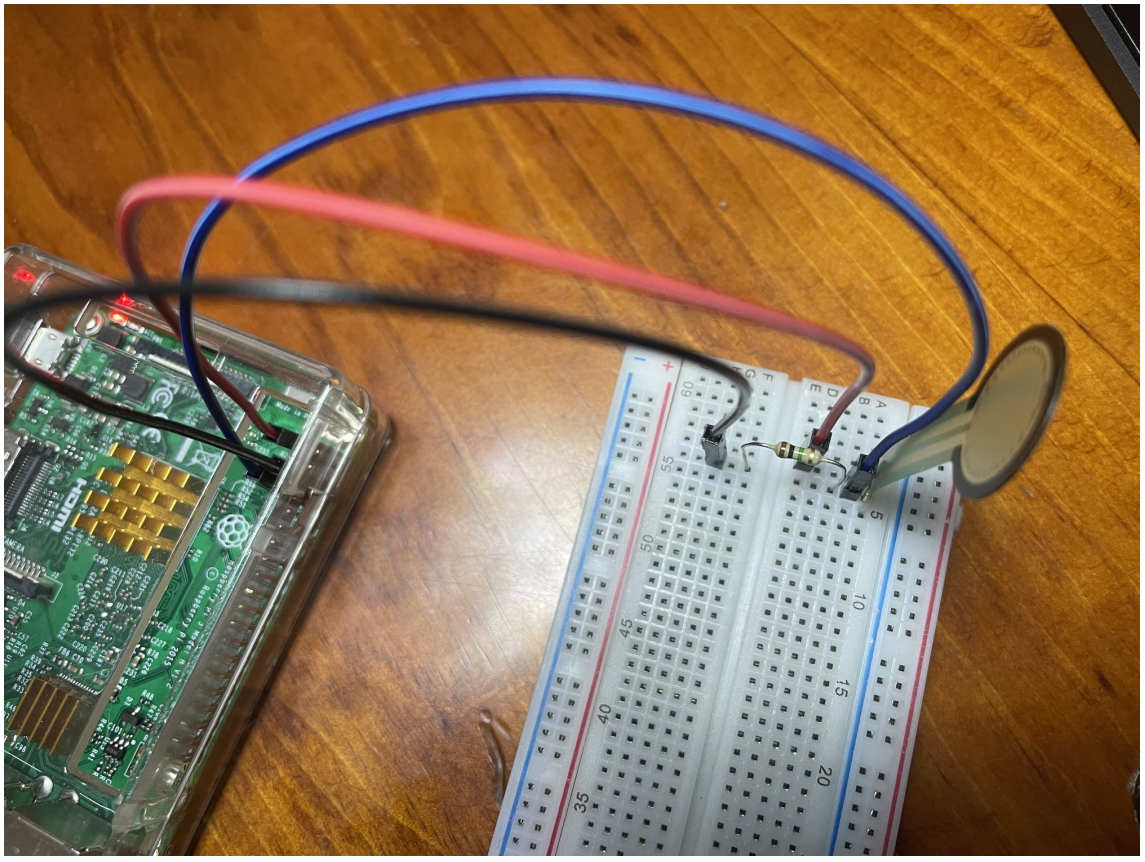

Figura 5: Esquema de conexión del circuito del sensor de fuerza

Por ello, y tal como podemos ver en la Figura 5, la línea 56 de la protoboard está bifurcada entre el pin GPIO de lectura y el pin GND de la Raspberry Pi. Este mismo comportamiento digital del sensor FSR se podría conseguir insertando un capacitor en lugar de la resistencia.

Por otro lado, y como ya ocurría con el sensor de humedad, si quisiéramos obtener una salida analógica de este sensor, necesitaríamos diseñar un circuito que emplease un convertidor de señal analógica a digital (ADC).

Ejercicios

Códigos de lectura de ambos dispositivos

Implementa dos códigos, uno para la lectura del sensor de presencia (llámalo `presencia.py`), y otro para el sensor de presión (llámalo `fuerza.py`).

Puedes partir del código que hayas implementado y que hayas elegido como más eficiente de la `Práctica 8` pues, igual que en esa práctica, tan solo has de leer el estado del pin, tanto para detectar presencia como para detectar presión.

Describe convenientemente en el `README.md`, y graba varios vídeos, del comportamiento del sensor de presencia según vayas regulando los tres elementos descritos en la Sección 2.

Diseño de un sistema de iluminación inteligente

Como ya hemos descrito en la Sección 2, el sensor de presencia detecta cualquier ser vivo pues, en mayor o menor medida, todos emitimos luz infrarroja susceptible de ser percibida por este sensor.

Vamos a diseñar un sistema de iluminación inteligente cuyo objetivo práctico sería —imaginémoslo— iluminar el jardín de una casa solo cuando haya paso de personas, no de animales.

Para ello, vamos a usar el sensor de presión a modo de botón, y solo cuando este se haya presionado (por un humano, se entiende), se activará el detector de presencia durante un tiempo prudencial (e.g. 30 segundos). Si en ese tiempo se detecta presencia, se deberá encender un LED de cortesía, igualmente durante un tiempo determinado.

En un entorno real como el que nos hemos marcado sería muy conveniente un sistema así, pues no es de extrañar que en un jardín pasen numerosos animales amigos de la noche que podrían hacer activar el sistema de iluminación (con la consecuente molestia y gasto energético y económico). Instalando, por ejemplo, un sensor de presión en cada extremo de entrada-salida de la casa que activase todo el circuito de detección que esté instalado haría que este solo entrara en funcionamiento con nuestra presencia.

La elección del sensor de presión en lugar de un botón o interruptor no es casual. Este tipo de elementos son los más apropiados para ser usados en exteriores, pues —como ya hemos visto en teoría— no se ven afectados por la lluvia/humedad, como sí los botones o interruptores.

Práctica 9

Manejo de servo con retroalimentación posicional

Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Introducción

En esta práctica vamos a trabajar con un servo continuo; esto es, se trata de un motor que gira vueltas completas, en un sentido y en otro, pero además nos puede dar realimentación de la posición que tiene, de ahí su denominación de servo.

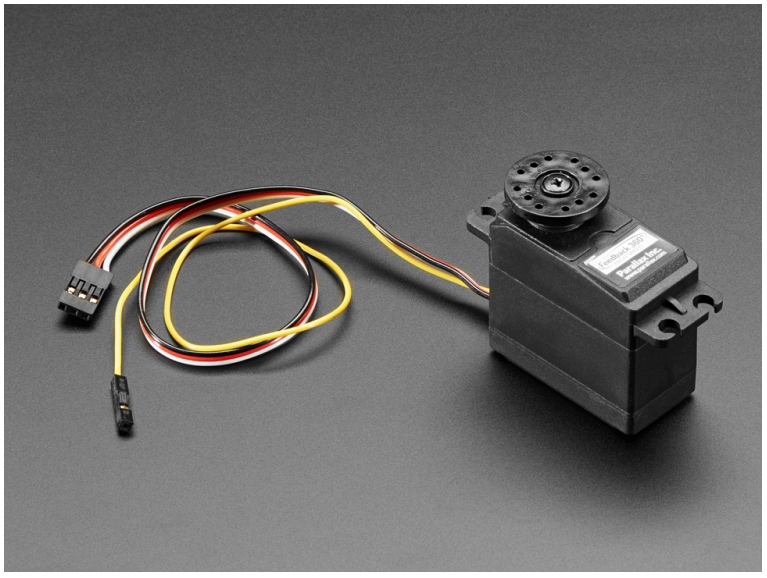

Figura 1: Modelo Feedback 360 High Speed Servo de Parallax

Un problema habitual con los servos continuos que podemos encontrar en cualquier tienda de electrónica es que generalmente no tienen rotación de 360 grados y además retroalimentación posicional. Uno de los pocos servos que podemos encontrar en el mercado que cuente con ambas cualidades y que sea económico (*DIY*) es el *Feedback* 360 *High Speed Servo* (Figura 1), de la compañía *Parallax*, y es con el que vamos a trabajar nosotros.

Otra compañía que también ofrece algo similar, y que además es muy importante en el mundo DIY, es *Dynamixel*, con su económico modelo AX-12A. Pero tengamos en cuenta que la mayoría de servos no disponen de esta retroalimentación posicional.

2. Mecanismo interno: sensor de efecto Hall

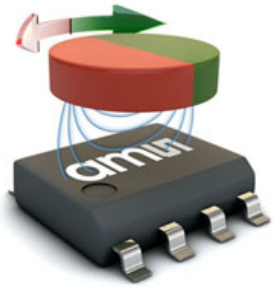

Figura 2: Sensor AS5035 de efecto Hall sin contacto

El modelo de *Parallax* es básicamente un servo continuo normal que utiliza un sensor de los que ya hemos estudiado; concretamente, un sensor de efecto Hall sin contacto (modelo AS5600, Figura 2), que proporciona retroalimentación posicional como un tren de pulsos.

3. Funcionamiento y conexión del servo

Para poder programar el servo, y como siempre, lo primero es conocer cómo funciona, y para ello lo mejor es leerse las especificaciones (adjuntas a esta práctica). De estas podemos extraer información muy útil, por ejemplo los siguientes aspectos:

- Pin de señal de retroalimentación de posición: cable amarillo.
- Periodo de envío de señal de control: 20 ms.
- Ancho de pulso de la señal: p.ej. $1280 - 1480\mu$s para girar en un sentido.
- Periodo de señal de retroalimentación ($tCycle$): $\frac{1}{910}Hz$ ($\simeq$ 1.1 ms).
- Ciclo de trabajo de $tCycle$: $2{,}9 - 91{,}7\,\%$

El esquema de conexión se puede visualizar en la Figura 3.

4. Para practicar

Se adjunta el código `servo.py` con el que podemos manipular el servo para que se mueva en las dos direcciones a máxima velocidad. Para ello, se hace uso de la librería `pigpio`; concretamente —y como novedad—, el demonio de esta, `pigpiod`. Una vez se ha lanzado el demonio, con `sudo pigpiod`, la librería `pigpio` se estará ejecutando en segundo plano aceptando comandos por cualquier intefaz, por ejemplo, del teclado a través del terminal.

Ejercicios

Ejercicio 1

En este primer ejercicio se pide encontrar los valores para hacerlo girar en ambos sentidos a distintas velocidades. Es normal que estos valores difieren ligeramente de unos servos a otros, y por ello hemos de probar hasta encontrar los valores del servo que tengamos.

Figura 3: Esquema de conexión del servo

Para hacerlo más interesante, puedes incorporar esas distintas velocidades como comandos a leer por teclado. Así como ahora se leen los caracteres w, s o x, puedes incluir otros comandos, por ejemplo los valores numéricos, lo cual sería bastante intuitivo para referenciar las distintas marchas.

Ejercicio 2

Partiendo de lo conseguido en el ejercicio anterior, en este caso se pide encontrar la ecuación lineal que permita asociar una velocidad (en m/s) a un comando de ciclo de trabajo que efectúa tal velocidad. Como no tenemos forma de medir la velocidad, lo haremos más fácil: establecemos una velocidad máxima (`MAX_SPEED`) y una velocidad mínima (`MIN_SPEED`) con valores normalizados intermedios.

Área de sección transversal de un alambre (Tema 1)

Sensores y actuadores
Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. La resistencia de un alambre

Aunque el alambre de cobre conduce electricidad extremadamente bien, sigue teniendo algo de resistencia, como todos los conductores. La resistencia de un alambre depende de tres características físicas: (a) tipo de material, (b) longitud del alambre, y (c) área de sección transversal. Además, la temperatura también puede afectar la resistencia.

Cada tipo de material conductor tiene una característica llamada resistividad, ρ. Para cada material, ρ es un valor constante a una temperatura dada. La fórmula para la resistencia de un alambre de longitud l y área de sección transversal A es:

$$R = \rho \frac{l}{A} \tag{1}$$

Esta fórmula muestra que la resistencia se incrementa con un aumento en la resistividad y la longitud, y que disminuye al aumentar su área de sección transversal. Para calcular la resistencia en $ohms$, la longitud debe estar en $pies$, el área de sección transversal en $milscirculares$, y la resistividad en $\frac{MC \cdot \Omega}{pie}$.

2. Problemas

2.1. Problema 1

Como ya hemos visto, la ecuación que relaciona el área de sección transversal con el diámetro de un alambre es la siguiente:

$$A = d^2 \tag{2}$$

Siendo A el área de sección transversal expresado en MC, y d el diámetro expresado en $Mils$.

Siguiendo la Ecuación 2, determina el área de sección transversal de un alambre con diámetro de $0{,}005 pulgadas$.

Solución

$$d = 0{,}005 pulg = 5 mils \implies A = d^2 = 5^2 = 25 MC \tag{3}$$

Problema relacionado ¿Cuál es el área de sección transversal de un alambre de $0{,}0015 pulgadas$ de diámetro?

Sol:

$$d = 0{,}0015 pulg = 1{,}5 mils \implies A = d^2 = 1{,}5^2 = 2{,}25 MC \tag{4}$$

2.2. Problema 2

Encuentra la resistencia de un alambre de cobre de $100 pies$ de longitud y área de sección transversal igual a $810{,}1 MC$. La resistividad del cobre es de $10{,}37 \frac{MC \cdot \Omega}{pie}$.

Solución

$$R = \rho \frac{l}{A} = \frac{\left(10{,}37 \frac{MC \cdot \Omega}{pie}\right)(100 pies)}{810{,}1 MC} = 1{,}280 \Omega \tag{5}$$

Problema relacionado Busca en Internet la tabla de resistencias del cobre sólido redondo y comprueba si el valor calculado anteriormente coincide. ¿De qué calibre AWG es el alambre del problema?

Sol.: sí, coincide, aunque en la tabla viene valor de resistencia $12{,}80 \frac{\Omega}{1000 pies}$, por lo que, expresado en $\frac{\Omega}{100 pies}$, que es como hemos trabajado en el ejercicio anterior, sería efectivamente el valor calculado de $1{,}280 \Omega$. Su calibre es $21 AWG$.

Manejo del calibre AWG y la sección transversal de un alambre (Tema 1)

Sensores y actuadores
Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. El valor AWG

El AWG se popularizó en Norteamérica en el año 1857 debido a que el diámetro para referenciar los diferentes calibres de elementos metálicos tales como alambres o láminas era asignado mediante una relación matemática. Dicha relación consiste en una progresión geométrica que relaciona el diámetro de dos conductores base medainte el número de divisiones intermedias. Posteriormente se extendería a latinoamérica y, actualmente, los conductores se comercializan en función de su calibre AWG.

1.1. Escala de los calibres AWG

La progresión geométrica que propuso su inventor, J.R. Brown, para calcular los diferentes calibres consiste en considerar dos calibres base, que son los correspondientes a los espesores máximo y mínimo que se podían conseguir por aquel entonces: el calibre $4/0AWG$, que tiene un diámetro de $0{,}46in$ y el calibre $36AWG$, cuyo diámetro es de $0{,}005in$. Los calibres existentes entre esas dos cotas son 39, teniendo en cuenta que existen cuatro valores $AWG_0 : 0000(4/0), 000(3/0), 00(2/0), 0(1/0)$. Así, la relación matemática o razón que relaciona un calibre con el siguiente, viene dada por la siguiente fórmula:

$$\sqrt[39]{\frac{0{,}46}{0{,}005}} = \sqrt[39]{92} = 1{,}1229 \tag{1}$$

1.2. Problema: calcular diámetro de alambre por su calibre AWG

Sabiendo el factor que relaciona un calibre con el siguiente (Ecuación 1), y tomando como referencia el calibre mayor, $36AWG$, cuyo diámetro es de $0{,}005in$, ¿podrías determinar el diámetro del alambre calibre $27AWG$?

Solución Si queremos encontrar el valor del diámetro del alambre calibre $27AWG$, necesitamos saber cuántos pasos hay entre el calibre mayor (36) y este que queremos calcular: $36 - 27 = 9$. Este valor nos indica cuántas veces hemos de aplicar el factor de la Ecuación 1. Por tanto, el diámetro del alambre calibre $27AWG$ es:

$$\varnothing_{AWG_{27}} = AWG_{36} \cdot (1{,}123)^9 = 0{,}005 \cdot (1{,}123)^9 \approx 0{,}0142in \approx 0,3607mm(1in = 25{,}4mm) \tag{2}$$

Corolario Partiendo de este sencillo ejemplo, con la Ecuación 2, se puede deducir la siguiente ecuación general para obtener el diámetro de un conductor en función de un calibre n dado en el sistema AWG:

$$\varnothing_{[in]} = 0{,}005 \cdot (1{,}123)^{(36-n)} \tag{3}$$

Problema relacionado Como se comentó previamente, existen cuatro calibres menores al $1AWG$. ¿Cómo aplicarías la fórmula que hemos deducido, Ecuación 3, para estos calibres?

Sol.: $n = 1 - c$, siendo c el número de ceros del calibre a determinar (e.g. para calibre $4/0AWG, c = 4 \implies n = 1 - 4 = -3$).

Circuitos divisores de voltaje (Tema 2)

Sensores y actuadores
Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Problema 1

Un circuito en serie actúa como divisor de voltaje. El divisor de voltaje es una aplicación importante de los circuitos en serie. Calcule la caída de voltaje a través de cada resistencia dispuesta en el divisor de voltaje mostrado en la Figura 1.

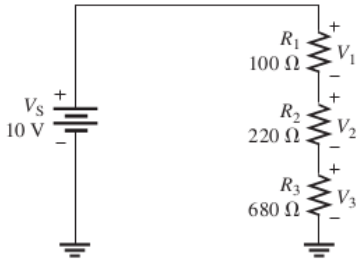

Figura 1: Circuito

Solución Si examinas el circuito unos instantes, puedes considerar realizarlo del siguiente modo. La resistencia total es de 1000 Ω. Un diez por ciento del voltaje total se encuentra a través de R_1 porque es el 10 % de la resistencia total (100 Ω es el 10 % de 1000 Ω). Asimismo, el 22 % del voltaje total se encuentra a través de R_2 porque es el 22 % de la resistencia total (220 Ω es el 22 % de 1000 Ω). Por último, R_3 cae al 68 % de la resistencia total porque 680 Ω es el 68 % de 1000 Ω.

Con los valores que se han establecido en este problema, es sencillo obtener los mentalmente: $V_1 = 0{,}10 \times 10V = 1V, V_2 = 0{,}22 \times 10V = 2{,}2V, V_3 = 0{,}68 \times 10V = 6{,}8V$. Pero no siempre será así, en cuyo caso habría que aplicar la Ley de Ohm como habitualmente. A continuación, y aunque hayamos obtenido mentalmente los valores de voltaje solitictados, vamos a comprobar que efectivamente es así:

$$V_1 = \frac{R_1}{R_T} V_S = \frac{100\Omega}{1000\Omega} 10V = 1V \tag{1}$$

$$V_2 = \frac{R_2}{R_T} V_S = \frac{220\Omega}{1000\Omega} 10V = 2{,}2V \tag{2}$$

$$V_3 = \frac{R_3}{R_T} V_S = \frac{680\Omega}{1000\Omega} 10V = 6{,}8V \tag{3}$$

Como era de esperar, y una forma sencilla de saber si lo hemos hecho bien, es comprobar que la suma de las caídas de voltaje es igual al voltaje de fuente, de acuerdo con la ley del voltaje de Kirchhoff.

Problema relacionado Si los R_1 y R_2 presentados en la Figura 1 cambian a 680 Ω, ¿cuáles son las caídas de voltaje? Respuesta: $V_1 = V_2 = V_3 = 3{,}33V$.

2. Problema 2

Calcula los voltajes entre los siguientes puntos en el divisor de voltaje de la Figura 2: (a) A a B, (b) A a C, (c) B a C, (d) B a D, (e) C a D.

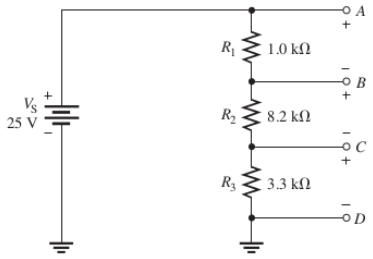

Figura 2: Circuito

Solución Primero, determina R_T:

$$R_T = R_1 + R_2 + R_3 = 1{,}0k\Omega + 8{,}2k\Omega + 3{,}3k\Omega = 12{,}5k\Omega \tag{4}$$

A continuación, aplica la fórmula del divisor de voltaje para obtener cada voltaje requerido.
(a) El voltaje de A a B es la caída de voltaje a través de R_1.

$$V_{AB} = \frac{R_1}{R_T}V_S = \frac{1{,}0k\Omega}{12{,}5k\Omega}25V = 2V \tag{5}$$

(b) El voltaje de A a C es la caída de voltaje a través tanto de R_1 como de R_2. En este caso, y siguiendo la ley de Ohm, se puede expresar la caída de voltaje en R_X como $V_X = IR_X$. Y la corriente que circula por el circuito es igual al voltaje de la fuente (*source*, V_S) dividido entre la resistencia total: $I = \frac{V_S}{R_T}$. Sustituimos esta I en la anterior ecuación y nos queda que $V_X = \frac{V_S}{R_T}R_X \implies V_X = \frac{R_X}{R_T}V_S$. Así, en este caso, $R_X = R_1 + R_2$, y nos queda:

$$V_{AC} = \frac{R_1 + R_2}{R_T}V_S = \frac{9{,}2k\Omega}{12{,}5k\Omega}25V = 18{,}4V \tag{6}$$

(c) El voltaje de B a C es la caída de voltaje a través de R_2:

$$V_{BC} = \frac{R_2}{R_T}V_S = \frac{8{,}2k\Omega}{12{,}5k\Omega}25V = 16{,}4V \tag{7}$$

(d) El voltaje de B a D es la caída de voltaje combinada a través tanto de R_2 como de R_3. En este caso, en la fórmula general, $R = R_2 + R_3$.

$$V_{BD} = \frac{R_2 + R_3}{R_T}V_S = \frac{11{,}5k\Omega}{12{,}5k\Omega}25V = 23V \tag{8}$$

(e) Por último, el voltaje de C a D es la caída de voltaje a través de R_3.

$$V_{CD} = \frac{R_3}{R_T} V_S = \frac{3{,}3k\Omega}{12{,}5k\Omega} 25V = 6{,}6V \tag{9}$$

Problema relacionado Determina cada uno de los voltajes previamente calculados si V_S se duplica.
Respuesta: $V_{AB} = 4V; V_{AC} = 36{,}8V; V_{BC} = 32{,}8V; V_{BD} = 46V; V_{CD} = 13{,}2V$.

Aplicación del código Gray quebrado a un encoder absoluto (Tema 3)

Sensores y actuadores
Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. El código Gray

En cualquier sistema de ingeniería se suele monitorizar el cambio de estado de los sensores, para así verificar que estos están funcionando adecuadamente. Pero las transiciones en el código binario normal son más difíciles de monitora en comparación con el código Gray.

El código Gray es un código alternativo al binario cuya distinción radica en el hecho de que solo cambia un bit en cada transición. Esto puede llegar a ser muy útil en sistemas de tiempo real donde el objetivo es medir una señal binaria; por ejemplo, un encoder. Podemos obtener cómodamente una conversión a este código usando una calculadora online para tal propósito[1].

Un encoder se compone de un sensor que —como ya hemos visto— hoy en día suele ser óptico o magnético, y un disco con algún tipo de codificación. Por ejemplo, en un encoder absoluto, que es el vamos a tratar en este ejercicio, el disco tiene varias bandas distribuidas de forma concéntrica y codificadas en código Gray. A cada una de estas bandas le corresponde un par emisor-sensor, lo que se traducirá en un bit; esto es, 0 o 1 según el instante.

Dicho de otro modo, según la posición del disco en un momento determinado, cada par emisor-receptor estará frente a una zona hueca o una zona opaca, lo que verterá un 0 o un 1; y así con cada par emisor-receptor, dando como salida del sistema un código binario. Es decir que, leyendo ese código binario, podemos saber la posición absoluta del disco en un momento determinado.

2. Problema

Se pretende desarrollar un encoder absoluto con 10 posiciones/giro. ¿Cómo lo harías, empleando el código Gray para codificar esas 10 posiciones?

Solución Para poder codificar 10 posiciones se necesitarán cuatro pares emisor-receptor, que darán como salida cuatro bits. La codificación en el disco, siguiendo el código Gray normal sería como se puede ver en la Tabla 1.

Como vemos en la Tabla 1, al pasar de vuelta, de la posición 9 a la 0 inicial, hay un problema, y es que —en esa transición— cambian tres bits. Para solucionarlo, se ha de reconfigurar el código Gray original a

[1] https://ncalculators.com/digital-computation/binary-gray-code-converter.htm

Posición	Código Gray
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
0	0000

Cuadro 1: Código Gray para codificar 10 posiciones

un nuevo código Gray conocido como *código Gray quebrado* o *código Gray con exceso N*.

Lo de *exceso N* proviene de que, ahora, la posición 0 del código Gray ya no será la 0000, sino que será N. Y N es el valor que, restándolo al valor en binario de cada posición del código Gray original, se corresponde con la posición que le correspondería en este nuevo código Gray quebrado.

Para calcular este número N aplicaremos la siguiente fórmula:

$$N = \frac{2^n - Imp}{2} \tag{1}$$

donde:

Imp : número de posiciones/giro (solo considerando impulsos pares)
2^n : número de posiciones múltiplo potencia de 2 inmediatamente superior a Imp

Lo veremos mejor con un ejemplo. Empecemos viendo cómo se obtiene el valor 2^n:

$$Si\ Imp = 12, 10, 8 \implies 2^n = 16 \tag{2}$$
$$Si\ Imp = 6 \implies 2^n = 8 \tag{3}$$

En nuestro caso, $Imp = 10$ (pues queremos 10 posiciones codificadas por vuelta de disco) y, por tanto, $2^n = 16$. Y ya aplicamos la Ecuación 4 para calcular N:

$$N = \frac{2^n - Imp}{2} \implies N = \frac{16 - 10}{2} = 3 \tag{4}$$

Estamos ante un *código Gray con exceso 3*, y es porque el código Gray excede en 3 al que debería para no tener el problema con que iniciábamos este ejercicio. Es decir que, el código que correspondía a la posición 9 ahora será el que corresponda a la posición 6; el de la 8 a la 5, el de la 7 a la 4,... y las nuevas posiciones 7, 8 y 9 serán las *antiguas* posiciones (que no habíamos expuesto inicialmente porque —en principio— no hacían falta) 10, 11 y 12. Veamos mejor gráficamente cómo quedaría la nueva Tabla 2:

Problema relacionado Obtén la codificación de un disco de encoder absoluto de 6 posiciones.

Solución Nos queda la Ecuación 5.

$$N = \frac{2^n - Imp}{2} \implies N = \frac{8 - 6}{2} = 1 \tag{5}$$

Y la Tabla 3.

Problema relacionado El problema anterior de que hay un salto de más de 1 bit entre la última posición y la primera, ¿se dará siempre?

Respuesta: No, solo cuando el número de posiciones a codificar no sea potencia de 2.

Posición	Código Gray original	Código Gray quebrado
0	0000	0010
1	0001	0110
2	0011	0111
3	0010	0101
4	0110	0100
5	0111	1100
6	0101	1101
7	0100	1111
8	1100	1110
9	1101	1010
0	0000	0010

Cuadro 2: Código Gray quebrado para codificar 10 posiciones

Posición	Código Gray original	Código Gray quebrado
0	000	001
1	001	011
2	011	010
3	010	110
4	110	111
5	111	101
0	000	001

Cuadro 3: Código Gray quebrado para codificar 6 posiciones

Configuración del sensor RTD como circuito puente Wheatstone (Tema 6)

Sensores y actuadores
Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. El circuito puente de Wheatstone

Un circuito puente Wheatstone, como el que vemos en la Figura 1, puede ser operado en una condición equilibrada o desequilibrada. Esta depende del tipo de aplicación. Se utiliza en una condición de equilibrio para calcular con precisión una resistencia desconocida, siendo el voltaje de salida igual a cero.

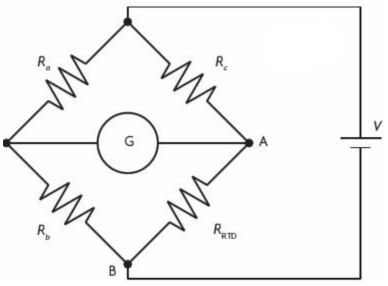

Figura 1: Circuito puente de Wheatstone

Sin embargo, más comúnmente se opera en condición de desequilibrio, conectando un transductor en una pata del puente, para medir cantidades físicas tales como deformación, presión o temperatura (como en este caso). Ocurre una condición de puente desequilibrado cuando el voltaje de salida no es igual a cero.

2. Resistencia RTD como brazo resistivo del circuito

En este ejercicio, el transductor es la resistencia RTD. Esta, como hemos visto en la teoría, cambia proporcionalmente según la variación de temperatura. Si el puente está equilibrado en un punto conocido, entonces la cantidad de desviación con respecto a la condición de equilibrio, indicada por el voltaje de salida, señala la cantidad de cambio de la temperatura, que es el parámetro que se está midiendo. Por consiguiente, el valor de temperatura puede ser determinado mediante la cantidad de desequilibrio del puente.

Dicho de otro modo, un circuito puente se diseña para medir temperatura de modo que se equilibre a cierta temperatura de referencia y desequilibre a una temperatura medida. En este ejercicio, el termistor empleado tiene un valor conocido de resistencia a 0^oC. Por simplicidad, asumimos que las otras tres resistencias del puente son iguales a la resistencia del termistor a 0^oC, por tanto $R_{RTD} = R_a = R_b = R_c$. En este caso

particular, el cambio en el voltaje de salida (ΔV_{salida}) está relacionado con el cambio en R_{RTD} mediante la siguiente ecuación:

$$\Delta V_{salida} \cong \Delta R_{RTD} \left(\frac{V_{in}}{4R} \right) \tag{1}$$

El símbolo $\cong$ es una relación de igualdad especial; indica que *está relacionado con* o *es congruente con*. Esta fórmula solo es aplicable cuando todas las resistencias del puente son iguales cuando este está equilibrado. El puente podría estar inicialmente equilibrado sin que todas las resistencias fuesen iguales; esto es, $R_a = R_c$ y $R_b = R_{RTD}$, aunque la fórmula para ΔV_{salida} sería más complicada.

3. Ejercicio

En el diagrama eléctrico de la Figura 1 se muestra la configuración que se usa generalmente para el sensor RTD. Se pide calcular el voltaje de salida (dado por el galvanómetro (G), que es el dispositivo empleado para medir la magnitud de las corrientes eléctricas) si se trabaja a 50^oC y se utiliza un sensor de temperatura resistivo (RTD) PT100.

Datos:

- $R_{RTD_{0^o}} = 100\Omega$ a 0^oC.
- $\alpha = 0{,}385\frac{\%}{^oC}$.
- $R_a = R_b = R_c = R_{RTD_{0^o}} = 100\Omega$.
- $V_{in} = 15V$.

Solución El valor de la resistencia RTD trabajando a 50^oC se calcula a partir de la siguiente ecuación:

$$R_{RTD_{50^o}} = R_0(1 + \alpha\Delta T) = 100(1 + 0{,}00385 \cdot 50) = 119{,}25\Omega \tag{2}$$

Por lo que:

$$\Delta R_{RTD} = 119\Omega - 100\Omega = 19{,}25\Omega \tag{3}$$

Y aplicando la Ecuación 1, obtenemos que:

$$\Delta V_{salida} \cong \Delta R_{RTD} \left(\frac{V_{in}}{4R} \right) = 19{,}25\Omega \left(\frac{15V}{400\Omega} \right) = 0{,}72V = 720mV \tag{4}$$

Como $V_{salida} = 0V$ cuando el puente está equilibrado a 0^oC y cambia a $720mV$, entonces:

$$V_{salida} = 720mV \tag{5}$$

Problema relacionado Calcula el valor de V_{salida} si la temperatura se incrementa a 60^oC.

Problema relacionado Calcula el valor de V_{salida} si $R_a = R_c = 1000\Omega$ y $R_b = R_{RTD} = 100\Omega$.

Solución El valor de $R_{RTD_{50^o}}$ ya lo habíamos calculado previamente:

$$R_{RTD_{50^o}} = R_0(1 + \alpha\Delta T) = 100(1 + 0{,}00385 \cdot 50) = 119{,}25\Omega \tag{6}$$

La tensión de salida en este caso se calcula como:

$$V_{salida} = V_{dcho.} - V_{izdo.} \tag{7}$$

La tensión en el lado izquierdo sería:

$$V_{izdo.} = V_{in}\frac{R_b}{R_b + R_a} = 15V\frac{100\Omega}{(100 + 1000)\Omega} = 1{,}36V \tag{8}$$

Y en el lado derecho:

$$V_{dcho.} = V_{in}\frac{R_{RTD}}{R_{RTD} + R_c} = 15V\frac{119{,}25\Omega}{(119{,}25 + 1000)\Omega} = 1{,}6V \tag{9}$$

Sustituyendo estos valores en la Ecuación 7 nos queda que:

$$V_{salida} = V_{dcho.} - V_{izdo.} = (1{,}6 - 1{,}36)V = 0{,}24V = 240mV \tag{10}$$

Configuración de celda de carga como circuito puente Wheatstone (Tema 7)

Sensores y actuadores
Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Introducción

Como ya hemos visto anteriormente, es frecuente usar un circuito puente de Wheatstone en condición de desequilibrio para medir cantidades físicas como deformación, presión o temperatura, como analizamos en el Tema 6. En este caso el enfoque reside en emplear un puente Wheatstone desequilibrado como medidor de deformación para medir ciertas fuerzas.

Un medidor de deformación es un dispositivo que exhibe un cambio de resistencia cuando se comprime o alarga por la aplicación de una fuerza externa. A medida que cambia la resistencia del medidor de deformación, el puente previamente equilibrado se desequilibra. Este desequilibrio provoca que cambie el voltaje de salida a partir de cero, y este cambio puede ser medido para determinar la cantidad de deformación. En medidores de deformación, la resistencia es extremadamente pequeña. Este cambio minúsculo desequilibra un puente Wheatstone debido a su alta sensibilidad. Por ejemplo, comúnmente se utilizan puentes Wheatstone con medidores en básculas de peso.

2. La celda de carga como composición de galgas extensiométricas

Algunos transductores resistivos experimentan cambios de resistencia extremadamente pequeños, y estos cambios son difíciles de medir con precisión por medición directa. En particular, las galgas extensiométricas (SG, *strain gages*) o medidores de deformación son uno de los transductores resistivos más útiles que convierten el alargamiento o compresión de un alambre fino en un cambio de resistencia. Cuando la deformación provoca que el alambre instalado en la galga se alargue, la resistencia se incrementa en una pequeña cantidad, pues el diámetro del alambre se hace más estrecho; y cuando el alambre se comprime (se achata), la resistencia disminuye.

Se utilizan galgas extensiométricas en muchos tipos de básculas, desde las empleadas para pesar piezas pequeñas hasta aquellas para pesar enormes camiones. En general, las galgas se montan sobre un bloque especial de aluminio que se deforma cuando hay algún peso sobre la báscula. Estas galgas son extremadamente delicadas y deben montarse apropiadamente de modo que, en general, todo el ensamble se prepare como una sola unidad llamada celda de carga (Figura 1).

Como ya hemos visto en la teoría, existe una amplia variedad de celdas de carga de diferentes formas y tamaños, según la aplicación. Una celda típica es en forma de cubo, como la mostrada en la Figura 1,

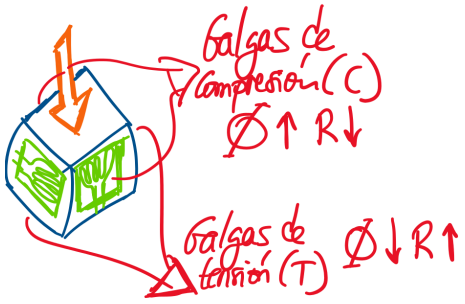

Figura 1: Celda de carga

aunque también la podemos encontrar en forma de S, utilizada para pesar y provista de cuatro galgas extensiométricas.

3. Configuración de celda de carga como puente Wheatstone

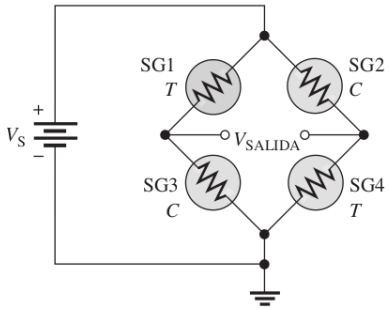

Figura 2: Circuito Wheatstone de una celda de carga

Ya hemos estudiado también que las galgas extensiométricas se montan en la celda de carga de manera que dos de ellas se alarguen (tensión) cuando se coloca una carga sobre la báscula y las otras dos se compriman. Y, casi siempre, estas celdas de carga se configuran como puente Wheatstone tal como indica la Figura 2; esto es, con las galgas a tensión (T) y a compresión (C) en brazos diagonales opuestos. De este modo siempre tendremos un V_{SALIDA} positivo, cuyo cálculo se realiza según la siguiente fórmula:

$$V_{SALIDA} = V_S \frac{SG_3^C}{SG_3^C + SG_1^T} - V_S \frac{SG_4^T}{SG_4^T + SG_2^C} \tag{1}$$

La salida del puente normalmente se digitaliza y convierte en una lectura que aparece en pantalla o es enviada a un ordenador para su procesamiento. La ventaja principal del circuito puente Wheatstone es que es capaz de medir con precisión diferencias de resistencia muy pequeñas. El uso de cuatro transductores activos incrementa la sensibilidad de la medición y hace del puente el circuito ideal para instrumentación. El circuito puente Wheatstone tiene el beneficio agregado de compensar en cuanto a variaciones de temperatura y resistencia de los alambres conectores, que de lo contrario contribuirían a provocar imprecisiones.

Además de en básculas, las galgas extensiométricas se utilizan con puentes Wheatstone en otros tipos de medición que incluyen mediciones de presión, desplazamiento, y aceleración, por mencionar algunas. En mediciones de presión, los medidores de deformación se pegan a una diafragma flexible que se alarga cuando al transductor se le aplica presión. La cantidad de flexión está relacionada con la presión, la que de nuevo se transforma en un muy pequeño cambio de resistencia.

4. Ejercicio

Calcula el valor de la tensión de salida (V_{SALIDA}) del puente mostrado en la Figura 2 considerando los siguientes datos:

- $V_S = 15V$
- R_0 (R inicial de las galgas)= 350Ω
- $GF = 2$
- $\varepsilon = 1\,\%$

Solución Calculamos en primer lugar la resistencia de las galgas, aplicando el factor de galga y teniendo en cuenta cuáles son de tensión y cuáles de compresión. Para ello, hacemos uso de la ecuación de factor de galga vista en teoría:

$$GF = \frac{\Delta R/R_0}{\varepsilon} \implies R = R_0 \cdot (1 + GF \cdot \varepsilon) \tag{2}$$

Por tanto, nos quedan las siguientes resistencias:

$$R_{SG_1^T} = R_{SG_4^T} = 350(1 + GF \cdot \varepsilon) = 350(1 + 0{,}02)\Omega = 357\Omega \tag{3}$$
$$R_{SG_2^C} = R_{SG_3^C} = 350(1 - GF \cdot \varepsilon) = 350(1 - 0{,}02)\Omega = 343\Omega \tag{4}$$

A continuación calculamos la tensión de ambos lados del puente (izdo. y dcho.):

$$V_{izdo.} = V_S \frac{R_{SG_3^C}}{R_{SG_3^C} + R_{SG_1^T}} = 7{,}35V \tag{5}$$
$$V_{dcho.} = V_S \frac{R_{SG_4^T}}{R_{SG_4^T} + R_{SG_2^C}} = 7{,}65V \tag{6}$$

Por tanto, V_{SALIDA} resulta:

$$V_{salida} = V_{dcho.} - V_{izdo.} = (7{,}65 - 7{,}35)V = 0{,}3V = 300mV \tag{7}$$

Estiramiento y estrechamiento de un cable de cobre que sufre tensión (Tema 8)

Sensores y actuadores
Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

1. Descripción

Tenemos un cable de cobre, cuyo diámetro es de $4mm$, y tiene $1m$ de largo. Si consultamos en la tablas de propiedades de los materiales, obtenemos que el módulo de Young del cobre es de $110GN$, y que su coeficiente de Poisson es de 0,34. Se pide calcular el estiramiento en longitud y estrechamiento en diámetro que sufre el alambre cuando se le cuelga un peso de $100kgf$.

2. Solución

2.1. Conversión de unidades al SI

Primeramente, vamos a pasar las cantidades que tenemos expresadas en unidades del sistema técnico al sistema internacional.

El módulo de Young se expresa en pascales (Pa). Hacemos la conversión:

$$1GN(GigaNewton) = 10^9N = 1000000000N \implies 110GN = 110000000000N = 110E9N \tag{1}$$

$$P = \frac{F}{S} \implies 1Pa = \frac{1N}{1m^2} \implies 110E9Pa = \frac{110E9N}{1m^2} \tag{2}$$

$$110GN = 110E9N = 110E9Pa \cdot m^2 = 110000MPa \cdot m^2 \tag{3}$$

Por otro lado, en el SI, el peso se expresa en newtons (N). El KilogramoFuerza o Kilopondio es la unidad de fuerza en el antiguo Sistema Técnico de Unidades (STU), y expresa la fuerza ejercida sobre una masa de 1kg por la gravedad terrestre ($9{,}80665\frac{m}{s^2}$). Hacemos la conversión:

$$1kgf = 1kp = 1kg \cdot 9{,}80665\frac{m}{s^2} = 9{,}80665\frac{kg \cdot m}{s^2} = 9{,}80665N \tag{4}$$

$$1kgf = 9{,}80665N \implies 100kgf = 100 \cdot 9{,}80665N = 980{,}665N \approx 980{,}7N \tag{5}$$

Por último, pasamos el diámetro a metros:

$$\varnothing = 4mm = 0{,}004m \tag{6}$$

2.2. Esfuerzo de tracción que ejerce el peso

En segundo lugar, vamos a calcular el esfuerzo de tracción que ejerce el peso sobre el alambre. Para ello, recordemos la fórmula que hemos estudiado en clase:

$$\sigma = \frac{F}{A} \tag{7}$$

La fuerza que ejerce el peso, F, ya lo teníamos: $F = 980{,}7N$. Nos falta calcular el área o sección del alambre, cuyo diámetro ya conocemos ($d = 0{,}004m$).

$$A = \pi r^2 = \pi 0{,}002^2 \approx 0{,}000012566 \approx 1{,}2566 \cdot 10^{-5} m^2 \tag{8}$$

Volviendo a la Ecuación 7, y sustituyendo los valores ya conocidos, F y A, nos queda que el esfuerzo de tracción es:

$$\sigma = \frac{980{,}7}{1{,}2566E-5} \approx 78043928Pa \cong 78{,}043928MPa \tag{9}$$

2.3. Deformación longitudinal que sufre el alambre

Para calcular la deformación longitudinal (L), hemos de usar la ecuación que ya hemos visto en teoría y que relaciona el módulo de elasticididad o de Young de un material con el esfuerzo y deformación que sufre este.

$$E = \frac{\sigma_L}{\varepsilon_L} \implies \varepsilon_L = \frac{\sigma_L}{E} = \frac{78{,}043928MPa}{110000MPa} = 0{,}00070949 \approx 71E-5 \tag{10}$$

2.4. Deformación transversal o de sección que sufre el alambre

Para conocer la deformación que sufre el alambre en su sección, se aplica el coeficiente de Poisson (ν), cuya relación recordamos es la que vemos en la siguiente ecuación. Aplicando el valor de deformación longitudinal (ε_L) obtenido en la Ecuación 10, y sabiendo el coeficiente de Poisson del cobre (0,34, recordemos que es un valor característico de cada material), ya podemos calcular la deformación transversal (ε_T). Veamos:

$$\nu = -\frac{\varepsilon_T}{\varepsilon_L} \implies \varepsilon_T = -\nu\varepsilon_L = -0{,}34 \cdot 71E-5 = -24{,}14E-5 \tag{11}$$

2.5. Estiramiento que sufre el alambre

Para conocer la variación longitudinal que sufre el alambre, tomamos el valor de deformación longitudinal (ε_L) calculado en la Ecuación 10, y le aplicamos la siguiente ecuación, que relaciona el esfuerzo con la longitud inicial del alambre (dado en el enunciado) y variación de esta:

$$\varepsilon_L = \frac{\Delta L}{L} \implies \Delta L = 1m \cdot 71E-5 = 71E-5m = 0{,}71mm \tag{12}$$

El alambre se ha estirado un total de $0{,}71mm$ debido al peso que se le ha colgado.

2.6. Disminución en diámetro que sufre el alambre

Por último, para estimar cuánto ha disminuido el diámetro o sección del alambre, aplicamos la misma fórmula anterior, Ecuación , pero en este caso aplicándola al diámetro del alambre; esto es, considerando la deformación transversal (ε_T) obtenida en la Ecuación 11:

$$\varepsilon_T = \frac{\Delta\varnothing}{\varnothing} \implies \Delta\varnothing = 0{,}004m \cdot -24{,}14E-5 = -9{,}66E-7m = -0{,}966\mu m \cong -1\mu m \tag{13}$$

Cálculo de presión en manómetro con resorte como indicador (Tema 8)

Sensores y actuadores
Grado en Ingeniería en Robótica Software
GSyC, Universidad Rey Juan Carlos

Descripción

Como ya hemos visto en la teoría, en los manómetros —y de forma generalizada— se usa una escala sobre la cual algún indicador, como una aguja o un resorte, refleja el valor de presión medido, relacionando la altura alcanzada por el líquido del manómetro con la presión aplicada, siempre y cuando se conozca la presión de referencia, que suele ser la atmosférica.

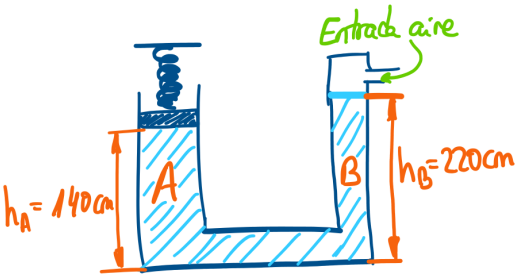

Figura 1: Manómetro de tipo columna en U

Para este problema, disponemos de un manómetro de tipo columna, cuyo líquido interior es aceite. El lado cerrado está conectado a un pistón, que a su vez está conectado a un muelle o resorte. Inicialmente, cuando el pistón indica el valor 0, este toca el muelle sin comprimirlo; sería la posición de referencia.

Se pide obtener la presión medida por el manómetro, p, cuando el muelle se comprime $30cm$.

Datos:

1. Constante elástica del muelle: $K = 340\frac{kN}{m}$
2. Densidad del aceite a 20^oC y $1atm$: $\rho_m = 0{,}85\frac{g}{cm^3}$
3. Densidad del aire a 20^oC y $1atm$: $\rho = 1{,}29E-3\frac{g}{cm^3}$
4. Diámetro del lado A: $\varnothing_A = 50cm$

5. Diámetro del lado B: $\varnothing_B = 30cm$
6. Presión atmosférica: $1atm = 1013{,}25hPa$
7. Variación longitudinal del muelle: $\Delta_l = \Delta_{h_A} = 30cm$
8. Altura del lado cerrado: $h_A = 140cm$
9. Altura del lado abierto: $h_B = 220cm$

Solución

0.1. Peso del pistón

En primer lugar, vamos a calcular cuál es el peso del pistón. Recordemos que, gracias al peso de este elemento, se consigue la posición inicial de equilibrio cuyos datos se han enumerado.

Para este propósito, debemos emplear la ecuación vista en teoría, y que nos relaciona la presión que se ejerce por uno de los vasos del manómetro (la que se quiere medir) relativamente a la presión atmosférica, que es la aplicada de forma natural en el otro vaso.

$$p = p_{atm} + \rho_m gh - \rho gd \tag{1}$$

Como $\rho \lll \rho_m$ ($\rho_m = 0{,}85$ y $\rho = 1{,}29E-3$), se puede simplificar la Ecuación 1, y nos quedaría del siguiente modo:

$$p = p_{atm} + \rho_m gh \tag{2}$$

No debemos confundir la h con las alturas de los líquidos en ambos lados del manómetro. Esta h es la diferencia de dichas alturas en la situación de equilibrio. Esto es:

$$h = h_B - h_A = 2{,}2 - 1{,}4 = 0{,}8m \tag{3}$$

Sustituyendo este valor de h, y los otros ya conocidos ($p_{atm} = 1atm = 1013{,}25hPa = 101325Pa$, $\rho_m = 0{,}85\frac{g}{cm^3} = 0{,}85\frac{kg}{l}$), en la Ecuación 2, nos queda:

$$p = p_{atm} + \rho_m gh \implies p = 101325 + 0{,}85 \cdot 9{,}8 \cdot 0{,}8 = 101325 + 6{,}664 \approx 101331{,}7Pa \tag{4}$$

Siendo esta la presión ejercida por el pistón, y conociendo la superficie de este ($\varnothing = 50cm = 0{,}5m \implies r = 0{,}25m$), podemos ya calcular el peso del mismo:

$$p = \frac{F}{S} = \frac{P}{S} \implies P = p \cdot S = 101331{,}7 \cdot \pi 0{,}25^2 \approx 19896{,}5N \tag{5}$$

0.2. Presión medida por manómetro

Una vez conseguido el peso del pistón, gracias al cual se consigue la situación de equilibrio descrita, podemos calcular qué presión se está ejerciendo en el lado B (es la presión que queremos medir) cuando el muelle sufre un acortamiento $\Delta_l = 30cm = 0{,}3m$.

En primer lugar, calculamos qué desnivel se sufre en el lado B (Δ_{h_B}), cuando en lado A se produce esa diferencia ($\Delta_{h_A} = 0{,}3m$. Para ello, igualamos el volumen que se ocupa en el lado A (V_A) al volumen que se desaloja en el lado B (V_B).

$$V_A = V_B \tag{6}$$
$$S_A \cdot \Delta_{h_A} = S_B \cdot \Delta_{h_B} \tag{7}$$
$$\cancel{\pi} \cdot r_A^2 \cdot \Delta_{h_A} = \cancel{\pi} \cdot r_B^2 \cdot \Delta_{h_B} \tag{8}$$
$$\frac{r_A^2 \cdot \Delta_{h_A}}{r_B^2} = \Delta_{h_B} \tag{9}$$
$$\Delta_{h_B} = \frac{0{,}25^2 \cdot 0{,}3}{0{,}15^2} = \frac{5}{6} \approx 0{,}83m \tag{10}$$

En este punto conviene recordar la ley de Hooke. Cuando estiramos o comprimimos un muelle, la fuerza recuperadora es proporcional al cambio de longitud x de la posición de equilibrio (Ecuación 12). Con K como constante elástica del muelle. El signo menos se debe a que la fuerza recuperadora es siempre opuesta a la deformación.

$$P = -K \cdot \Delta_{h_A} \tag{11}$$
$$\tag{12}$$

Sustituimos la P de la Ecuación 12 en la ecuación de presión, para conocer la presión que se está ejerciendo sobre el pistón para desplazarlo (en sentido ascendente):

$$P = p_A \cdot S \tag{13}$$
$$-K \cdot \Delta_{h_A} = p_A \cdot S \tag{14}$$
$$p_A = \frac{-K \cdot \Delta_{h_A}}{S} \tag{15}$$
$$p_A = \frac{-340000 \cdot 0{,}3}{\pi \cdot 0{,}25^2} \tag{16}$$
$$p_A = \frac{-102000}{0{,}1964} \tag{17}$$
$$p_A \approx -519348{,}3Pa (= \frac{N}{m^2}) \tag{18}$$

0.3. Bonus: demostración

Recordemos que en la Ecuación 4 ya obtuvimos la presión que ejerce el pistón por su propio peso: $101331{,}7Pa$ (sentido descendente). Así, la presión real de empuje o avance que se está ejerciendo sobre el pistón para que este se desplace es:

$$p_{empuje} = p_{aparente} - p_{peso} \tag{19}$$
$$p_{empuje} = -519348{,}3Pa - 101331{,}7Pa \tag{20}$$
$$p_{empuje} \approx -620680Pa \tag{21}$$

Como era de esperar, la presión real de empuje ejercida sobre el pistón es mayor que la obtenida en la Ecuación 18, pues para desplazar el pistón Δ_{h_A}, ha habido que *vencer* además su peso. Nótese que el símbolo menos indica el sentido en que se está ejerciendo la presión, y que en este caso hemos tomado como tal el sentido ascendente. Sería perfectamente válido haber considerado el sentido descendente como negativo en la Ecuación 4 y, por tanto, en la Ecuación 18 se consideraría el sentido ascendente como positivo. El resultado no se vería alterado; solo su signo.

Pero claro, hemos de tener en cuenta que la presión atmosférica está *ayudando* en esa presión de empuje calculada en la Ecuación 18. Así que, para calcular la presión absoluta que se está ejerciendo y, por tanto,

la que debería medir el manómetro, debemos restar esa *ayuda* de la atmósfera.

Y es en este punto donde adquiere vital importancia el hecho de tener en cuenta la situación de equilibrio inicial, pues es así porque la presión debida al peso del pistón es la misma que la presión atmosférica. Por tanto, se anulan y obtenemos que que:

$$p_{absoluta} = p_{empuje} - p_{atm} \tag{22}$$
$$p_{absoluta} = -620680Pa + 101325Pa \tag{23}$$
$$p_{empuje} = -519335Pa \tag{24}$$

Vemos que, efectivamente, la presión absoluta obtenida en la Ecuación 24, $519335Pa$, es similar a la presión aparente calculada en la Ecuación 18, $519348Pa$. No es exactamente igual por los redondeos aplicados de forma sucesiva, pero en la práctica serían valores iguales; de otro modo, no habría situación de equilibrio.