7 SENSORES

Más divertido que hacer parpadear LEDs es utilizar sensores de todo tipo para detectar qué está pasando "ahí fuera" y reaccionar en consecuencia. Desgraciadamente, cada sensor tiene sus propios métodos de conexión: algunos necesitan resistencias "pull-up" y otros no, algunos necesitan fuentes de alimentación propias y otros no, algunos trabajan a mucha tensión y otros no, etc. En este capítulo se presentarán los sensores más comunes, con ejemplos de circuitos donde se usan y código Arduino que los hacen funcionar.

También se indicará para cada tipo de sensor específico qué productos concretos podemos encontrar en diferentes distribuidores. De todas formas, si se desea, se puede adquirir cómodamente de una sola vez un conjunto de diversos sensores gracias al "Sensor pack 900" de Adafruit (código de producto nº 176) o el "Sensor Kit" de Sparkfun (código de producto 11016). El primero incluye un LED infrarrojo y un sensor infrarrojo específicos para control remoto, un sensor de luz, un sensor de temperatura, un sensor de inclinación, un sensor de golpes (usable como zumbador), un sensor de campo magnético (junto con un imán), un sensor de fuerza y un acelerómetro. El segundo incluye un sensor infrarrojo específico para control remoto, un sensor de luz, un sensor de flexión, un sensor de golpes y vibraciones, un sensor de campo magnético (junto con un interruptor sensible a él –lo que se llama un "reed switch"–), un sensor de fuerza, un sensor de humedad, un sensor de distancia, un sensor de movimiento, un acelerómetro, un giroscopio, una brújula (un magnetómetro) y un sensor de presión atmosférica (un barómetro). Además, incluye un potenciómetro de membrana fina con recorrido lineal (producto nº 8680).

Otro kit de sensores interesante es el ofrecido por Cutedigi con código de producto H21, el cual contiene un sensor de temperatura, de humedad, de sonido, de efecto Hall, de inclinación, de obstáculos, de fuego, de metal, un acelerómetro, una brújula, un LDR, un "reed switch"… además de un emisor y receptor de infrarrojos, un pulsador, un zumbador, un LED RGB, un optointerruptor, y más.

SENSORES DE LUZ VISIBLE

Fotorresistores

Los sensores de luz, tal como su nombre indica, son sensores que permiten detectar la presencia de luz en el entorno. A veces se les llama "celdas CdS" (por el material con el que suelen estar fabricados, sulfuro de cadmio) o también "fotorresistores" y LDRs (del inglés "Light Dependent Resistor"), ya que básicamente se componen de una resistencia que cambia su valor dependiendo de la cantidad de luz que esté incidiendo sobre su superficie. Concretamente, reducen su resistencia a medida que reciben más intensidad de luz.

Suelen ser pequeños, baratos y fáciles de usar; por esto aparecen mucho en juguetes y dispositivos domésticos en general. Pero son imprecisos: cada fotorresistor reacciona de forma diferente a otro, aunque hayan sido fabricados en la misma tongada. Es por eso que no deberían ser usados para determinar niveles exactos de intensidad de luz, sino más bien para determinar variaciones en ella, las cuales pueden provenir de la propia luz ambiente ("amanece o anochece") o bien de la presencia de algún obstáculo que bloquee la recepción de alguna luz incidente. También se podría tener un sistema de fotorresistores y comparar así cuál de ellos recibe más luz en un determinado momento (para construir por ejemplo un robot seguidor de caminos pintados de blanco en el suelo o de focos de luz, entre otras muchas aplicaciones).

Otro dato que hay que saber es que su tiempo de respuesta típico está en el orden de una décima de segundo. Esto quiere decir que la variación de su valor resistivo tiene ese retardo respecto los cambios de luz. Por tanto, en circunstancias donde la señal luminosa varía con rapidez su uso no es muy indicado (aunque

también es verdad que esta lentitud en algunos casos es una ventaja, porque así se filtran variaciones rápidas de iluminación).

A la hora de adquirir un fotorresistor hay que tener en cuenta además otra serie de factores: aparte del tamaño y precio, sobre todo hay que mirar también la resistencia máxima y mínima que pueden llegar a ofrecer. Estos datos lo podremos obtener del datasheet que ofrece el fabricante. De hecho, en el datasheet no solo podemos consultar estos dos datos extremos sino también todos los valores intermedios de resistencia, gracias a un conjunto de gráficas que nos indican cómo varía de forma continua (y generalmente logarítmica) el valor resistivo del fotorresistor en función de la cantidad de luz recibida, medida en unidades lux. Con esta información podemos conocer, sabiendo la luz que incide sobre el LDR, qué valor de resistencia ofrece este (y viceversa: sabiendo la resistencia que ofrece, podemos deducir la cantidad de luz que recibe el sensor).

En el párrafo anterior se menciona un "conjunto de gráficas" y no una sola porque el comportamiento del fotorresistor depende de la temperatura ambiente: según sea ésta, la variación de la resistencia respecto la luz incidente será una u otra. También se menciona que esta variación es "generalmente logarítmica"; es posible que haya casos donde no lo sea, pero lo que es seguro es que, por la propia naturaleza intrínseca de los fotorresistores, la relación entre cantidad de luz recibida y resistencia resultante nunca será lineal. Esto significa, por ejemplo, que si la intensidad lumínica es de 10 lux y se mide una resistencia de 100 Ω, cuando esta sea 20 lux la resistencia no tendrá por qué ser de 50 Ω.

Otro dato a consultar en el datasheet para tenerlo en cuenta es la sensibilidad. Los fotorresistores no detectan del mismo modo los diferentes tipos de luz; concretamente, suelen ser más sensibles a cambios en luces de color verde que en los de otros colores. Además, existen una longitudes de onda mínimas (400 nm, normalmente) y máximas (600 nm, normalmente) más allá de las cuales no detectan nada. Esta información la podemos encontrar en forma de gráfica que muestra la respuesta del fotorresistor en función de la longitud de onda recibida.

Breve nota sobre el espectro electromagnético:

Llamamos "luz" a las ondas que son de tipo electromagnético. Por tanto, como ondas que son, una de sus características que podemos estudiar es su frecuencia –medida en Hz– o, alternativamente, su longitud de onda –medida en nanómetros (10^{-9} metros)–. Ambas magnitudes están relacionadas por la expresión $\lambda = c/v$

(donde λ representa la longitud de onda en el vacío, v la frecuencia y c la velocidad de la luz en el vacío, que es una constante igual a 299.792.458 m/s).

Podemos clasificar diferentes tipos de luz según su longitud de onda: así tenemos los rayos gamma y los rayos X (con menor longitud de onda), pasando de forma continua (¡la luz es una magnitud analógica!) por la luz ultravioleta, la luz visible y los rayos infrarrojos, hasta llegar a las ondas electromagnéticas de mayor longitud de onda como son las ondas de radio. El conjunto de todas estas ondas es lo que se llama el espectro electromagnético

El ojo humano no es capaz de percibir todo el espectro electromagnético: solo una pequeña parte identificada como "luz visible". Esto significa que en nuestro día a día, realmente estamos rodeados de radiación electromagnética que no vemos. Aunque no hay límites exactos para la zona visible del espectro (depende de cada persona), se suelen tomar como valores aceptados las ondas que tengan una longitud de onda entre los 400 nm y 700 nm.

Dentro del espectro visible, dependiendo de la longitud de onda concreta que tenga una determinada onda, esta se verá de un color u otro. Así, podemos decir aproximadamente que la luz visible de longitud de onda entre 400 y 450 nanómetros es de color violeta, entre 450 y 495 de color azul, entre 495 y 570 de color verde, entre 570 y 590 de color amarillo, entre 590 y 620 de color anaranjado y entre 620 y 700 de color rojo.

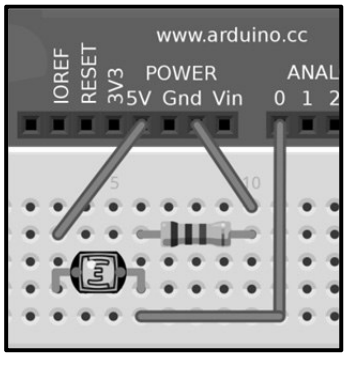
Por otro lado, la tensión aceptada por estos dispositivos puede ser prácticamente cualquiera (hasta los 100 V). Debido a que los fotorresistores no son más que resistencias, no están polarizados, así que sus terminales se pueden conectar en nuestros circuitos en ambos sentidos.

La manera más fácil de comprobar que un sensor de luz funcione es conectar sus terminales a un multímetro en modo medida de resistencia y hacerle incidir más o menos luz. Si vemos que responde (hay que vigilar con el cambio de escala), ya podremos empezar a diseñar nuestros proyectos con él.

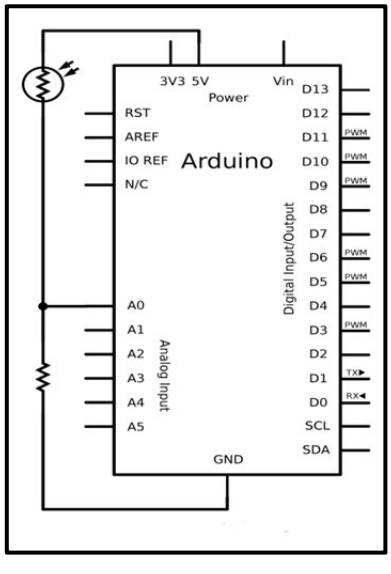
Lo primero que hemos de saber es cómo se conecta un sensor de luz a nuestro circuito. Lo que haremos será conectar uno de los terminales del sensor a la alimentación y el otro, a través de una resistencia "pull-down" (sobre cuyo valor resistivo adecuado discutiremos unos párrafos más abajo), a tierra. Además, desde un punto entre el fotorresistor y la resistencia "pull-down" conectaremos un "tercer cable" hacia una entrada analógica de nuestra placa Arduino para que esta pueda leer el voltaje analógico a medir. Este voltaje recibido podrá oscilar entre 0 V y el

voltaje que alimente al fotorresistor: en las figuras siguientes mostraremos el fotorresistor alimentado con los 5 V del pin “5V” de la placa Arduino, pero también podría alimentarse perfectamente con los 3,3 V del pin “3V3”, por ejemplo.

Si usamos una resistencia “pull-down”, cuanto mayor voltaje recibamos por la entrada analógica de la placa Arduino significará que más luz incide en el sensor. Si hubiéramos utilizado una resistencia “pull-up”, sería al revés: a mayor voltaje recibido significaría que hay más oscuridad. Nosotros, tal como hemos comentado, utilizaremos una resistencia “pull-down”, por lo que, en definitiva, el montaje sería similar a este:



Y el esquema eléctrico a este (aquí se puede apreciar un símbolo para identificar el fotorresistor, no visto hasta ahora):



El truco para entender en profundidad este montaje es ver que a medida que la resistencia del fotorresistor va decreciendo (porque le incide más luz), la resistencia total del conjunto de resistencias en serie fotorresistor + pull-down también decrece. Por la Ley de Ohm, esto hará que (al mantenerse un voltaje de alimentación fijo en todo el circuito –de 5 V–) la intensidad de corriente aumente a través de todo ese circuito. Pero como la resistencia "pull-down" es fija, por la misma Ley de Ohm, si la intensidad que la atraviesa ha aumentado, también lo habrá hecho el voltaje entre sus terminales. Que es de hecho lo que medimos con el "tercer cable": el voltaje existente entre los terminales de la resistencia "pull-down". Si la resistencia del fotorresistor fuera despreciable –al haber mucha luz–, ese voltaje medido sería 5 V; si la resistencia del fotorresistor fuera tan grande que abriera el circuito interrumpiendo el paso de electrones –al haber mucha oscuridad–, el voltaje medido sería 0 V. Entre ambos casos extremos tendremos medidas intermedias.

Lo explicado en el párrafo anterior se puede resumir en la siguiente fórmula, obtenida a partir de la Ley de Ohm y del hecho que la intensidad que atraviesa ambas resistencias es la misma: $\mathbf{V_{med} = (R_{pull} / (R_{pull} + R_{foto}))\cdot V_{fuente}}$, donde V_{fuente} es el voltaje aportado por la fuente de alimentación, V_{med} es el voltaje recibido por el pin de entrada analógico (es decir, el existente entre los terminales de la resistencia "pull-down", el cual puede valer entre 0 V y V_{fuente}), R_{pull} es el valor de la resistencia "pull-down" (fijo) y R_{foto} es el valor de la resistencia del fotorresistor. De aquí se puede comprobar lo que ya hemos dicho: que cuando aumenta la intensidad de luz (como la resistencia del fotorresistor disminuye), también aumenta el voltaje medido, y viceversa.

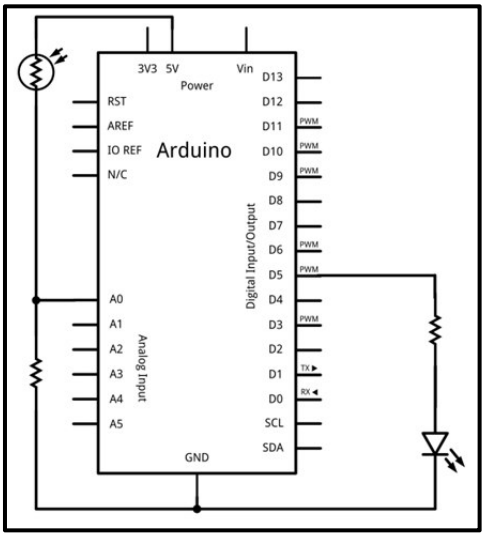
No obstante, en realidad V_{med} no es el valor con el que trabajamos en nuestra placa Arduino, porque esta utiliza siempre un conversor analógico-digital para realizar un mapeo de todos valores analógicos recibidos (los cuales pueden oscilar entre 0 V y 5 V –suponiendo que el voltaje proporcionado por la fuente son 5 V–) a valores digitales (que van entre 0 y 1023). Estos valores digitales son los que la placa Arduino entiende en realidad y con los que trabajaremos en nuestros sketches. La buena noticia es que la conversión de valores analógicos a digitales se puede expresar por una simple regla de proporcionalidad así: $\mathbf{V_{convertido} = V_{med} \cdot 1023/5}$. A partir de aquí, si sustituimos esta expresión en la fórmula del párrafo anterior, y despejamos de allí R_{foto}, llegamos a la expresión siguiente: $R_{foto}=(R_{pull}\cdot 1023/V_{convertido}) - R_{pull}$, la cual nos permite conocer por fin cuál es el valor actual de la resistencia del fotorresistor a partir del voltaje digitalizado obtenido por la placa Arduino.

El paso natural siguiente, una vez conocido el valor de la resistencia del fotorresistor, sería averiguar a qué cantidad de iluminación se corresponde. Esto se

puede consultar en las gráficas del datasheet, tal como se ha comentado previamente. No obstante, este paso no suele hacerse ya que normalmente los fotorresistores se utilizan para comparar iluminaciones (entre diferentes sitios o entre distintos intervalos de tiempo) más que para obtener valores concretos de iluminación.

Para saber el valor adecuado de la resistencia "pull-down" que tenemos que colocar en nuestro circuito, tendríamos que conocer (a partir del datasheet) los distintos valores numéricos concretos que puede adquirir nuestro fotorresistor a lo largo de distintas iluminaciones (R_{foto}) y utilizarlos, junto con el valor del R_{pull} obtenido, en la fórmula ya vista en la página anterior $V_{med} = (R_{pull} / (R_{pull} + R_{foto})) \cdot V_{fuente}$ para observar qué V_{med} hipotético obtendríamos. Haciendo esto, veremos que en la gran mayoría de los casos, el comportamiento de R_{foto} respecto a la iluminación recibida hace que valores elevados de R_{pull} (por ejemplo, 10 KΩ) saturen rápidamente las lecturas en entornos brillantes. Es decir, hace que el sensor llegue a medir el tope de los 5 V con una iluminación relativamente baja y no sea por tanto capaz de distinguir entre un ambiente bien iluminado de otro muy bien iluminado. En cambio, valores menores de R_{pull} (como por ejemplo, 1 KΩ), sí permitirán detectar cambios en la luz más brillante pero harán que no sea posible distinguir diferencias en niveles oscuros. Por lo tanto, dependiendo del entorno donde situemos nuestro proyecto, deberemos elegir una R_{pull} de 10 KΩ (para entornos oscuros) o de 1 KΩ (para entornos iluminados), o bien utilizar algún tipo de potenciómetro ajustable.

Ejemplo 7.1: Comprobemos ya cómo se comporta un fotorresistor cuando lo conectamos a una placa Arduino. Para ello podemos montar el siguiente circuito; fijarse que el LED se ha de conectar a un pin PWM.



Lo que queremos es utilizar el valor del voltaje leído en el pin de entrada analógico (el nº 0 en este caso) para iluminar consecuentemente el LED: cuanto menos luz detecte el fotorresistor, más brillante iluminará el LED (por eso es necesario que el LED reciba una señal analógica también, a través de un pin PWM, el nº 5 en este caso). Además, se muestra por el canal serie los valores obtenidos por el fotorresistor.

```
int valorcds;   //Valor obtenido
int brilloLED;  //Valor enviado al LED
void setup(void) {
      Serial.begin(9600);
}
void loop(void) {
      valorcds = analogRead(0);
/*Además de imprimir "valorcds" tal cual, también se podría haber
comprobado si este es menor o mayor que una cantidad dada, y haber
imprimido un mensaje tal como "Oscuro", "Normal", "Brillante", etc*/
      Serial.println(valorcds);
/*El valor obtenido "valorcds" será mayor cuanto más brillante sea el
entorno. En cambio, el LED ha de iluminar más cuanto más oscuro sea
el entorno. Es decir, "brilloLED" ha de ser mayor cuanto menor sea el
valor de "valorcds". Por eso, hemos de invertir "valorcds" para que
su valor pase de una escala de 0 a 1023 a otra de 1023 a 0.*/
      valorcds=1023-valorcds;
/*Y ahora, tal como ya hemos visto en ejemplos anteriores, hemos de
mapear "valorcds" para que caiga dentro del rango admitido para la
salida PWM. Es decir, pasar un valor que está entre 0 y 1023 ha otro
que está entre 0 y 255.*/
      brilloLED = map(valorcds, 0, 1023, 0, 255);
      analogWrite(5, brilloLED);
      delay(100); //Para que se pueda ver el nuevo brillo
}
```

Con lo ya sabido, podríamos realizar un simple detector de presencia. Si mantuviéramos iluminado el fotorresistor de forma constante, al interponerse algún obstáculo entre la fuente de luz y el fotorresistor, este detectaría una bajada brusca de intensidad lumínica. Usando el mismo circuito del ejemplo anterior, podríamos modificar levemente el sketch para enviar una señal digital de salida al LED, de forma que esta valiera HIGH (encendiendo el LED) si el fotorresistor detectara un valor por debajo de un cierto valor umbral elegido por nosotros (y por tanto, detectara que alguien se interpone entre la luz y el sensor) o que valiera LOW (apagando el LED) si el valor "valorcds" fuera mayor que dicho umbral (y por tanto, se detectara una incidencia "normal" de la luz en el sensor). Se deja como ejercicio.

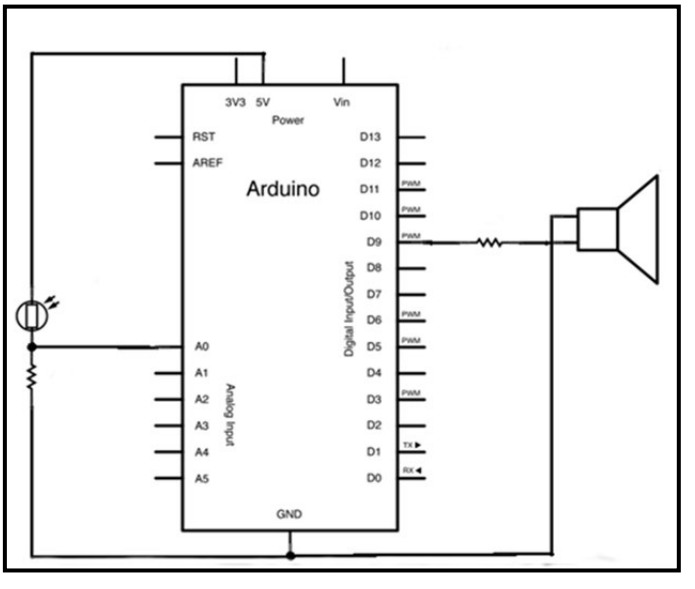
Por otro lado, ya hemos comentado antes que las lecturas obtenidas por un fotorresistor como los que usamos en nuestros proyectos (por ejemplo, el producto nº 9088 de Sparkfun o el nº 161 de Adafruit) no suelen ser muy precisas. Es muy recomendable, por tanto, calibrar estos componentes antes de empezar a trabajar con ellos (por ejemplo, dentro de la función "setup()". Calibrar significa fijar el valor mínimo y máximo del rango de posibles valores a leer si ya sabemos que estos no llegan nunca a 0 o a 1023, respectivamente. De esta manera, se pueden interpretar mejor las lecturas realizadas porque los valores intermedios son más coherentes.

Ejemplo 7.2: El siguiente sketch pretende calibrar un fotorresistor, aunque el procedimiento es generalizable sin apenas cambios a cualquier otro sensor analógico. El circuito necesario es el mismo que el de los ejemplos anteriores: un fotorresistor conectado a un pin de entrada analógico (supondremos el 0) acompañado de una resistencia "pull-down" de 10 KΩ, y un LED conectado a un pin de salida PWM (supondremos el 5) acompañado de su divisor de tensión de 220 Ω correspondiente. El sketch lo que hace básicamente es leer durante sus primeros cinco segundos de ejecución una serie de valores del sensor analógico para establecer cuál será su lectura con valor mínimo y cuál será la que tenga el valor máximo. Es evidente que durante esos cinco segundos, deberemos someter el sensor a ambas circunstancias extremas para ver cómo reacciona (en caso de un fotorresistor, iluminándolo con la máxima luz prevista en el proyecto y con la mínima). Una vez realizada la calibración, el resto del código es muy parecido al visto anteriormente: la clave está en la función *map(),* primero porque realiza ella misma el mapeo en rangos invertidos (por lo ya explicado de iluminar el LED cuando se detecte poca luz), pero sobre todo porque el mapeo lo establece en el rango de valores calibrados, no los típicos 0 y 1023.

```
int valorcds = 0;
int sensorMin = 1023; //Irá disminuyendo
int sensorMax = 0;    //Irá aumentando
void setup() {
     //Calibramos durante los primeros 5 segundos de programa
     while (millis() < 5000) {
           valorcds = analogRead(0);
     /*Si se lee un valor mayor que el actual máximo,
       lo guardo como el nuevo valor máximo */
           if (valorcds > sensorMax) {
                sensorMax = valorcds;
           }
     /*Si se lee un valor menor que el actual mínimo,
       lo guardo como el nuevo valor mínimo */
           if (valorcds < sensorMin) {
```

```
                sensorMin = valorcds;
            }
        }
}
void loop() {
            valorcds = analogRead(0);
/*Aplico la calibración a la lectura recién leída a la transformación
que ha de sufrir valorcds para ser usada en analogWrite() */
        valorcds = map(valorcds, sensorMax, sensorMin, 0, 255);
/*En el caso de que la lectura recién leída caiga fuera del rango
establecido durante la calibración…*/
            valorcds = constrain(valorcds, 0, 255);
            //Ilumino el LED usando el nuevo valor calibrado
            analogWrite(5, valorcds);
}
```

Ejemplo 7.3: Si tenemos el circuito mostrado en la figura siguiente (es decir, un LDR con resistencia "pull-down" de 10 KΩ y un zumbador con divisor de tensión de unos 100 Ω, por ejemplo), podemos hacer que según sea la luz detectada por el LDR, la frecuencia del sonido emitido por el zumbador vaya cambiando.

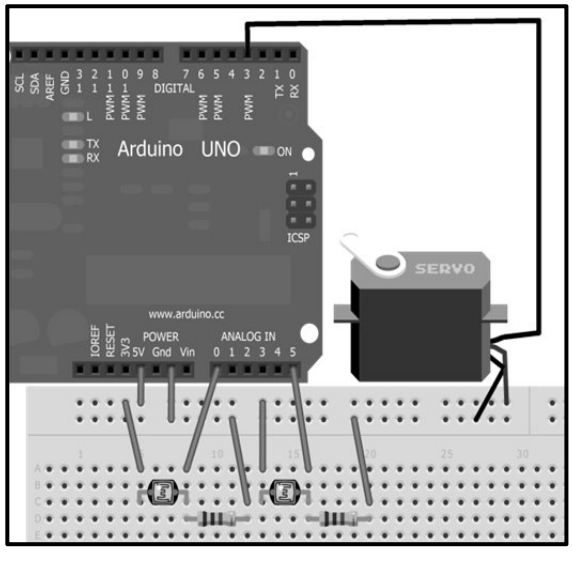


El código es este: simplemente realiza la lectura del LDR, la mapea a un rango de valores audibles y la envía al zumbador. Se debería de calibrar primero el LDR para ajustar el mapeo convenientemente, ya que los valores de *map()* del sketch son solamente orientativos.

```
void setup() {}
void loop() {
     int lectura;
     int sonido;
     lectura = analogRead(0);
/*En este caso, la calibración del LDR da valores entre 400 y 1000,
pero esto puede variar. El rango de salida son, en Hz, las
frecuencias mínimas y máximas del sonido que queremos emitir*/
     sonido = map(lectura, 400, 1000, 120, 1500);
     tone(9, sonido, 10);
     delay(1); //Por estabilidad entre lecturas
}
```

No sería demasiado difícil sustituir en el circuito anterior el LDR por un potenciómetro. De hecho, esta misma variante ya se vio en el apartado relativo al sonido del capítulo 6.

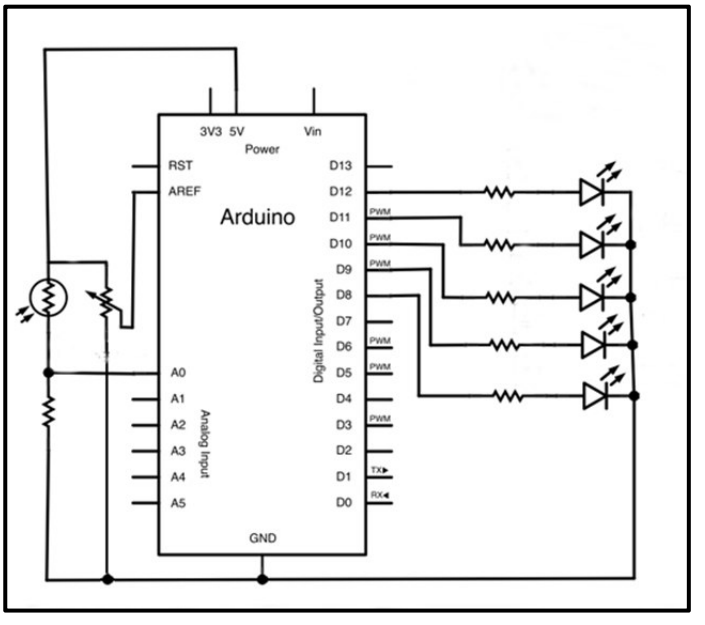
Ejemplo 7.4: Otro ejemplo práctico con LDRs es el diseño de un dispositivo capaz de orientarse hacia el punto más oscuro de su entorno, gracias a los valores obtenidos por dos sensores de luminosidad estratégicamente situados. La idea es comparar la lectura de ambos sensores y orientar un servomotor en un sentido u en otro según si la lectura de un sensor es mayor o menor que la del otro. El montaje sería algo parecido al siguiente:



Y el código, este:

```
#include <Servo.h>
int valorLDRderecha = 0;
int valorLDRizquierda = 0;
int angulo = 0;
Servo miservo;
void setup() {
   miservo.attach(3);
}
void loop() {
   valorLDRizquierda = analogRead(0);
   valorLDRderecha = analogRead(5);
   if (valorLDRderecha < valorLDRizquierda) {
         angulo = angulo - 10;
         if (angulo < 0) {angulo = 0;}
   } else {
         angulo = angulo + 10;
         if (angulo > 179) {angulo = 179;}
   }
   miservo.write(angulo);
}
```

Ejemplo 7.5: Como último ejemplo de demostración práctica del uso de LDRs, vamos a diseñar el siguiente circuito:



Un valor adecuado para los divisores de tensión de los LEDs puede ser 220 Ω. Un valor adecuado para la resistencia "pull-down" asociada al LDR puede ser de 1 KΩ. Un valor adecuado para la resistencia máxima del potenciómetro puede ser 10KΩ. El LDR, tal como se puede ver, está conectado al pin de entrada analógica nº 0.

La idea es iluminar nuestro entorno con LEDs a medida que este vaya oscureciendo. Pero en vez de utilizar un solo LED regulado analógicamente como vimos en uno de los ejemplos anteriores, ahora utilizaremos varios LEDs controlados por señales digitales, de manera que se iluminen más LEDs a medida que se vaya detectando más oscuridad.

La gran novedad de este circuito es el uso de un voltaje de referencia controlado por un potenciómetro. La razón de haber incluido este elemento es la siguiente: ya sabemos que la lectura obtenida del fotorresistor puede variar desde 0 (correspondiente a 0 V, cuando el ambiente está completamente a oscuras) hasta 1023 (correspondiente a 5 V, cuando el ambiente está completamente iluminado). Pero estos dos casos extremos pueden ser difíciles de conseguir, por lo que quizás solo vayamos a trabajar en un rango de 500 o 600 valores intermedios, despreciando así mucha resolución. Para solucionar esto y para permitir además que nuestro circuito se adapte a entornos con diferentes extremos de iluminación sin tener que cambiar cada vez los valores umbral escritos en nuestro código, se ha optado por utilizar un voltaje de referencia ajustable. Este voltaje marca el punto a partir del cual, si decrece la luz ambiente, empiezan a decrecer los valores obtenidos en el pin analógico de entrada. Si esta referencia es baja, los LEDs empezarán a iluminarse con poca luz ambiente pero serán más sensibles a variaciones leves de iluminación; si esta referencia es alta, los LEDs empezarán a iluminarse con mucha luz ambiente pero esta deberá variar mucho para modificar el número de LEDs iluminados. En otras palabras: el potenciómetro sirve para fijar el umbral de luz mínima, a partir del cual, comenzará a funcionar nuestro circuito de luz artificial.

Esto es así porque variando la señal de referencia, le estamos diciendo que el rango de 1024 valores estén ubicados entre 0 V y una determinada tensión máxima que será una u otra según las circunstancias. En nuestro sketch hemos dividido el rango 0-1024 en cinco secciones para que se activen progresivamente cada uno de los cinco LEDs. Si nuestro entorno ofrece una variación muy baja de iluminación con el que jugar, podemos ajustar la tensión de referencia a un valor bajo (por ejemplo, 1 V) y así nos seguirá distribuyendo proporcionalmente la activación de las salidas, con lo que conseguiremos una mayor sensibilidad. El "precio a pagar" es el no activar el circuito hasta que la entrada analógica no reciba esos 1 V, ya que mientras reciba un voltaje mayor, el conversor analógico-digital estará saturado.

Con este truco, pues, no nos será necesario realizar ningún cambio en el código si cambiamos a un entorno con cambios en la iluminación más o menos extremos: tan solo variando la posición del potenciómetro, el rango 0-1024 automáticamente se adaptará a las nuevas circunstancias lumínicas externas. El código es el siguiente:

```
int valorLDR = 0;
byte i;
void setup() {
  //Utilizaremos 5 LEDs, conectados a los pines 8,9,10,11 y 12
  for (i=8;i<=12;i++) { pinMode(i,OUTPUT); }
  analogReference(EXTERNAL);
}
void loop() {
  valorLDR = analogRead(0);
  /*Si el entorno está muy iluminado (según la referencia marcada
   por el potenciómetro), no se enciende ningún LED */
  if(valorLDR >= 1023){
        for (i=8;i<=12;i++){ digitalWrite(i,LOW); }
  //Si está algo menos, se enciende un LED
  } else if(valorLDR >= 823){
        digitalWrite(8, HIGH);
        for (i=9;i<=12;i++){ digitalWrite(i,LOW); }
  //Si está algo menos, se encienden dos LEDs
  } else if(valorLDR >= 623){
        for (i=8;i<=9;i++) { digitalWrite(i,HIGH); }
        for (i=10;i<=12;i++) { digitalWrite(i,LOW); }
  //Si está algo menos, se encienden tres LEDs
  } else if(valorLDR >= 423){
        for (i=8;i<=10;i++){ digitalWrite(i,HIGH); }
        for (i=11;i<=12;i++) { digitalWrite(i,LOW); }
  //Si está algo menos, se encienden cuatro LEDs
  } else if(valorLDR >= 223){
        for (i=8;i<=11;i++){ digitalWrite(i,HIGH); }
        digitalWrite(12,LOW);
  //Si el entorno está muy oscuro, se encienden los cinco LEDs
  } else {
        for (i=8;i<=12;i++){ digitalWrite(i,HIGH); }
  }
}
```

El sensor digital TSL2561

Además de los fotorresistores (que son sensores analógicos), también existen sensores de luz que son digitales, como por ejemplo el chip TSL2561 que Adafruit distribuye sobre una cómoda plaquita breakout. Los sensores digitales son más precisos que los fotorresistores (ya que permiten lecturas exactas, medidas en unidades lux) y su sensibilidad puede ser configurada dependiendo de la intensidad de luz con la que se trabaje en ese momento (intensidad cuyo rango admitido es además mucho más amplio que el de los fotorresistores). Además, el TSL2561 concretamente detecta, además de todo el espectro visible, también la luz infrarroja; pudiéndose configurar para medir separadamente la luz visible, la luz infrarroja o ambos.

Este chip se alimenta con un voltaje de entre 2,7 V y 3,6 V y funciona como mucho a 0,5 mA, por lo que es ideal para sistemas de bajo consumo. Su sistema de comunicación con el exterior es el protocolo I^2C, por lo que en la plaquita breakout en la que se comercializa, además de los contactos de alimentación y tierra, aparecen los contactos "SDA" (a conectar al pin analógico nº 4 de Arduino) y "SCL" (a conectar al pin analógico nº 5 de Arduino).

La exactitud de este sensor tiene un "precio", y es la dificultad de su uso: además de la propia complejidad interna que aporta del protocolo I^2C, para deducir la cantidad de luminosidad exacta leída por el chip se han de utilizar muchos cálculos matemáticos poco intuitivos. Afortunadamente, Adafruit ofrece una librería Arduino propia que facilita mucho la obtención e interpretación de datos, descargable desde https://github.com/adafruit/TSL2561-Arduino-Library. Por falta de espacio no podemos profundizar en el uso de esta librería, pero si se quiere empezar a aprender a usarla, después de instalarla como cualquier librería Arduino, recomiendo observar el código comentado de los sketches de ejemplo que vienen junto con la librería.

El sensor analógico TEMT6000

También existen chips sensores de luz con comportamiento analógico. Un ejemplo es el TEMT6000, distribuido en forma de plaquita breakout por Sparkfun (producto nº 8688). Este sensor tiene la ventaja de ser mucho más preciso que un fotorresistor (reacciona mejor a cambios de iluminación en un rango mayor) sin añadir más complejidad a nuestros circuitos. Está adaptado a la sensibilidad del ojo humano, por lo que no reacciona ante la luz infrarroja o ultravioleta.

Las conexiones de la plaquita breakout distribuida por Sparkfun son muy simples: el conector "VCC" puede ir conectado directamente al pin "5V" de la placa Arduino, el conector "GND" a tierra" y el conector "SIG" a cualquier entrada analógica de Arduino. Cuanto más voltaje leamos del sensor, más iluminado estará el entorno.

Ejemplo 7.6: Su programación también es muy simple. He aquí un ejemplo:

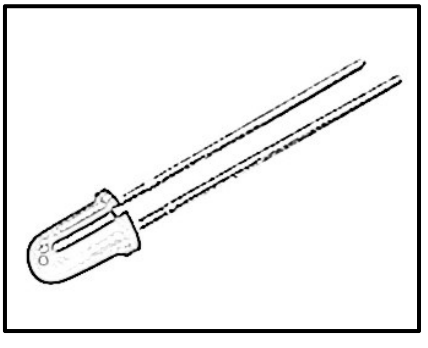
```
int pinsensor = 0; //Entrada analógica donde está conectado el sensor
void setup() {
  Serial.begin(9600);
}
void loop() {
  int lectura;
  lectura = analogRead(pinsensor);
  Serial.println(lectura); //0=muy oscuro; 1023=muy iluminado
  delay(100);
}
```

Otra plaquita breakout que incluye el mismo chip TEMT6000 es la distribuida por Freetronics bajo el nombre de "Light sensor module". Igualmente, dispone de tres conectores (VCC, GND y OUT) para comunicarse con nuestra placa Arduino y se programa exactamente igual.

Otra plaquita breakout muy parecida (con también los tres conectores y programable de la misma forma) es la llamada "AMBI Light sensor" de Modern Device, pero esta incluye otro chip, el GA1A1S201WP.

SENSORES DE LUZ INFRARROJA

Fotodiodos y fototransistores



Un fotodiodo es un dispositivo que, cuando es excitado por la luz, produce en el circuito una circulación de corriente proporcional (y medible). De esta manera, pueden hacerse servir como sensores de luz, aunque, si bien es cierto que existen fotodiodos especialmente sensibles a la luz visible, la gran mayoría lo son sobre todo a la luz infrarroja. Se pueden adquirir en cualquier distribuidor de componentes

básicos, tales como Mouser o Jameco, por poner un par de ellos. Ejemplos de dispositivos concretos que nos pueden venir bien son (el código es del fabricante) el TEFD4300F, el BPV22F, el BPV10NF o el SFH235FA.

Hay que tener en cuenta que, a pesar de tener un comportamiento en apariencia similar a los LDRs, una diferencia muy importante respecto estos (además de la sensibilidad a otras longitudes de onda) es el tiempo de respuesta a los cambios de oscuridad a iluminación, y viceversa, que en los fotodiodos es mucho menor.

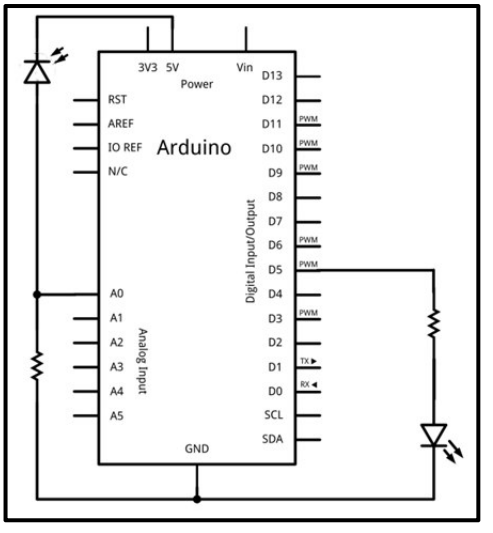
Igual que los diodos estándar, los fotodiodos poseen un ánodo y un cátodo, pero atención, para que funcione como deseamos, un fotodiodo siempre se ha de conectar al circuito en polaridad inversa. Eso sí, igual que ocurre con los diodos comunes, normalmente el ánodo es más largo que el cátodo (en caso de ser de igual longitud, el cátodo deberá estar marcado de alguna forma).

Su funcionamiento interno es el siguiente: cuando el fotodiodo está polarizado en directa, la luz que incide sobre él no tiene un efecto apreciable y por tanto el dispositivo se comporta como un diodo común. Cuando está polarizado en inversa y no le llega ninguna radiación luminosa, también se comporta como un diodo normal ya que los electrones que fluyen por el circuito no tienen energía suficiente para atravesarlo, con lo que el circuito permanece abierto. Pero en el momento en el que el fotodiodo recibe una radiación luminosa dentro de un rango de longitud de onda adecuado, los electrones reciben suficiente energía para poder "saltar" la barrera del fotodiodo en inversa y continuar su camino.

Ejemplo 7.7: Para probar su comportamiento, podemos utilizar un circuito como el de la página siguiente. Este circuito es idéntico al que ya vimos con los LDRs, sustituyendo estos por un fotodiodo (el cual se identifica por un nuevo símbolo que no habíamos visto hasta ahora). El valor de su divisor de tensión dependerá de la cantidad de luz (infrarroja) presente en el ambiente: resistencias mayores mejoran la sensibilidad cuando solo hay una fuente de luz y resistencias menores la mejoran cuando hay muchas (el propio sol o las lámparas son fuentes de infrarrojos); un valor de 100 KΩ puede ir bien para empezar. Fijémonos además que es el cátodo del fotodiodo (el terminal más corto, recordemos) el que se conecta a la alimentación.

El funcionamiento de este circuito es el siguiente: mientras el fotodiodo no detecte luz infrarroja, por la entrada analógica de la placa Arduino (en este caso la número 0) se medirá un voltaje de 0 V porque el circuito actuará como un circuito abierto. A medida que vaya aumentando la intensidad lumínica sobre el fotodiodo, aumentará la cantidad de electrones que lo traspasa (es decir, la intensidad de

corriente). Esto implica que, al ser la resistencia "pull-down" fija, por la Ley de Ohm el voltaje medido en el pin de entrada analógico también aumentará, hasta llegar un momento en el cual al recibir mucha luz el fotodiodo no cause apenas resistencia al paso de los electrones y por tanto la placa Arduino lea un voltaje máximo de 5 V.



Hemos añadido un LED conectado al pin de salida PWM nº 5 tal como hicimos cuando vimos los LDRs para tener una manera visible (nunca mejor dicho) de detectar la incidencia de luz infrarroja sobre el fotodiodo. Tal como se puede observar en el código utilizado (mostrado a continuación), hemos hecho depender la intensidad del brillo del LED de la cantidad de luz infrarroja detectada por el fotodiodo: cuanta más radiación infrarroja se reciba, más iluminado estará el LED.

```
int valorfotodio;   //Valor obtenido del fotodiodo
int brilloLED;      //Valor enviado al LED
void setup(void) {
      Serial.begin(9600);
}
void loop(void) {
      valorfotodio = analogRead(0);
      Serial.println(valorfotodio);
      /*El brillo del LED es proporcional a la
       cantidad de luz infrarroja recibida */
      brilloLED = map(valorfotodio, 0, 1023, 0, 255);
      analogWrite(5, brilloLED);
      delay(100); //Para que se pueda ver el nuevo brillo
}
```

Podemos jugar a aproximar el fotodiodo del ejemplo anterior a diferentes fuentes de luz infrarroja (prácticamente cualquier objeto relativamente caliente funciona como tal), pero lo más conveniente es añadir al circuito anterior una fuente de infrarrojos algo más manejable. Lo más habitual para ello es utilizar un LED emisor de infrarrojos. Simplemente conectando su ánodo al pin 5 V de la placa Arduino y su cátodo a tierra (a través de un divisor de tensión, de 220 Ω está bien) ya tendríamos una fuente constante y estable de radiación infrarroja. Si quisiéramos una fuente de emisión controlable, no tendríamos más que conectarlo como cualquier otro LED: su ánodo a un pin de salida de la placa Arduino (digital o PWM según lo que queramos) y su cátodo a tierra (a través de un divisor de tensión también, lógicamente). Sparkfun distribuye un LED emisor de infrarrojos de 850 nm con nº de producto 9469 y otro de 950 nm con nº de producto 9349. Adafruit distribuye un LED de 940 nm con nº de producto 387.

Otro tipo de sensores de luz además de los fotodiodos son los llamados fototransistores, es decir, transistores sensibles a la luz (también normalmente infrarroja). Su funcionamiento es el siguiente: al incidir luz sobre su base, en ella se genera una corriente que lleva al transistor a un estado de conducción. Por tanto, un fototransistor es igual a un transistor común con la única diferencia de que la corriente de base I_b es dependiente de la luz recibida. De hecho, existen fototransistores que pueden trabajar de las dos formas: o bien como fototransistores o bien como transistores comunes con una corriente de base I_b concreta dada.

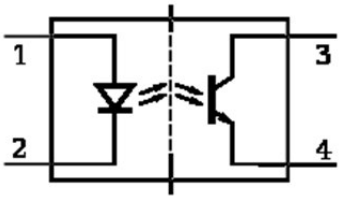
El fototransistor es mucho más sensible que el fotodiodo (por el efecto de ganancia del propio transistor), ya que las corrientes que se pueden obtener con un fotodiodo son realmente limitadas. De hecho, se puede entender un fototransistor como una combinación de fotodiodo y amplificador, por lo que, en realidad, si quisiéramos construir un fototransistor casero, bastaría con agregar a un transistor común un fotodiodo, conectando el cátodo del fotodiodo al colector del transistor y el ánodo a la base. En esta configuración, la corriente que entrega el fotodiodo (que circularía hacia la base del transistor) se amplificaría β veces.

En muchos circuitos podemos encontrar un fototransistor a poca distancia de un LED emisor de infrarrojos de una longitud de onda compatible. Esta pareja de componentes es útil para detectar la interposición entre ellos de un obstáculo (debido a la interrupción del haz de luz) y actuar por tanto como interruptores ópticos. Se pueden utilizar en multitud de aplicaciones, como por ejemplo en detectores del paso de una tarjeta de crédito (en un cajero) o de la introducción del papel (en una impresora) o como tacómetros, entre muchas otras. Un tacómetro es un dispositivo que cuenta las vueltas por minuto que realiza un obstáculo sujeto a

una rueda o aspa que gira (normalmente debido al funcionamiento de un motor); es decir, sirve para medir la velocidad de giro de un objeto.

Podemos adquirir ya de fábrica bajo un encapsulado común la pareja de componentes LED más fototransistor con el nombre genérico de "fotointerruptor". En Sparkfun por ejemplo distribuyen uno con el código nº 9299, el cual consta de dos terminales correspondientes al ánodo y cátodo del LED, y dos terminales correspondientes al colector y emisor de un fototransistor NPN. Por lo general, querremos conectar los terminales del LED a un circuito cerrado continuamente alimentado (ánodo a fuente, cátodo a tierra), el terminal del colector del fotointerruptor a una fuente de alimentación y el terminal del emisor del fotointerruptor a una entrada digital de nuestra placa Arduino, para poder detectar así la aparición de corriente cuando se reciba iluminación. Por otro lado, tanto esta entrada de la placa Arduino como el emisor deberían estar conectados a tierra a través de la misma resistencia "pull-down", para obtener unas lecturas más estables (un valor típico de 10 KΩ puede funcionar, pero dependiendo del circuito tal vez se necesiten valores mayores).

También podemos encontrar la pareja LED infrarrojo más fototransistor en unos componentes llamados "optoacopladores" o "optoaislador". Su representación esquemática suele ser así:



A grandes rasgos un optoacoplador actúa como un circuito cerrado cuando llega luz desde el LED a la base del transistor y abierto cuando el LED está apagado. Su principal función es controlar y a la vez aislar dos partes de un circuito que normalmente trabajan a tensiones diferentes (tal como lo haría un transistor común, pero de una forma algo más segura). Físicamente suelen ser chips que ofrecen como mínimo cuatro patillas (igual que los fotointerruptores): dos correspondientes a los terminales del LED y dos correspondientes al colector y emisor del fototransistor (aunque pueden tener una patilla más correspondiente a la base si se permite controlar la intensidad que fluye por esta también de forma estándar). Ejemplos de optoacopladores son el 4N35 o el CNY75, fabricados por varias empresas y disponibles en Mouser, Jameco y similares.

La pareja LED-fototransistor también es útil para detectar objetos situados a pequeñas distancias de ella. Esto lo estudiaremos en el apartado correspondiente a los sensores de distancia.

Control remoto

Una utilidad práctica inmediata de una pareja emisor-receptor de infrarrojos (como un LED y un fotodiodo/fototransistor) ubicados a una cierta distancia es el envío de "mensajes" entre ellos. Es decir, ya que la luz infrarroja no es visible (y por tanto, no "molesta"), se pueden emitir pulsos de determinada duración y/o frecuencia que pueden ser recibidos y procesados a varios metros de distancia sin que "se note". El dispositivo que los reciba deberá entonces estar programado para realizar diferentes acciones según el tipo de pulso leído. Sparkfun por ejemplo vende un "pack" con ambos componentes, con número de producto 241.

De hecho, cualquier dispositivo que funcione con un "mando a distancia" funciona de forma parecida porque en su parte frontal he de tener un sensor de infrarrojos (también llamados sensores "IR", del inglés "infra-red") que reciba las señales infrarrojas emitidas por el mando. Y lo que hay dentro de este es básicamente un LED que emite pulsos de luz infrarroja siguiendo un determinado patrón que señala al dispositivo la orden a realizar: existe un código de parpadeos para encender el televisor, otro para cambiar de canal, etc.

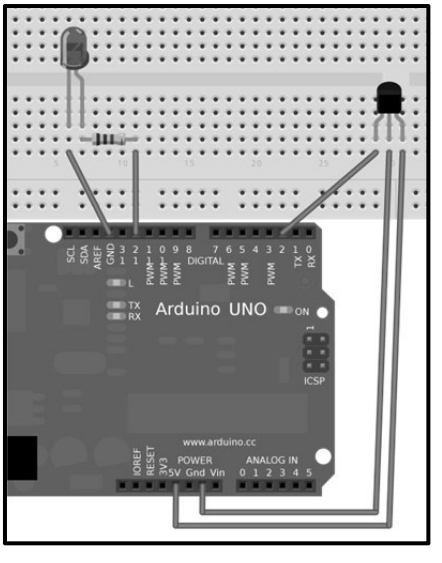
En el párrafo anterior hablamos de "sensores IR" y no de fotodiodos/fototransistores porque los primeros son algo más sofisticados. Concretamente, los sensores IR no detectan cualquier luz infrarroja, sino solo aquella que (gracias a que incorporan un filtro pasa-banda interno y un circuito demodulador) está modulada mediante una onda portadora de una frecuencia de 38 KHz $\pm$ 3 KHz. Esto básicamente quiere decir solo las señales cuya información es transportada mediante una onda de 38 KHz serán leídas. Esto es así para evitar que los sensores IR se "vuelvan locos" al recibir la luz infrarroja que existen proveniente de todos lados (sol, luz eléctrica…): de esta forma solo responden a emisiones muy concretas ya estandarizadas.

Otra diferencia con los fotodiodos/fototransistores es que los sensores IR ofrecen una respuesta binaria: si detectan una señal IR de 38 KHz el valor que se puede leer de ellos en la mayoría de los casos es LOW (0 V), y si no detectan nada, su lectura ofrece un valor HIGH (5 V). Este comportamiento es lo que se suele llamar "activo a bajo, o "low-active".

Ejemplos de sensores IR pueden ser el TSOP32838 de Vishay (producto nº 157 de Adafruit o nº 10266 de Sparkfun) o el GP1UX311QS. Como características más destacadas tienen que su rango de sensibilidad está entre longitudes de onda de 800 nm a 1100 nm con un máximo de respuesta en 940 nm y que necesitan alrededor de 5 V y 3 mA para funcionar. Sparkfun también comercializa (con código de producto 8554) una plaquita breakout muy simple con otro sensor IR, el chip TSOP85.

El chip TSOP32838 ofrece tres patillas: teniendo de cara su dorso semiesférico, la patilla de más a la izquierda es la salida digital que ofrece el sensor, la patilla central ha de estar conectado a tierra y la patilla de más a la derecha ha de conectarse a la alimentación (de entre 2,5 V y 5,5 V).

Ejemplo 7.8: Para probar su funcionamiento, podríamos diseñar un circuito como el siguiente. El divisor de tensión para el LED puede ser de entre 200 y 1000 ohmios También se podría haber añadido un divisor de tensión de entre 100 y 900 Ω en serie a la patilla de alimentación del sensor y además, un condensador by-pass de 1 µF conectado entre esta y tierra para estabilizar el comportamiento del sensor, pero no es imprescindible.



La idea es encender durante unos instantes el LED cuando el sensor IR detecte una señal infrarroja. Pero atención, no vale cualquier señal infrarroja, sino solamente aquella modulada a 38 KHz. Por tanto, para probar este circuito, no podemos usar un LED infrarrojo cualquiera: deberemos usar algún mando de control remoto que tengamos a mano (de un televisor, un reproductor de DVD, un

computador, etc.). Una vez cargado en la placa Arduino el sketch presentado a continuación, si apuntamos con ese mando al sensor IR y pulsamos algunos de sus botones, deberíamos ver que el LED se ilumina. De esta manera, estaremos utilizando el sensor IR como si fuera un interruptor, el cual ilumina el LED mientras detecta esa señal y lo apaga cuando ya no la detecta.

El sketch necesario para que el circuito anterior funcione como deseemos es el siguiente (la salida del sensor está conectada a la entrada digital nº 2 de la placa Arduino y el LED está conectado a su salida digital nº 12):

```
int irPin=2;
int ledPin=12;
void setup() {
     pinMode(irPin,INPUT);
     pinMode(ledPin,OUTPUT);
}
void loop() {
/*Ya que la señal emitida por el sensor es normalmente HIGH, cuando
se pulsa el botón de un mando, esta cambia a LOW. Lo que hace la
función pulseIn()es pausar el sketch hasta que se detecte una señal
LOW, cuya duración en realidad no nos interesa pero lógicamente
siempre será mayor de cero. Por tanto, si se cumple la condición del
if significa que se ha pulsado un botón de un mando */
        if(pulseIn(irPin,LOW) > 0) {
/*Es necesario esperarse un tiempo determinado (que dependerá del
modelo concreto de mando a distancia) tras la detección de la primera
señal LOW debido a que cada pulsación de botón produce múltiples
oscilaciones entre valores HIGH y LOW. Aunque físicamente no tiene
nada que ver, podemos entender esta espera como si fuera una manera
de evitar un "bounce" (fenómeno estudiado cuando tratamos los
pulsadores). Una vez transcurrido ese tiempo de espera, la señal del
sensor debería haber vuelto a su estado de reposo (valor HIGH).*/
             delay(100);
/*Mantenemos encendido el LED durante unos cuantos milisegundos.
Durante este tiempo el sketch no podrá detectar otras pulsaciones
provenientes del control remoto. También podríamos haber enviado un
mensaje al "Serial monitor" notificando la pulsación. */
             digitalWrite(ledPin,HIGH);
             delay(200);
             digitalWrite(ledPin,LOW);
        }
}
```

Sin embargo, más allá de aquí no podremos hacer gran cosa con nuestro Arduino si no conocemos los patrones concretos de pulsos IR emitidos por el mando a distancia utilizado. Es decir, para evitar que un mando a distancia Sony pueda cambiar los canales de un televisor Philips (por ejemplo), cada marca utiliza una codificación distinta para sus señales, aunque todas estén moduladas a 38 KHz. Es decir, cada marca emite diferentes duraciones, secuencias y combinaciones de señales HIGH y LOW, para evitar así posibles interferencias. Por lo tanto, si queremos controlar una placa Arduino mediante un mando a distancia de algún aparato que tengamos a mano de una determinada marca, deberíamos conocer primero el protocolo utilizado por este para que nuestra placa lo procese adecuadamente. Y también a la inversa: si queremos controlar remotamente un electrodoméstico de una determinada marca mediante una placa Arduino (que haría en este caso de mando a distancia) deberíamos conocer el protocolo de comunicación reconocido por ese electrodoméstico. Desgraciadamente, hay casi tantos patrones de pulsos IR como aparatos, pero tenemos varias soluciones para ello.

La primera sería adquirir una pareja de mando a distancia más sensor IR que fueran compatibles de fábrica, sin tenernos que preocupar de su protocolo interno de comunicación. En este sentido, podemos adquirir el “IR Kit” de DFRobot (producto nº DFR0107), que incluye un mando a distancia y un sensor IR compatible (con nº de producto DFR0094 si se adquiere por separado). Este sensor viene dentro de una plaquita breakout con tres pines (5V, GND y señal digital) que podemos conectar muy fácilmente a nuestra placa Arduino o a una breadboard. Lo interesante es que este kit viene acompañado de una librería descargable de la web del producto, que permite a nuestra placa Arduino interpretar de una forma adecuada los datos recibidos por el pin de señal digital del sensor IR.

Otra alternativa similar es la adquisición, en Adafruit, de la pareja formada por el mando a distancia con nº de producto 389 (que sí se sabe que utiliza el patrón de señales NEC) y el sensor IR TSOP38238, utilizado en el circuito de ejemplo anterior. Si se utiliza con el mando a distancia indicado, este sensor puede ser controlado mediante una librería propia de Adafruit, la “Adafruit NEC Remote Control Library”, descargable de https://github.com/adafruit/Adafruit-NEC-remote-control-library

Sparkfun por su parte distribuye un kit completo con nº de producto 10783, el cual incluye un mando a distancia (con nº de producto 10280 si se adquiere por separado), un par de sensores IR también de modelo TSOP38238 y además un par de LEDs IR también. Para poder procesar correctamente las señales emitidas por el mando a distancia distribuido en este kit (y solo ese mando en particular), no existe

una librería oficial, pero en la página del producto se nos ofrece un código Arduino de ejemplo donde se hace uso de una función propia muy sencilla de utilizar llamada *getIRkey()*, la cual devuelve el código numérico correspondiente al botón pulsado en ese momento (o 0 si no hay pulsado ninguno). Por otro lado, junto con el mismo mando a distancia, también se puede utilizar la plaquita breakout con nº de producto 8554 mencionada unos párrafos más arriba; en este caso, sí se puede utilizar una librería, disponible en https://github.com/konstantint/ArduinoSparkfunIRReceiver.

Otro enfoque diferente a las soluciones anteriores para conseguir el control remoto de nuestro proyecto es no utilizar ningún mando a distancia, sino construírselo uno mismo. Es más fácil de lo que puede parecer a priori, ya que para emitir una señal infrarroja modulada a 38 KHz lo único que se necesita es un simple LEDs infrarrojo. Eso sí, es necesario controlarlo de tal forma que pueda encenderse y apagarse a 38 KHz el número de veces que sea necesario. Es decir, si queremos emitir con un LED infrarrojo una señal HIGH que dure (por ejemplo) un segundo y el LED simplemente permanece encendido durante ese tiempo y ya está, el sensor IR no detectará nada porque la señal no está modulada. Para poder modular esa señal HIGH el LED deberá encenderse y apagarse 38000 veces durante ese segundo. De esta manera, el sensor IR reconocerá ese patrón de frecuencia e interpretará que le ha llegado un pulso. Si quisiéramos emitir una señal LOW, con mantener el LED apagado ya valdría.

Pensando en esta aplicación, Sparkfun incluso comercializa una plaquita breakout (con código 10732) que incluye un LED infrarrojo y un transistor, de forma que la luz emitida tenga una mayor intensidad y sea más fácilmente captarla a mayores distancias. Tan solo es necesario enviar la señal modulada a la base del transistor (correspondiente al conector de la plaquita etiquetado como “CTL”), además de alimentar dicha plaquita y conectarla a tierra.

Ejemplo 7.9: A continuación, se muestra un código de ejemplo de la emisión de un patrón ficticio de pulsos modulados a 38 KHz, repetido infinitamente cada segundo. Este patrón está formado por pulsos de diferente duración, separados por intervalos de tiempo también de distinta duración. Así se pretende simular el patrón de un producto comercial cualquiera, en el cual los pulsos suelen ser de distinta longitud y aparecer en intervalos diferentes. Supondremos que el LED está conectado directamente al pin de salida digital nº 8 de la placa Arduino (y a tierra, a través de un divisor de tensión, como siempre).

```
void setup(){
    pinMode(8, OUTPUT);
}
void loop(){
    pulseIR(2080);//Emito un pulso modulado durante 2080 µs
    delay(27);    //No emito nada durante 27 milisegundos
    pulseIR(440); //Emito un segundo pulso modulado durante 440 µs
    delayMicroseconds(1500);//No emito nada durante 1500 µs
    pulseIR(460);
    delayMicroseconds(3440);
    pulseIR(480);
    delay(1000);
}
/*Esta función envía un pulso modulado a 38KHz
de una duración determinada por su parámetro.*/
void pulseIR(long microsecs) {
    cli();
    /*Voy pasando los periodos que "caben"
    en el intervalo de tiempo especificado */
    while (microsecs > 0) {
/*Una frecuencia de 38KHz equivale a un periodo de 26 microsegundos:
durante 13 microseconds (semiperiodo) la señal ha de ser HIGH y
durante 13 microseconds será LOW */
        digitalWrite(8,HIGH);
/*Es conocido que la ejecución de la función digitalWrite tarda sobre
3 microsegundos, por lo que tras ella espero 10 microsegundos más */
        delayMicroseconds(10);
        digitalWrite(8, LOW);        //Se tarda también 3 µs
        delayMicroseconds(10);       //Me espero 10 µs más
        microsecs = microsecs - 26;  //Ya se ha cumplido un periodo
    }
    sei();
}
```

En el código anterior aparecen dos funciones del lenguaje Arduino que no habíamos visto hasta ahora: **cli()** y **sei()**. La función *cli()* sirve para desactivar ciertas tareas que la placa Arduino siempre está realizando constantemente en segundo plano (tales como escuchar posibles datos recibidos por el puerto serie, llevar la cuenta del tiempo, etc.). Cuando se están tratando señales de alta velocidad, es muy recomendable mantener la placa Arduino lo más inactiva posible para poder realizar un seguimiento preciso y limpio. La segunda sirve para volver activar estas tareas.

Ejemplo 7.10: No serviría de nada tener un emisor de pulsos modulados si no tenemos un sensor que los reconozca. Para ello, utilizaremos el sensor IR TSOP38238 (con su salida conectada al pin nº 2 de la placa Arduino) y el siguiente código, el cual mostrará por el Serial monitor la duración de los pulsos modulados recibidos (valores HIGH), y el tiempo existentes entre ellos (en realidad, la duración de los pulsos LOW). Con este código, de hecho, podremos conocer los patrones de cualquier mando a distancia comercial y por tanto, podremos construirnos un clon:

```
//Pin de la placa Arduino donde está conectada la salida del sensor
const byte irPin=2;
//Duración máxima que reconoceremos para un pulso (¡65 ms es mucho!)
const int MAXPULSO = 65000;
/*Resolución temporal que utilizaremos. Cuanto menor sea su valor,
más precisa será la lectura de la señal, pero si es demasiado pequeño
puede haber problemas de sincronización. El valor de 20 suele ser el
adecuado en la mayoría de los casos*/
const byte RESOLUCION = 20;
/*Guardaremos 100 parejas formadas por un pulso (valor HIGH) y su
valor LOW siguiente. Para ello hacemos uso de un array bidimensional:
el valor HIGH de la primera pareja se guardará en el elemento
pulsos[0][0], el valor LOW de la primera pareja se guardará en el
elemento pulsos[0][1], el valor HIGH de la segunda pareja se guardará
en el elemento pulsos[1][0], el valor LOW de la segunda pareja se
guardará en el elemento pulsos[1][1], el valor HIGH de la tercera
pareja se guardará en el elemento pulsos[2][0], y así. */
word pulsos[100][2];
//Índice del pulso que estamos guardando en ese momento en el array
byte pulsoactual = 0;
setup(){
       Serial.begin(9600);
}
loop(){
      word contadortiempohigh =0;
      word contadortiempolow  =0;
/*La siguiente línea es igual a "while(digitalRead(irPin)){", pero no
la escribimos así porque la función digitalRead() es demasiado lenta
para poder leer las rápidas variaciones de la señal modulada. Por
tanto, hemos recurrido a código C optimizado para la plataforma AVR,
el cual nos da un acceso más directo al hardware de la placa. No
profundizaremos más en ello. */
      //Mientras se recibe una señal HIGH
       while ((PIND & _BV(irPin))) {
              contadortiempohigh++;
```

```
            //Cuento unos cuantos microsegundos más
            delayMicroseconds(RESOLUCION);
/*Si el pulso es demasiado largo significa que ya se acabó el patrón,
por lo que lo imprimimos, nos preparamos para rellenar otra vez el
array y, mediante la función return, volvemos al inicio de la función
loop()*/
      if ((contadortiempohigh >= MAXPULSO) && (pulsoactual != 0)) {
                  printpulsos();
                  pulsoactual=0;
                  return;
            }
      }
      /*Ya se ha dejado de recibir un pulso HIGH,
       por lo que guardamos el tiempo que ha durado.*/
      pulsos[pulsoactual][0] = contadortiempohigh;
      //Seguimos leyendo el siguiente pulso de la señal por el pin 2
      //Mientras se recibe una señal LOW
      while (!(PIND & _BV(irPin))) {
            contadortiempolow++;
            delayMicroseconds(RESOLUCION);
      if ((contadortiempolow >= MAXPULSO) && (pulsoactual != 0)) {
                  printpulsos();
                  pulsoactual=0;
                  return;
            }
      }
      pulsos[pulsoactual][1] = contadortiempolow;
//Hemos leído una pareja de valores HIGH-LOW con éxito. Continuamos
      pulsoactual++;
}
void printpulsos() {
      byte i;
      Serial.println("OFF \tON");
      for (i = 0; i < pulsoactual; i++) {
            Serial.print(pulsos[i][0] * RESOLUCION);
            Serial.print(" usec, ");
            Serial.print(pulsos[i][1] * RESOLUCION);
            Serial.println(" usec");
      }
}
```

De todas formas, existe una forma mucho más sencilla de poder utilizar señales moduladas: descargar e instalar la librería “Arduino IR”, disponible en la página https://github.com/shirriff/Arduino-IRremote . Con ella no tendremos ni que

adquirir ningún nuevo mando específico (por lo que podremos reciclar alguno que tengamos a mano de un aparato ya inútil) ni tampoco tendremos que investigar su patrón particular para poderlo controlar. Esta librería permite que nuestra placa Arduino pueda tanto enviar como recibir códigos de control remoto en múltiples patrones ya incluidos "de fábrica" (soporta los protocolos NEC, Sony SIRC, Philips RC5 y RC6, y muchos más) e incluso tiene un mecanismo para añadirle más protocolos si es necesario. Con ella, nuestra placa Arduino incluso podría funcionar como un reenviador de códigos, recibiéndolos de un mando a distancia comercial y retransmitiéndolos a otro lugar. Veamos algún ejemplo.

Ejemplo 7.11: Suponiendo que tenemos conectado el sensor IR TSOP38238 (u otro similar) a un pin de entrada digital cualquiera de Arduino, el siguiente código muestra por el "Serial monitor" los comandos recibidos de un mando a distancia cualquiera.

```
#include <IRremote.h>
/*Pin de entrada digital de la placa Arduino donde
 hemos colocado la patilla de señal del receptor */
int pinreceptor = 11;
//Creo un objeto llamado "irrecv" de tipo IRrecv
IRrecv irrecv(pinreceptor);
//Declaro una variable de un tipo especial, "decode_results".
decode_results resultados;
void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn(); //Inicio el receptor
}
void loop() {
/*Miro si se ha detectado algún patrón IR modulado. Si es así, lo leo
y lo guardo enteramente en la variable especial "resultados", en
forma de número hexadecimal*/
  if (irrecv.decode(&resultados) !=0) {
        /*Primero miro qué tipo de patrón comercial es,
        si es que es de alguno reconocido por la librería */
        if (results.decode_type == NEC) {
                    Serial.print("NEC: ");
        } else if (results.decode_type == SONY) {
                    Serial.print("SONY: ");
        } else if (results.decode_type == RC5) {
                    Serial.print("RC5: ");
        } else if (results.decode_type == RC6) {
                    Serial.print("RC6: ");
          } else if (results.decode_type == UNKNOWN) {
                      Serial.print("Desconocido: ");
```

```
        }
      /*Y seguidamente muestro el patrón recibido
      (en formato hexadecimal) por el canal serie */
      Serial.println(resultados.value, HEX);
      /*Una vez el patrón ha sido decodificado, reactivo otra vez la
      escucha para poder detectar el siguiente posible patrón */
      irrecv.resume();
  }
/*Aquí se pueden hacer otras cosas mientras
 se espera a recibir un comando IR*/
}
```

Tal como se expresa en los comentarios del código anterior, la función *decode()* no es bloqueante; esto quiere decir que el sketch puede realizar otras operaciones mientras se está esperando la detección de un patrón nuevo.

Cada botón de un mando a distancia está asociado a un patrón particular (generalmente de 12 a 32 pulsos) identificable mediante un número hexadecimal. Si el botón se mantiene pulsado, este código usualmente es repetido constantemente, aunque existen mandos que solo envían el código una sola vez y utilizan otros métodos para detectar el fin de una pulsación (como marcas de final de pulsación o códigos especiales que funcionan como contadores, etc.).

Ejemplo 7.12A: Una vez conocido el patrón de un mando a distancia, podemos desarrollar sketches que hagan reaccionar a nuestra placa Arduino según el botón que haya sido pulsado. Por ejemplo, en el siguiente sketch podemos observar por el "Serial monitor" el botón pulsado de un mando a distancia de un televisor, concretamente de la marca Sony.

```
#include <IRremote.h>
int pinreceptor = 11;
IRrecv irrecv(pinreceptor);
decode_results resultados;
void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
}
void loop(){
   int i;
   if (irrecv.decode(&resultados)!=0) {
```

```
          accion();
          /*Los mandos Sony envían 3 veces el mismo patrón repetido.
          Este "for" sirve para ignorar el 2° y 3° patrón.*/
          for (i=0; i<2; i++) {
               //Recibo el siguiente patrón y no hago nada
                irrecv.resume();
          }
   }
}
void accion() {
  switch(results.value)  {
    case 0x37EE: Serial.println("Favoritos"); break;
    case 0xA90: Serial.println("Encendido/Apagado"); break;
    case 0x290: Serial.println("Mute"); break;
    case 0x10:  Serial.println("1"); break;
    case 0x810: Serial.println("2"); break;
    case 0x410: Serial.println("3"); break;
    case 0xC10: Serial.println("4"); break;
    case 0x210: Serial.println("5"); break;
    case 0xA10: Serial.println("6"); break;
    case 0x610: Serial.println("7"); break;
    case 0xE10: Serial.println("8"); break;
    case 0x110: Serial.println("9"); break;
    case 0x910: Serial.println("0"); break;
    case 0x490: Serial.println("Aumentar volumen"); break;
    case 0xC90: Serial.println("Disminuir volumen"); break;
    case 0x90:  Serial.println("Aumentar canal"); break;
    case 0x890: Serial.println("Disminuir canal"); break;
    default: Serial.println("Otro botón");
  }
  delay(500);
}
```

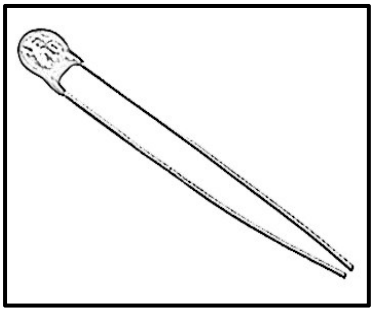
Ejemplo 7.12B: Para hacer lo contrario, es decir, para enviar un patrón comercial determinado mediante un LED infrarrojo conectado a nuestra placa Arduino, sería tan sencillo como ejecutar un sketch similar al siguiente. Mediante este código de ejemplo, nuestra placa Arduino encenderá (o apagará) un televisor que utilice el protocolo de Sony cada vez que reciba cualquier dato a través de su canal serie. Un detalle muy importante a tener en cuenta es que para que el código siguiente funcione, el LED infrarrojo ha de estar conectado obligatoriamente al pin digital de salida nº 3 de nuestra placa Arduino.

```
#include <IRremote.h>
IRsend irsend;
void setup(){
  Serial.begin(9600);
}
void loop() {
  if (Serial.read() != -1) {
  /*El patrón se ha de enviar tres veces porque
  el protocolo Sony lo establece así */
       for (int i = 0; i < 3; i++) {
/*La función sendSony() envía un patrón especificado como primer
parámetro perteneciente a ese protocolo (en este caso, el de
encendido/apagado). El segundo parámetro indica el número de bits que
componen ese patrón. Este número es simplemente el número de dígitos
hexadecimales de los que se compone el patrón multiplicado por 4.*/
              irsend.sendSony(0xA90, 12);
              delay(100);
         }
    }
}
```

Existen otras funciones predefinidas diferentes de *sendSony()* para poder enviar otro tipo de protocolos comerciales comunes, como por ejemplo *sendNEC() , sendRC5() , sendRC6(), sendSharp()* o *sendPanasonic() .* Todas ellas funcionan con los dos mismos parámetros. Además, se puede usar la función *sendRaw()* para enviar un patrón que no venga por defecto codificada dentro de la librería. Esta función tiene tres parámetros: el primero es un array de valores word que forman en conjunto el patrón completo, el segundo es el tamaño de ese array y el tercero es la frecuencia de modulación en KHz de la señal a enviar (que para nuestros proyectos siempre será 38). Lo más interesante de esta función es que podemos utilizarla para simular cualquier protocolo comercial a pesar de que este no venga por defecto codificado dentro de la librería. Esto lo podemos hacer eligiendo previamente de entre los sketches de ejemplo que vienen con la librería uno llamado “IRrecvDump.ino” y ejecutándolo con nuestro Arduino conectado a un sensor IR. Haciendo esto podremos observar en el “Serial monitor” precisamente los valores word a incluir en el array (el primer parámetro de *sendRaw()*), y también, entre paréntesis, el tamaño de dicho array (el segundo parámetro de *sendRaw()*).

SENSORES DE TEMPERATURA

Termistores



Un termistor es un resistor que cambia su resistencia con la temperatura. Técnicamente, todos los resistores son termistores ya que su resistencia siempre cambia ligeramente con la temperatura, pero este cambio es usualmente muy pequeño y difícil de medir. Los termistores están fabricados de manera que su resistencia cambia drásticamente, de tal manera que pueden cambiar 100 ohmios o más por grado centígrado.

Hay dos tipos de termistores, los llamados NTC (del inglés "negative temperature coefficient") y los PTC (de "positive temperature coefficient"). En los primeros, a medida que aumenta la temperatura, decrece su resistencia; en los segundos, a medida que aumenta la temperatura, aumenta su resistencia. En nuestros proyectos normalmente usaremos NTCs para medir temperatura; los PTCs se suelen usar más dentro de fusibles reseteables (donde si la temperatura crece, incrementan su resistencia para "ahogar" la corriente y proteger así de un posible sobrecalentamiento a los circuitos).

Los termistores son mucho más baratos que otros tipos de sensores de temperatura; además, son resistentes al agua (son solo resistores al fin y al cabo) y trabajan a cualquier voltaje. Son difíciles de estropear debido a su sencillez y son increíblemente precisos en las medidas. Por ejemplo, un termistor de 10 KΩ (valor nominal, tomado a 25 °C como referencia estándar) puede medir temperatura con un margen de error de ±0,25 °C (suponiendo que el conversor analógico-digital sea lo suficientemente preciso también). No obstante, no suelen soportar temperaturas más allá de los 100 y poco grados, y su constante de tiempo (es decir, los segundos que necesita el termistor para reducir un 63% la diferencia entre su temperatura inicial y la final) es normalmente de más de diez segundos.

Para medir la resistencia de un termistor se puede usar un multímetro, como cualquier otra resistencia. El valor que obtengamos dependerá de la temperatura del lugar donde estemos.

Si lo que queremos es medir la temperatura propiamente dicha con una placa Arduino, primero debemos medir con ella la resistencia del termistor, y a partir de

ella deducir la temperatura correspondiente. Pero nuestra placa Arduino no tiene un medidor de resistencias incorporado, por lo que, al igual ya hicimos con los fotorresistores, tendremos que utilizar las entradas analógicas de la placa para detectar variaciones de voltaje y deducir a partir de estas el valor buscado de la resistencia actual. Así pues, el esquema de conexiones es idéntico al que ya usamos con los fotorresistores (y fotodiodos): debemos conectar un terminal del termistor a la alimentación (por ejemplo, el pin 5V de la placa Arduino) y el otro conectarlo en serie a un terminal de una resistencia de valor fijo (que hará de resistencia "pull-down"); el otro terminal de esta resistencia "pull-down" ha de conectarse a tierra. Además, deberemos conectar una entrada analógica de nuestra placa Arduino a un punto intermedio entre ambas resistencias para obtener una lectura de la caída de potencial entre ese punto y tierra.

Mediante el circuito acabado de describir, cuando detectemos que ese voltaje medido está aumentando, podremos deducir, por pura Ley de Ohm, que la resistencia del termistor NTC está disminuyendo y, por tanto, que la temperatura está aumentando (y viceversa: si el voltaje disminuye, la resistencia aumenta y la temperatura disminuye también). La relación exacta entre el voltaje medido y la resistencia del termistor se puede calcular de la misma manera que ya vimos cuando estudiamos los fotorresistores, mediante la fórmula $V_{med}=(R_{pull}/(R_{pull}+R_{termistor}))\cdot V_{fuente}$.

Ya sabemos también que, en realidad, V_{med} no es el voltaje con el que trabajamos en nuestra placa Arduino, porque esta utiliza siempre un conversor analógico-digital con el que realiza un "mapeo". Este mapeo convierte los valores analógicos leídos (los cuales pueden ir oscilando entre 0 V y 5 V si suponemos que el voltaje proporcionado por la fuente son 5 V) a valores digitales (que van entre 0 y 1023). Estos valores digitales son los que la placa Arduino entiende en realidad y con los que trabajaremos en nuestros sketches. La conversión de valores analógicos a digitales se puede expresar por la misma regla de proporcionalidad que vimos con los fotorresistores: $V_{convertido} = V_{med} \cdot 1023/5$. A partir de aquí, tal como hicimos con ellos, si sustituimos esta expresión en la fórmula del párrafo anterior, y despejamos de allí $R_{termistor}$ llegamos a la expresión siguiente: $R_{termistor}=(R_{pull}\cdot 1023/V_{convertido}) - R_{pull}$, la cual nos permite conocer por fin cuál es el valor actual de la resistencia del termistor a partir del voltaje digitalizado obtenido por la placa Arduino. La expresión anterior demuestra, tal como ya sabíamos, que $R_{termistor}$ es inversamente proporcional a $V_{convertido}$.

Pero ¿qué valor ha de tener R_{pull}? El valor más habitualmente empleado para la resistencia "pull-down" que acompaña a nuestro termistor es de 10 KΩ, aunque a veces también se usa 1 KΩ. Ambos son valores ampliamente utilizados, pero si queremos tener un control mayor de la sensibilidad del termistor, tendríamos que

elegir su valor con algo más de criterio. Para ello, el procedimiento sería sustituir en la misma fórmula ya conocida $V_{med}=(R_{pull}/(R_{pull}+R_{termistor}))\cdot V_{fuente}$ diferentes valores numéricos concretos de $R_{termistor}$ correspondientes a temperaturas conocidas (esto se puede consultar en la tabla de equivalencia del datasheet) y elegir un valor determinado de R_{pull} : observando qué valores de V_{med} vamos obteniendo podremos elegir el valor R_{pull} que nos permita tener un rango más amplio de V_{med} sin saturarlo.

El paso natural siguiente, una vez conocido el valor de la resistencia del termistor, sería averiguar a qué temperatura se corresponde. La manera más sencilla de conseguir esto es consultar la tabla de equivalencia que siempre viene en el datasheet, donde para unos determinados valores de $R_{termistor}$ se especifica ya directamente la temperatura correspondiente.

Este sistema nos irá bien si lo único que queremos es diseñar un circuito que realice comparaciones rápidas del tipo "si la temperatura es inferior a tal haz esto y si es superior a tal haz lo otro". Si lo que queremos es, sin embargo, obtener los valores de temperatura reales y precisos, tenemos la suerte de disponer de una ecuación matemática que hace esto con muy buena aproximación (de hecho, logra errores de tan solo ±0,02 ^{o}C en un rango de 100 ^{o}C). Se trata de la ecuación de Steinhart-Hart: $\frac{1}{T} = A + B \cdot \ln(R) + C \cdot (\ln(R))^3$, donde R es el valor en ohmios del termistor en un momento dado, T es la temperatura medida en grados Kelvin (un grado Kelvin es igual a uno centígrado + 273,15) y A, B y C son coeficientes que son diferentes según el tipo y modelo del termistor (y que son válidos solamente para un determinado rango de temperaturas, especificado en el datasheet).

No obstante, como esta fórmula es algo compleja y requiere el conocimiento del valor de varios coeficientes que puede que no conozcamos, en general podemos usar una ecuación simplificada: $\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \cdot \ln\left(\frac{R}{R_0}\right)$, donde T_0 es la llamada temperatura nominal (casi siempre 25 °C = 298,15 K), R_0 es la resistencia del termistor a esa temperatura (la llamada "resistencia nominal", dato disponible en el datasheet) y B es el único coeficiente que necesitamos saber, el cual se puede consultar en el datasheet siempre.

Ejemplo 7.13: Una vez sabida toda la teoría previa, ya podemos implementar el circuito de prueba y empezar a escribir código. Tal como hemos dicho, conectaremos un terminal del termistor a 5 V y el otro lo conectaremos a un terminal de la resistencia "pull-down". El otro terminal de esta última lo conectaremos a tierra y finalmente usaremos un "tercer cable" para conectar un punto central entre ambos dispositivos a un pin analógico de la placa Arduino (por ejemplo, el número 0).

El código mostrado a continuación muestra por el canal serie los valores de $R_{termistor}$, ya calculados a partir de la lectura obtenida de la entrada analógica. No se realiza ningún cálculo de temperatura.

```
const int Rpull=10000;
void setup(void) {
       Serial.begin(9600);
}
void loop(void) {
//El tipo de datos es importante para los cálculos posteriores
       float lectura;
       lectura = analogRead(0);
       Serial.print("Lectura analógica directa ");
       Serial.println(lectura);
      /*Convierto "lectura", que es un valor de voltaje,
        en resistencia, según la fórmula ya vista.*/
       lectura = (Rpull * 1023/lectura)  - Rpull);
       Serial.print("Resistencia del termistor ");
       Serial.println(lectura);
       delay(1000); //Para la estabilidad en las lecturas
}
```

El siguiente trozo de código realiza todos los cálculos necesarios para obtener la lectura en grados centígrados a partir de la ecuación de Steinhart-Hart simplificada. Estas líneas se tendrían que añadir al código anterior, justo antes de su última línea *delay(1000);*. Para que funcione además hay que declarar e inicializar una serie de variables extra: "termistorNominal" con valor 10000, "temperaturaNominal" con valor 25 y "coeficienteB" con el valor concreto de nuestro termistor (en nuestro caso, 3950).

```
float steinhart;
steinhart = media / termistorNominal;           // (R/Ro)
steinhart = log(steinhart);                     // ln(R/Ro)
steinhart /= coeficienteB;                      // 1/B * ln(R/Ro)
steinhart += 1.0/(temperaturaNominal + 273.15); // + (1/To)
steinhart = 1.0 / steinhart;                    // Se invierte
steinhart -= 273.15;                            // Se convierte a ºC
Serial.print("Temperatura: ");
Serial.print(steinhart);
Serial.println(" *C");
```

El valor devuelto tras ejecutar las líneas anteriores es una cifra con dos dígitos decimales. ¿Esto quiere decir que se tiene una precisión de 0,01 °C? ¡No! El termistor tiene errores y el conversor analógico-digital también. Afortunadamente, podemos

estimar el error de nuestra medida a partir del error nominal del termistor (dato consultable en el datasheet con el nombre de "tolerancia"). Supongamos que a la temperatura nominal (25 °C) tenemos un termistor de 10000 Ω con un error (una tolerancia) del 1%, es decir, 100 Ω. Esto implica que para esa temperatura se podrían leer valores desde 9900 Ω hasta 10100 Ω. De la ecuación de Steinhart-Hart simplificada se puede deducir que, si ese termistor tiene un coeficiente B de por ejemplo 3950, una diferencia de 450 Ω arriba y abajo representaría 1 °C de diferencia, así que un error de 100 Ω arriba y abajo (su tolerancia) significa un error de alrededor ±0,25 °C.

Desgraciadamente, aunque existen termistores con tolerancias incluso del 0,1% (los cuales permiten reducir el error hasta ±0,03 °C), la inexactitud en las medidas en realidad siempre es mayor, porque siempre existe como mínimo otra fuente de error que no podemos olvidar: el producido por el conversor analógico-digital. En el caso del conversor presente en la placa Arduino (que tiene una resolución de 10 bits), por cada bit erróneo la resistencia medida puede variar hasta 50 ohmios de la real. Esto representa un error de ±0,1 °C más a sumar a los errores anteriores. En general, por tanto, es recomendable asumir una precisión no mejor que ±0,5 °C.

También hay que tener en cuenta que el termistor sufre un autocalentamiento consecuencia de la potencia disipada, por lo que alcanza una temperatura por encima de la del ambiente, que es a su vez la temperatura que detecta. Esto obliga a limitar el valor de la tensión (o de la corriente, recordemos que $P = V \cdot I$) con la que se alimenta. En este sentido, la resistencia "pull-down" es de mucha ayuda, porque funciona como un divisor de tensión.

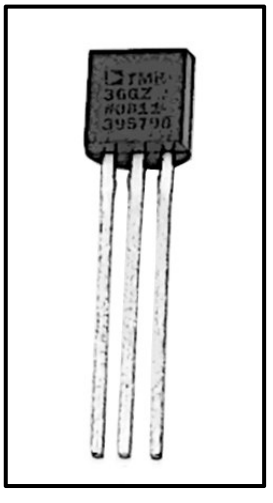
Ejemplo 7.14: En general, cuando se hacen lecturas de valores analógicos, hay dos trucos y que sirven para mejorar la precisión de los resultados. Uno es utilizar el pin 3,3 V como voltaje de referencia analógico (para aumentar la precisión de la conversión analógico-digital) y el otro es tomar un conjunto de lecturas y obtener su media como el resultado a usar (para evitar fluctuaciones o ruido en las medidas).

Para conseguir el primer truco, simplemente conectaremos el pin "3V3" de la placa a su pin AREF (además de añadir una línea con *analogRead()* específica a nuestro código). Para conseguir el segundo truco, modificaremos el código para incluir bucles que recojan varias medidas (cinco están bien) y realicen las medias pertinentes. Así:

```
/*Dependiendo del número de muestras, la media
tarda más (por el bucle) pero es más suave*/
const int numMuestras=5;
const int Rpull=10000;
```

```
int muestras[numMuestras];
void setup(void) {
       Serial.begin(9600);
       analogReference(EXTERNAL);
}
void loop(void) {
       byte i;
       float media;
       for (i=0; i< numMuestras; i++) {
          muestras[i] = analogRead(0);
          delay(10);  //Para la estabilidad entre lecturas
       }
       //Calculo la media
       media = 0;
       for (i=0; i< numMuestras; i++) {
          media = media + muestras[i];
       }
      media = media/numMuestras;
      Serial.print("Lectura analógica media: ");
      Serial.println(media);
      //Convierto "media" a resistencia, según la fórmula
      media= (Rpull * 1023/media)  - Rpull);
      Serial.print("Resistencia del termistor ");
      Serial.println(media);
      delay(1000);
}
```

El chip analógico TMP36



Este chip utiliza una tecnología de estado sólido para medir la temperatura: a medida que la temperatura crece, la caída de potencial entre la Base y el Emisor de un transistor incrementa también una cantidad conocida. Amplificando este cambio de voltaje, se genera una señal analógica que es directamente proporcional a la temperatura. Este tipo de sensores son precisos, no se desgastan, no necesitan calibración, pueden trabajar bajo condiciones climáticas diversas, son bastante baratos y fáciles de usar. En Adafruit se distribuye con código de producto 165 y en Sparkfun con código 10988.

Otros detalles técnicos: su rango de temperatura va desde los -4 0°C hasta los 125 °C y su rango de voltaje de salida

empieza desde 0,1 V (a los -40 °C) y aumenta 10 mV por cada grado centígrado hasta llegar a los 1,75 V (a los 125 °C). Por otro lado, para que su circuitería interna funcione, necesita estar alimentado por una fuente de entre 2,7 V y 5,5 V y 0,05 mA.

Sus conexiones son sencillas: si se tiene enfrente su parte plana (tal como se muestra en la figura), el pin de más a la izquierda (nº 1) ha de conectarse a la alimentación y el de más a la derecha (nº 3) a tierra. El pin del medio (nº 2) es el que sirve para obtener un voltaje analógico linealmente proporcional a la temperatura (e independiente del voltaje proporcionado por la fuente de alimentación). Por tanto, este pin nº 2 se tendrá que conectar a un pin analógico de entrada de nuestra placa Arduino.

Luego, para convertir el voltaje leído en temperatura, simplemente hay que utilizar la siguiente fórmula extraída del datasheet: T = (V - 0,5)*100 donde T se mediría en grados centígrados y V en voltios. Así, por ejemplo, si se mide un voltaje de 1V, la temperatura sería (1V - 0,5)*100 = 50 °C. De la fórmula anterior podemos deducir un par de cosas: que la resolución es de 10 mV por grado (tal como ya hemos comentado) y que a 0 °C existe un voltaje de 500 mV. Esta característica permite que el sensor pueda devolver lecturas de temperaturas bajo cero sin que tengamos que preocuparnos por manejar valores de voltaje negativos.

Si se desea probar este chip antes de incorporarlo a nuestros proyectos, podemos usar un multímetro en modo medición de voltaje DC. Tendremos que alimentar no obstante al chip mientras dure la medición. Para ello, podemos usar dos pilas AA (3 V entra perfectamente dentro del rango de voltaje admitido por el chip), de tal manera que conectemos su borne positivo y negativo a los pines correspondientes. El multímetro deberemos conectarlo al pin 3 (tierra) y al pin 2 para realizar la medición. En una habitación a 25 °C, el voltaje debería de ser de alrededor de 0,75 V. Se puede jugar a apretar los dedos sobre el encapsulado (para calentarlo) o ponerlo en contacto con hielo (para enfriarlo) y ver los cambios de valores medidos.

Ejemplo 7.15: A continuación mostramos un código Arduino que muestra por el canal serie la temperatura (ya calculada) del ambiente. El circuito tan solo consta de este componente conectado a los pines-hembra adecuados de Arduino (alimentación, tierra y pin de entrada analógico, que supondremos el nº 0), y ya está. El voltaje de alimentación podría ser 5 V como siempre o bien los 3,3 V proporcionados por el pin “3V3”: los resultados son independientes de eso. Lo que sí se puede hacer es utilizar el pin “3V3” también como voltaje de referencia conectándolo al pin “AREF” e indicándolo en el código: con esto se mejoraría la precisión de los resultados.

```
void setup(){ Serial.begin(9600); }
void loop() {
       int lectura;
       float voltaje;
       float temperaturaC;
       lectura = analogRead(0);
/*Convierto la lectura (0-1023) en voltaje (0-5). Si el chip se ha
alimentado con el pin "3V3", en la fórmula se ha de usar 3.3 en vez
de 5.0 */
       voltaje = lectura * 5.0 / 1024.0;
       Serial.print(voltaje);
       Serial.println(" voltios");
//Convierto este voltaje en temperatura mediante la fórmula
       temperaturaC = (voltaje - 0.5) * 100 ;
       Serial.print(temperaturaC);
       Serial.println(" grados C");
       delay(1000);  //Para la estabilidad de las lecturas
}
```

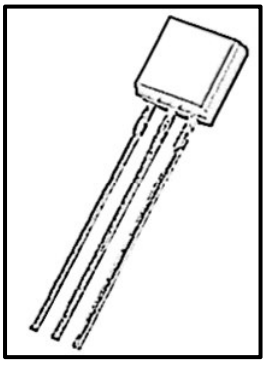
A partir de aquí, se pueden realizar multitud de proyectos interesantes. Por ejemplo, podríamos tener un circuito formado por este sensor y tres LEDs, uno de color rojo, otro azul y otro verde. Cuando se detectara una temperatura más alta que la de un umbral superior predefinido, se podría iluminar el LED rojo, cuando la temperatura fuera más baja de un umbral inferior predefinido, se podría iluminar el LED azul y cuando la temperatura estuviera entre esos dos umbrales, se podría iluminar el LED verde. Se deja como ejercicio.

Ejemplo 7.16: Un código algo más elaborado que el del ejemplo anterior es el siguiente, en el que se asume un circuito con la presencia de, además del sensor TMP36, una pantalla LCD alfanumérica compatible con la librería oficial LiquidCrystal de Arduino y dos pulsadores (conectados a las entradas digitales nº 2 y nº 3 de la placa Arduino). La pantalla irá mostrando la temperatura actual continuamente excepto cuando se pulse un botón, momento en el que mostrará durante unos instantes la temperatura mínima y máxima registrada desde el arranque del sketch. Si se desea eliminar esos registros y obtener una nueva temperatura mínima y máxima a partir de la actual, se deberá apretar el otro botón.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
float Tmin = 0;
float Tmax = 0;
void setup() {
```

```
 //Podríamos haber hecho Tactual global, como Tmin o Tmax
  float Tactual=0;
  lcd.begin(16,2); lcd.clear();
  //Establezco un mínimo y máximo inicial a partir de la T actual
  Tactual=obtenerTemp();
  Tmin=Tactual; Tmax=Tactual;
  pinMode (2,INPUT); //Botón para redefinir la T máxima y mínima
  pinMode (3, INPUT); //Botón para mostrar la T máxima y mínima
}
void loop() {
  float Tactual=0;
  Tactual=obtenerTemp();
  //Si la temperatura actual supera los límites, los redefino
  if (Tactual<Tmin) { Tmin = Tactual; }
  if (Tactual>Tmax) { Tmax = Tactual; }
  mostrarTempActual(Tactual);  delay(100);
  //Si pulso el botón de "reset"
  if (digitalRead(2) == HIGH) { Tmin = Tactual; Tmax = Tactual; }
  //Si pulso el botón de mostrar máximo y mínimo
  if (digitalRead(3) == HIGH) {
    mostrarTempMin(); delay(3000);  mostrarTempMax(); delay(3000);
  }
}
/*Función que obtiene directamente la temperatura a partir
del voltaje leído en la entrada del sensor TMP36 */
float obtenerTemp(){ return (((analogRead(5)*5.0)/1024)-0.5)*100; }
/*Función que muestra la temperatura actual. Como Tactual es una
variable local de loop(), la tengo que pasar como parámetro. En
cambio, con Tmax y Tmin, al ser globales, eso no es necesario */
void mostrarTempActual(float Tactual){
   lcd.clear(); lcd.setCursor(0,0); lcd.print("Temperatura actual:");
   lcd.setCursor(0,1);  lcd.print(Tactual);  lcd.print(" C/");
}
//Función que muestra la temperatura mínima
void mostrarTempMin(){
   lcd.clear(); lcd.setCursor(0,0); lcd.print("Temperatura mínima:");
   lcd.setCursor(0,1);  lcd.print(Tmin);  lcd.print(" C/");
}
//Función que muestra la temperatura máxima
void mostrarTempMax(){
   lcd.clear(); lcd.setCursor(0,0); lcd.print("Temperatura máxima:");
   lcd.setCursor(0,1);  lcd.print(Tmax);  lcd.print(" C/");
}
```

El chip digital DS18B20 y el protocolo 1-Wire



El fabricante Maxim (anteriormente conocido como Dallas Semiconductor) produce una familia de componentes electrónicos que pueden ser controlados mediante un protocolo de comunicación propietario llamado "1-Wire", el cual permite conectar a nuestra placa Arduino multitud de estos componentes mediante un solo cable de datos (de ahí su nombre). Otra característica destacable es que los componentes interconectados mediante este protocolo pueden estar situados a grandes distancias (de hasta incluso 30 metros).

El termómetro digital DS18B20 es un chip que utiliza el protocolo 1-Wire. Es muy popular debido a su bajo precio y facilidad de uso. Es capaz de medir temperaturas en un rango de -10 ^{o}C hasta 85 ^{o}C con una precisión de ±0,5 °C. En Sparkfun se puede encontrar con el código de producto nº 245, en Adafruit con el código nº 374 y en Freetronics podemos adquirirlo en forma de plaquita breakout, llamada "Temperature sensor module", entre otros.

Si observamos de frente la cara lisa de su encapsulado, podemos observar que tiene tres patillas: la de más a la izquierda se corresponde con la conexión a tierra, la patilla central es la salida digital de la señal de datos (a conectar a una entrada digital de la placa Arduino) y la patilla derecha sirve para recibir la alimentación, la cual puede ser perfectamente los 5 V ofrecidos por la placa Arduino. Por tanto, en principio necesitaríamos tres cables para utilizar este componente. Además, es muy importante, para que el sensor funcione, conectar una resistencia de 4,7 KΩ entre la patilla de señal de datos y la patilla de alimentación.

No obstante, este chip (como el resto de componentes 1-Wire, de hecho) tiene la característica de poder conectarse de otra manera, utilizando tan solo dos cables. Esto es muy conveniente en sensores alejados cierta distancia de nuestra placa Arduino y/o situados en un entorno exterior. Es lo que se llama modo "parásito", porque la alimentación que necesita la "parasita" de la señal de datos. Concretamente, las conexiones son: la patilla de la izquierda a tierra (como antes), la patilla central a una entrada digital de la placa Arduino (como antes) y la patilla derecha hay que unirla directamente a la patilla izquierda (para conectarla a tierra). Además, es muy importante conectar una resistencia de 4,7 KΩ entre el pin de entrada digital de la placa Arduino donde está conectada la patilla central y la alimentación de 5 V.

Otra característica muy interesante que ofrecen los componentes 1-Wire es la posibilidad de conectarse entre sí para formar una red (en nuestro caso, de sensores de temperatura) utilizando tan solo dos cables (es decir, aprovechando el modo "parásito"), independientemente del número de componentes conectados. El cableado en este caso es así: deberíamos conectar un primer sensor en modo parásito tal como lo hemos descrito en el párrafo anterior (incluyendo la resistencia de 4,7 KΩ), y los siguientes sensores han de tener todos su patilla izquierda conectada a la patilla izquierda del primer sensor (para mantener una misma tierra común), su patilla central conectada a la patilla central del primer sensor (para compartir el mismo –y único– cable de datos) y su patilla derecha conectada también a la patilla izquierda del primer sensor.

La conexión de múltiples componentes 1-Wire a un mismo cable es posible gracias a que cada uno de ellos tiene una dirección interna única formada por un código de 64 bits, dividido en un 8 bits para identificar el modelo de componente (el DS18B20 tiene el identificador 0x28), 48 bits para identificar ese componente individualmente en particular y 8 bits más (llamados código CRC) que sirven para comprobar que no haya errores en la identificación de ese componente por parte de los demás componentes del circuito (como por ejemplo, nuestra placa Arduino).

Para programar el DS18B20 mediante el lenguaje Arduino, podemos descargar de http://www.arduino.cc/playground/Learning/OneWire e instalar la librería "One-Wire". Gracias a ella, la placa Arduino puede establecer comunicación con cualquier dispositivo diseñado para trabajar bajo el protocolo 1-Wire. No obstante, precisamente porque esta librería es genérica para cualquier dispositivo compatible con 1-Wire (y por tanto, muy flexible y versátil), es relativamente complicada de utilizar, ya que hay que conocer relativamente bien algunos detalles internos del protocolo 1-Wire. Si lo que deseamos es utilizar en concreto uno o más sensores de tipo DS18B20, tenemos la suerte de disponer de una librería específica, mucho más sencilla. Esta librería, llamada "Dallas Temperature Control Library " y disponible en http://milesburton.com/Dallas_Temperature_Control_Library , necesita que se haya instalado previamente la librería "One-Wire" para funcionar, por lo que necesitaremos incluir las dos en nuestros sketches. Gracias a ella podremos leer el código identificador de cada sensor, configurar la resolución de las medidas (de 9 a 12 bits) y obtener el valor de la temperatura de cada uno de los sensores de una forma muy simple.

Ejemplo 7.17: A continuación, se muestra un código de ejemplo, donde se ha supuesto que la patilla de datos de un sensor está conectado al pin digital de entrada nº 2 de la placa Arduino:

```
#include <OneWire.h>
#include <DallasTemperature.h>
/*Creo un objeto de tipo "OneWire". Esta instrucción pertenece a la
librería One-Wire, por lo que es genérica para cualquier dispositivo
compatible con ese protocolo, no solamente el DS18B20. El valor del
parámetro indica el número de pin digital de la placa Arduino donde
está conectada la patilla de datos del sensor (o sensores). */
OneWire dispositivoOneWire(2);
/*Pasamos la referencia del objeto recién
creado a la librería DallasTemperature */
DallasTemperature sensores(&dispositivoOneWire);
setup(){
      Serial.begin(9600);
/*Se inicializa la librería. Opcionalmente, se puede especificar un
parámetro para ajustar la precisión de las medidas: por defecto es de
9 bits, pero puede aumentar hasta 12. La contrapartida es una mayor
lentitud en las lecturas*/
      sensores.begin();
}
loop(){
/*Solicito las temperaturas de todos los
posibles sensores presentes en el canal de datos */
      sensores.requestTemperatures();
      Serial.print("Temperatura para el dispositivo 1 es: ");
/*El parámetro de getTempCByIndex indica el número de sensor del cual
se quiere obtener la temperatura en grados Celsius: el "0" se
corresponde al primer sensor presente en el cable de señal de datos,
el "1" sería el siguiente, y así*/
      Serial.print(sensores.getTempCByIndex(0));
}
```

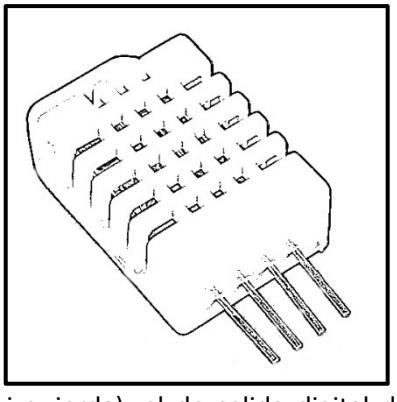
La plaquita breakout TMP421

Modern Device distribuye una plaquita que incluye el chip analógico TMP421. Debemos conectarla a los pines analógicos nº 2, 3, 4 y 5 de la placa Arduino. El primer servirá para alimentar la placa, el segundo para conectarla a tierra, y los dos últimos sirven para establecer el canal I^2C a través del cual se realizará la comunicación entre ambos dispositivos. Puede medir temperaturas desde -40 ^oC hasta 125 ^oC con un error de $\pm$1 ^oC. Es realmente fácil de usar mediante la librería disponible en https://github.com/moderndevice/LibTempTMP421.

SENSORES DE HUMEDAD

El sensor DHT22/RHT03

En este apartado hablaremos concretamente del sensor digital de temperatura y humedad RHT03 de Maxdetect (http://www.humiditycn.com) , el cual se distribuye (además de muchos otros) en Sparkfun con nº de producto 10167 y en Adafruit (aquí con el nombre de DHT22 y con nº de producto nº 385). También se le puede encontrar con el nombre de AM2302. Este sensor es muy básico y lento (solo se pueden obtener datos como mínimo cada 2 segundos), pero es de bajo coste y muy manejable para obtener datos básicos en proyectos caseros.



Sus características técnicas más destacables son: se puede alimentar con un voltaje de entre 3 V y 5 V y 2,5 mA como máximo, puede medir un rango de temperaturas entre -40 y 125 °C con una precisión de ±0,5 °C y un rango de humedad entre 0 y 100% con una precisión del 2-5%. Está formado básicamente por un sensor de humedad capacitivo y un termistor.

Este chip tiene cuatro pines; mirándolo de frente son: el de alimentación (nº 1, el de más a la izquierda), el de salida digital de datos (nº 2), uno no conectado a nada y que se puede ignorar (nº 3) y el de tierra (nº 4, el de más a la derecha). Así pues, para que nuestra placa Arduino pueda leer los datos que emite este chip, deberemos conectar su pin nº 1 al pin-hembra de 5 V de la placa (por ejemplo), su pin nº 4 a un pin "GND" y su pin nº 2 a una pin-hembra de entrada digital. Además, es recomendable conectar una resistencia ("pull-up") de 4,7 KΩ entre el pin nº 1 y nº 2.

Desgraciadamente, el protocolo que utiliza este sensor para transmitir los datos digitales no es estándar, así que en principio deberíamos de aprender su funcionamiento interno para poder interpretar correctamente la información que nos esté llegando en un momento determinado. Por suerte, tenemos a nuestra disposición gran cantidad de librerías Arduino, compatibles con este sensor. Estas librerías, a pesar de ser diferentes entre sí, son en gran medida equivalentes, ya que todas ellas lo que hacen básicamente es permitir centrarnos en la obtención sencilla de los datos de temperatura y humedad sin tener que conocer los detalles específicos del protocolo particular utilizado por el chip. Por lo tanto, la elección de una librería u otra no es determinante.

Ejemplo 7.18: La gente de Adafruit, por ejemplo, ha programado una librería Arduino compatible con este sensor (https://github.com/adafruit/DHT-sensor-library) Una vez instalada como cualquier otra librería, podemos probarla ejecutando el siguiente sketch de ejemplo, el cual nos muestra por el "Serial monitor" la temperatura y humedad ambiente. Para ver cambios en los datos obtenidos, se puede probar de expulsar vaho sobre el sensor si se quiere aumentar la humedad leída:

```
#include <DHT.h>
/*El primer parámetro es el número del pin-hembra de la placa Arduino
donde está conectado el pin de datos del sensor. El segundo dato
especifica el modelo de sensor que se está utilizando (ya que la
librería puede servir para varios. El valor correspondiente para el
DHT22 es "22". Para consultar los otros valores posibles, se puede
mirar el archivo "DHT.h" que viene incluido dentro de la librería.*/
DHT dht(2, 22);
void setup() {
       Serial.begin(9600);
       dht.begin();
}
void loop() {
      float h,t;
      //Las lecturas del sensor pueden tardar hasta 2 segundos
       h = dht.readHumidity();
       t = dht.readTemperature();
/*Compruebo que los datos recibidos son válidos. Concretamente, la
función isnan() mira si el dato no es un número ("IS Not A Number"),
en cuyo caso algo ha ido mal*/
      if (isnan(t) || isnan(h)) {
             Serial.println("Algo falló ");
       } else {
            Serial.print("Humedad: ");
            Serial.print(h);
            Serial.print(" %\t");
            Serial.print("Temperatura: ");
            Serial.print(t);
            Serial.println(" *C");
      }
}
```

Ejemplo 7.19: Si lo que queremos es (en vez de mostrar los datos obtenidos por el canal serie) guardarlos en una tarjeta SD para poderlos estudiar con calma

posteriormente, deberíamos modificar el sketch anterior para que se parezca al siguiente. En él hemos añadido comprobaciones extra para tener en cuenta los posibles errores que pueden ocurrir en la lectura en la tarjeta:

```
#include <SD.h>
#include <DHT.h>
DHT dht(2,22);
const int intervalo = 10*1000; //Intervalo entre lecturas, en ms
long tiempoUltimaLectura = 0;  //Tiempo de la última lectura, en ms
long tiempoActual = 0;         //Tiempo actual devuelto por millis()
float h,t;                     //Humedad y temperatura obtenidas
void setup() {
  /*Utilizo una función propia para inicializar la tarjeta SD. Si la
  tarjeta se inicializa correctamente, inicializo la librería DHT */
  if (inicioSD() == true) {
     dht.begin();
  }
}
void loop(){
  /*Uso el truco de repetir código cada determinado tiempo
  (especificado por "intervalo") sin utilizar delay() */
  tiempoActual = millis();
  if (tiempoActual > tiempoUltimaLectura + intervalo) {
    //Obtengo los datos de temperatura y humedad
    h = dht.readHumidity();
    t = dht.readTemperature();
    //Si las lecturas NO son erróneas (atención al "!")
    if (!isnan(t) || !isnan(h)) {
       //Abro el fichero y escribo a partir de la última línea
        File mifichero = SD.open("datalog.csv", FILE_WRITE);
        if (mifichero==true) {
            mifichero.print(h);
            mifichero.print("\t");  //Tabulador
            mifichero.println(t);
            mifichero.close();
            tiempoUltimaLectura = millis();
        }
    }
}
boolean inicioSD() {
  boolean resultado = false;
  //Aunque no se utilice, el pin 10 se ha de configurar como salida
  pinMode(10, OUTPUT);
  //Suponemos que la tarjeta está conectada al canal SS vía pin 4
```

```
  if (SD.begin(4)==false) {
    //Si no lo está, el sketch no hará nada
    resultado = false;
  }
  /*Si sí, abro el fichero "datalog.csv" para escribir una línea
  (lo que sería el título) a continuación de las posibles líneas
  anteriores que existan de otras ejecuciones previas.*/
  else {
    File mifichero = SD.open("datalog.csv", FILE_WRITE);
    if (mifichero == true) {
      mifichero.println();
      mifichero.println("rH (%) \t temp. (*C)");
      mifichero.close();
      resultado = true;
    }
  }
  return resultado;
}
```

Lo interesante de los dos sketches anteriores es que ambos envían (uno al canal serie y el otro a la tarjeta SD) los datos de temperatura y humedad de forma tabulada, con incluso un título para las columnas. Este hecho lo podemos aprovechar para importar fácilmente esta información en programas de procesamiento de datos, (como pueden ser las hojas de cálculo), para realizar así cálculos estadísticos o representación de gráficas, entre otras posibilidades.

En el caso del fichero guardado en la tarjeta SD, para realizar esta importación lo único que deberemos hacer es colocar dicha tarjeta en un lector conectado a nuestro computador y abrir ese fichero con nuestro programa preferido, como puede ser el software LibreOffice Calc (http://www.libreoffice.org, aplicación de hoja de cálculo libre, multiplataforma y gratuita) o, si tan solo se requiere el dibujo de gráficas y nada más, el software LiveGraph (http://www.live-graph.org, graficador también libre, multiplataforma y gratuito).

En el caso de obtener los datos a través del “Serial monitor”, lo que deberemos hacer es lo siguiente: pulsar dentro de él la combinación de teclas CTRL+A para seleccionar todos los datos y pulsar CTRL+C para copiarlos en memoria. A partir de aquí, podemos abrir un nuevo fichero de texto plano y pulsar CTRL+V para pegar los datos, o bien abrir una nueva hoja de cálculo usando por ejemplo LibreOffice Calc. En este último caso, una vez ya dentro de la hoja de cálculo, al pulsar también CTRL+V para pegar los datos veríamos como en ese momento aparece un cuadro emergente

de configuración de la importación de texto. En ese cuadro deberíamos seleccionar la opción "separado por tabulador" (o similar) para que los datos fueran correctamente incorporados.

Si utilizáramos otro programa diferente del "Serial monitor" para ver los datos recibidos por el canal serie, podríamos transferir estos a una hoja de cálculo de una forma más sencilla, ya que la gran mayoría de ellos tienen la opción de guardar a tiempo real los datos recibidos en un fichero de texto listo para ser importado.

Otra manera de procesar los datos leídos por el canal serie que permita realizar un análisis posterior sobre ellos es utilizar un programa gratuito llamado MegunoLink (http://www.blueleafsoftware.com). Este programa es capaz (entre otras funcionalidades) de dibujar por pantalla y en tiempo real una gráfica correspondiente a los datos recibidos por el canal serie. Para que funcione, deberemos utilizar dentro de nuestros sketches una librería que nos proporcionan los mismos desarrolladores de MegunoLink, la llamada "Arduino graphing library". Junto con la librería se ofrecen varios códigos de ejemplo donde se puede ver su comportamiento; allí se puede observar cómo podemos configurar las nombres de los ejes X e Y de la gráfica y sus diferentes leyendas. Desgraciadamente, MegunoLink solo funciona en sistemas Windows. Otra graficador en tiempo real (que también funciona solo para Windows) es SerialChart (http://code.google.com/p/serialchart).

Volviendo a los sensores de humedad, otra librería compatible con el sensor DHT22/RHT03 es la que podemos encontrar en https://github.com/ringerc/Arduino-DHT22. Otra más es la disponible en http://arduino.cc/playground/Main/DHTLib. Por otro lado, Freetronics distribuye una plaquita breakout con el mismo sensor DHT22/RHT03 (bajo el nombre de "Humidity and Temperature Sensor Module"), que ofrece tres conectores: tierra, señal de datos y alimentación (listados de izquierda a derecha) y se puede programar mediante una librería de la propia Freetronics, disponible en https://github.com/freetronics/DHT-sensor-library, muy parecida a la de Adafruit.

Los sensores SHT15 y SHT21

Otro sensor de humedad (y temperatura) es el SHT15, distribuido por Sparkfun en forma de plaquita breakout con código de producto 8257. Esta plaquita consta de cuatro pines: alimentación (5 V), tierra, conector "SCK" y conector "DATA". Estos dos últimos han de conectarse a dos entradas digitales cualesquiera de nuestra placa Arduino. Aunque el protocolo de comunicación que utiliza este chip requiere el uso de dos cables al igual que pasa con I^2C, no hay que confundirlos porque son

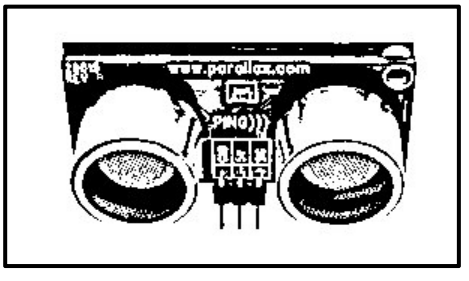
sistemas de comunicación diferentes. Como datos técnicos a destacar, podemos decir que alcanza una precisión de hasta ±0,3 °C en medidas de temperatura y ±2 % en medidas de humedad y tiene un tiempo de respuesta menor de 4 segundos.

La manera más sencilla de programar este sensor es utilizando la librería disponible en https://github.com/practicalarduino/SHT1x (también válida para otros sensores de la misma familia fabricada por Sensirion, como el SHT10, el STH11, el SHT71 o el SHT75).

Por otro lado, un chip del mismo fabricante que sí es capaz de utilizar la comunicación I^2C es el sensor SHT21. LoveElectronics distribuye una plaquita breakout que ofrece de forma muy cómoda los cuatro conectores necesarios: alimentación (3,3 V, atención), tierra, pin SDA y pin SCL. Se puede programar mediante la librería (también válida para el SHT25) disponible en https://github.com/misenso/SHT2x-Arduino-Library. Otra plaquita muy parecida la distribuye Modern Device, junto con la librería "LibHumidity", descargable de la web del producto.

SENSORES DE DISTANCIA

El sensor Ping)))



El sensor digital de distancia Ping)))™ de Parallax es capaz de medir distancias entre aproximadamente 3 cm y 3 m. Esto lo consigue enviando un ultrasonido (es decir, un sonido de una frecuencia demasiado elevada para poder ser escuchado por el oído humano) a través de un transductor (uno de los cilindros que se aprecian en la figura lateral) y espera a que este ultrasonido rebote sobre un objeto y vuelva, retorno que es detectado por el otro transductor. El sensor devuelve el tiempo transcurrido entre el envío y la posterior recepción del ultrasonido. Como la velocidad de propagación de cualquier (ultra)sonido en un medio como el aire es de valor conocido (consideraremos que es de 340 m/s –o lo que es lo mismo, 0,034 cm/µs–, aunque esta sea solo una aproximación) este tiempo transcurrido lo podremos utilizar para determinar la distancia entre el sensor y el objeto que ha provocado su rebote.

La plaquita en la que viene muestra tres pines marcados. Teniendo visibles enfrente los transductores son, de izquierda a derecha: el pin de tierra, el de

alimentación (5 V) y el pin para comunicarse con un pin-hembra digital de nuestra placa Arduino (lo llamaremos a partir de ahora "pin de señal").

Este sensor solo mide distancias cuando se le solicita. Para ello, nuestra placa Arduino envía a través de su pin digital (conectado al pin de señal de la plaquita) un pulso HIGH de una duración exacta de 5 microsegundos. Esta es la señal que activa el envío del ultrasonido. Tras un breve lapso de tiempo, el sensor recibirá el rebote del ultrasonido y como consecuencia, nuestra placa Arduino recibirá ese dato por el mismo pin digital utilizado anteriormente. En ese momento, la placa Arduino podrá calcular el tiempo transcurrido entre el envío y la recepción de ese ultrasonido. El funcionamiento descrito obliga a que se deba alternar el modo del pin digital de la placa Arduino conectado al sensor, de forma que sea de tipo INPUT o OUTPUT según convenga.

Para calcular la distancia, debemos recordar la fórmula **v = d/t** (que no es más que la definición de velocidad: distancia recorrida en un determinado tiempo). Si de la fórmula anterior despejamos la "d" (la distancia recorrida) obtenemos finalmente d = v·t, donde "v" la conocemos y la consideramos constante (0,034 cm/µs) y "t" es el tiempo medido por el sensor. Hay que tener en cuenta, no obstante, que el dato obtenido del sensor es el tiempo total que tarda el ultrasonido en "ir y volver", así que normalmente querremos dividir previamente este valor entre dos para asignárselo a "t".

Ejemplo 7.20: El siguiente sketch realiza todos estos cálculos y muestra el resultado a través del canal serie. Hemos supuesto que el pin de señal de la plaquita está conectado al pin-hembra digital nº 8 de Arduino. Se puede probar en tiempo real poniendo un obstáculo enfrente del sensor y moviéndolo adelante y atrás.

```
int distancia;
unsigned long tiempo=0;
void setup(){
      Serial.begin(9600);
}
void loop(){
      //Obtengo la lectura de la distancia medida por el sensor
      enviarYRecibir();
      /*Convertimos el tiempo a distancia, sabiendo
      la velocidad del ultrasonido (en cm/microsegundos) */
      distancia = int(0.034*tiempo);
      Serial.print("Distancia: ");
      Serial.print(distancia);
```

```
      Serial.println(" cm");
      delay(500);
}
void enviarYRecibir(){
/*Configuramos el pin de datos como
salida y estabilizamos el sensor para
poder enviarle el pulso de activación*/
      pinMode(8, OUTPUT);
      digitalWrite(8, LOW);
      delayMicroseconds(5);
/*Ahora es cuando enviamos el pulso
de 5 microsegundos para activar el sensor
y enviar el ultrasonido */
      digitalWrite(8, HIGH);
      delayMicroseconds(5);
      digitalWrite(8, LOW);
/*Mientras el ultrasonido viaja, cambiamos
el modo del pin de datos a entrada
para leer el rebote del pulso */
      pinMode(8, INPUT);
/*Medimos la longitud del pulso entrante. El truco aquí está en saber
que justo después de enviar el ultrasonido, el sensor comienza
siempre a emitir una señal HIGH, la cual será recibida por el pin de
la placa Arduino acabado de configurar como entrada (el n° 8 en este
caso). Pero en el momento que se reciba el rebote, el sensor
automáticamente cambiará esa señal HIGH a LOW. Esto permite utilizar
la función pulseIn() para medir el tiempo transcurrido entre un
instante y otro: tal como está escrita en el sketch, esta función
cuenta el tiempo que pasa desde que el pin n° 8 empieza recibiendo
una señal de valor HIGH (en el momento de enviar el pulso) hasta que
lo deja de recibir (en el momento de recibir el rebote). La duración
de esa señal HIGH recibida se corresponde con el tiempo que queremos
medir*/
      tiempo=pulseIn(8, HIGH);
/*Divido la longitud del pulso a la mitad, porque solo quiero
utilizar el tiempo tardado en la ida del ultrasonido, no el
de ida y vuelta*/
      tiempo=tiempo/2;
}
```

A partir de la detección de la presencia de un objeto, las reacciones del circuito pueden ser muy diversas: pueden consistir en la activación de alarmas acústicas o lumínicas, o en la apertura o cierre de diferentes compuertas (a través de servomotores) o en cualquier otra cosa que la imaginación sugiera. Por ejemplo,

como posible ejercicio proponemos la construcción de un radar casero: solo tendríamos que situar el sensor sobre un servomotor que continuamente estuviera moviéndose entre 0 y 180 grados (y viceversa). En el momento que se detectara algún objeto a una distancia menor de la establecida como umbral, se podría iluminar un LED o activar un zumbador, por ejemplo.

El sensor SRF05

Otro sensor digital que utiliza el método de contar el tiempo transcurrido entre la emisión de un pulso ultrasónico y su posterior recepción para medir distancias es el SRF05 de Devantech. Es capaz de medir distancias entre 3 cm y 3 m a un ritmo de hasta 20 veces por segundo.

Este sensor se puede conectar de dos maneras diferentes a nuestra placa Arduino: o bien utilizando cuatro cables ("modo 1" compatible con su predecesor, el sensor SRF04), o bien usando tres ("modo 2"). Si observamos el dorso del sensor y mantenemos la serigrafía "SRF05" visible a nuestra izquierda, en el modo 1 los cinco conectores de la zona inferior se corresponden, de izquierda a derecha, con: alimentación (5 V), entrada del rebote ultrasónico (pin "echo"), salida del pulso ultrasónico (pin "trigger"), pin que no se ha de conectar a nada y tierra. El pin "echo" se ha de conectar a un pin de entrada digital de nuestra placa Arduino y el pin "trigger" a un pin de salida digital. Este pin "trigger" es el responsable de generar una pulso con una duración exacta de 10 microsegundos, el cual marca el inicio del envío de la señal ultrasónica, y el pin "echo" utiliza el mismo truco que el sensor Ping))) para contar el tiempo transcurrido entre envío y recepción del ultrasonido: el mantener una señal HIGH mientras no se reciba el rebote.

Si utilizamos el modo 2, estos mismos conectores se corresponden (de izquierda a derecha también) con: alimentación (5 V), pin que no se ha de conectar a nada, salida y entrada todo en uno de la señal ultrasónica, tierra y tierra otra vez. En este modo, el sensor utiliza un solo pin para enviar el pulso y recibir el rebote (tal como ocurre de hecho con el sensor Ping))). Esto permite utilizar un cableado más simple, aunque la programación se complica, porque el mismo pin ha de alternar entre ser entrada y salida dependiendo de las circunstancias.

Ejemplo 7.21: El código siguiente, que utiliza el modo 1, gradúa el brillo de un LED enviándole una señal PWM que variará según lo cercano que esté un obstáculo al sensor: la idea es que cuanto más cerca esté, más brille el LED. Concretamente, a cada centímetro (por debajo de los 255 cm) más próximo se incrementa el brillo del LED en un punto más de la señal PWM. Si se deseara utilizar el rango máximo de

distancias que permite el sensor, el brillo del LED se debería calcular de otra manera, (como por ejemplo en función del tanto por ciento de la distancia medida respecto a la distancia máxima posible), pero tal como se presenta aquí, el código es más sencillo.

Otro detalle del código digno de comentar es que, para evitar obtener ruido (es decir, valores individuales muy fluctuantes) y por tanto conseguir lecturas más suavizadas, se ha utilizado un sistema ya visto anteriormente en otras ocasiones: el cálculo de la media de las últimas mediciones realizadas.

También se puede observar que para obtener la distancia, el tiempo transcurrido entre la señal de "trigger" y de "echo" (que es, en definitiva, lo que mide el sensor, calculado en microsegundos) se ha multiplicado por 0,017. ¿Por qué? Porque tal como se explicó cuando se vio el sensor Ping))), se ha hecho servir la fórmula básica d = v·t, donde v = 0,034 (la velocidad de un ultrasonido en el aire en unidades de cm/µs ,o lo que es lo mismo, 340 m/s) y "t" se ha de dividir entre 2 para contar solo el trayecto de "ida" y no de "ida y vuelta".

```
const int nLecturas=10; //Número de lecturas para hacer media
int lecturas[nLecturas]; //Array que guarda las últimas lecturas
int indice = 0;          //Posición actual dentro del array
int total = 0;           //Suma de las lecturas guardadas en el array
int media = 0;           //Media de las lecturas guardadas en el array
unsigned long tiempo = 0;
unsigned long distancia = 0;
void setup() {
  pinMode(9, OUTPUT); //Donde está conectado el LED (señal PWM)
  pinMode(2, INPUT);  //Donde está conectado el pin "echo" del sensor
  pinMode(3, OUTPUT); //Donde está conectado su pin "trigger"
  //Creo un array inicialmente vacío
  for (int i = 0; i < nLecturas; i++) {lecturas[nLecturas] = 0;}
  Serial.begin(9600);
}
void loop() {
  //Estabilizamos el sensor antes de enviar el pulso de activación
  digitalWrite(3, LOW);
  delayMicroseconds(5);
  digitalWrite(3, HIGH); //Envío el pulso de activación
  delayMicroseconds(10); //La duración de este pulso ha de ser 10 µs
  //Paro la activación y comienza el envío de la señal ultrasónica
  digitalWrite(3, LOW);
  /*Inmediatamente después de empezar el envío del ultrasonido, por
```

```
  el pin "echo" se empieza a recibir una señal HIGH. La función
  pulseIn() pausa entonces el sketch para contar el tiempo
  transcurrido hasta recibir el rebote, momento en el cual por el pin
  "echo" pasa a detectarse una señal LOW y pulseIn() devuelve su
  resultado.*/
  tiempo = pulseIn(2, HIGH);
  /*Calculo la distancia (distancia en cm = velocidad en cm/µs
  multiplicado por el tiempo en µs). Ya se ha dividido entre dos para
  contar solo el tiempo de ida*/
  distancia = 0.017*tiempo;
  /*Pero no quiero obtener un valor de distancia individual, porque
  puede ser muy fluctuante: quiero que el valor tenido en cuenta sea
  una media de los últimos diez valores individuales medidos. Para
  ello, primero elimino de la suma total el valor ubicado en el
  índice actual del array, que corresponde a la medida tomada hace
  diez iteraciones */
  total= total - lecturas[indice];
  /*Ahora añado la nueva distancia medida precisamente en ese
  elemento del array, sobrescribiéndolo. De esta manera, se hace una
  reescritura cíclica de cada elemento cada diez lecturas*/
  lecturas[indice] = distancia;
  //Y finalmente, añado este nuevo valor a la suma total otra vez
  total= total + lecturas[indice];
  indice = indice + 1; //Voy al siguiente elemento del array
  //Al final del array (10 elementos) vuelvo a empezar
  if (indice >= nLecturas)  {  índice = 0;  }
  media = total / nLecturas;  //Calculo la media
  /*Si la distancia es menor que 255 cm cambio el brillo del LED, de
  forma que a menor distancia más brillante sea */
  if (media < 255) {
     analogWrite(9, 255 - media);
  }
  delay(100); //El tiempo mínimo entre lecturas ha de ser de 20µs
}
```

Si quisiéramos utilizar el modo 2, simplemente tendríamos que haber sustituido las primeras líneas de la función "loop()" del sketch anterior (hasta la línea de *pulseIn(),* esta incluida) por las siguientes (donde suponemos que el pin "trigger-echo" del sensor está conectado al pin nº 2 de la placa Arduino):

```
/*Mandamos un pulso bajo de 5 microsegundos para asegurar que siempre
se inicie en LOW, y seguidamente mandamos una señal HIGH que sirve
para iniciar las mediciones*/
pinMode(2, OUTPUT);
```

```
digitalWrite(2, LOW);
delayMicroseconds(5);
digitalWrite(2, HIGH);
delayMicroseconds(10);
digitalWrite(2, LOW);
/*Como utilizamos el mismo pin para recibir el eco, lo cambiamos de
salida a entrada*/
pinMode(2, INPUT);
tiempo = pulseIn(2, HIGH);
```

El sensor HC-SR04

Este sensor es muy parecido a los anteriores. Dispone de cuatro pines: “VCC” (se ha de conectar a una fuente de 5 V), “Trig” (responsable de enviar el pulso ultrasónico; por tanto, se deberá conectar a un pin de salida digital de la placa Arduino), “Echo” (responsable de recibir el eco de ese pulso; luego se deberá conectar a un pin de entrada digital de la placa Arduino) y “GND” (a tierra). Se puede adquirir en IteadStudio o Elecfreaks por menos de diez euros.

Al igual que los anteriores sensores, tiene un rango de distancias sensible entre 3 cm y 3 m con una precisión de 3 mm, y su funcionamiento también es muy parecido: tras emitir por el pin “trigger” una señal de 10 µs para iniciar el envío de la señal ultrasónica, espera a detectar el eco mediante la detección del fin de la señal HIGH recibida por el pin “echo”.

De hecho, el código de ejemplo mostrado para el modo 2 del sensor SRF05 puede ser utilizado sin ningún tipo de cambio en este sensor. De todas maneras, si se desea, se puede utilizar la librería “New-Ping”, descargable desde http://code.google.com/p/arduino-new-ping, la cual ofrece una manera sencilla y común de gestionar diferentes modelos de sensores de distancia ultrasónicos, como por ejemplo el propio HC-SR04 (pero también el sensor Ping))) y el SRF05, entre otros).

El sensor LV-EZ0

Otro sensor de distancia que utiliza ultrasonidos es el sensor LV-EZ0 de Maxbotix. No obstante, a diferencia de los anteriores, el LV-EZO es un sensor analógico. Por ello, para usarlo con nuestra placa Arduino deberemos conectar (además del pin “+5 V” a la alimentación de 5V proporcionada por la placa Arduino y del pin “GND” a la tierra común) el pin etiquetado como “AN” a una entrada analógica de nuestra placa Arduino.

El rango de distancias que puede medir este sensor depende mucho del tamaño del obstáculo: si este es del tamaño de un dedo, el rango es aproximadamente de 2,5 metros; si este es del tamaño de una hoja de papel, el rango puede aumentar hasta 6 metros. En todo caso, no es capaz de detectar distancias más pequeñas de 30 cm. La buena noticia está en que el comportamiento de este sensor es lineal: si un obstáculo está por ejemplo a 2 metros, la lectura de tensión recibida por el pin de entrada analógica será la mitad que si está a 4 metros. Esto permite que las lecturas sean muy fáciles de realizar.

Ejemplo 7.22: Podemos utilizar un sketch tan simple como el siguiente para observar las diferentes distancias detectadas de un obstáculo a lo largo del tiempo.

```
int sensorPin = 0; //El sensor está conectado al pin analógico n° 0
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.println(analogRead(sensorPin));
  /*El delay() ralentiza la toma de lecturas para hacerlas más
fáciles de leer. No obstante, si queremos detectar la presencia de
obstáculos que rápidamente atraviesan de un lado al otro el espacio
sensible, este delay() se podría reducir o quitar */
  delay(100);
}
```

¿Cómo se podría modificar el código anterior para que, añadiendo un LED conectado a un pin de salida digital de nuestra placa Arduino, este se encendiera cuando se detectara la presencia de un obstáculo más cerca de una determinada distancia umbral? Se deja como ejercicio.

Existen varias versiones de este sensor con diferentes rangos de distancia detectables y distintas anchuras del espacio sensible dentro del cual se pueden reconocer los obstáculos. El sensor EZ1 tiene un espacio sensible más estrecho que el EZ0, el EZ2 lo tiene más estrecho que el EZ1, y así hasta el EZ4. Un espacio sensible estrecho es mejor si tan solo se desean detectar objetos directamente situados enfrente del sensor, y uno ancho es mejor si se quiere detectar cualquier objeto cercano. Maxbotix también ofrece una línea de sensores más precisos (los "XL") que tienen hasta una precisión de 1 cm, más rango de distancia y mejor supresión de ruido (es decir, que sus lecturas son menos tambaleantes). Sparkfun distribuye el sensor EZ0 con código de producto 8502, Adafruit con el código 979. Si se quieren encontrar otros sensores fabricados por Maxbotix en las tiendas online de estos distribuidores, basta con escribir "Maxbotix" en el buscador.

Los sensores GP2Yxxx

Existe una familia de sensores analógicos de distancia fabricados por Sharp y cuyo nombre empieza por GP2Y que, a diferencia de los anteriormente descritos, utilizan ondas infrarrojas. Su funcionamiento genérico es el siguiente: el chip contiene en su parte frontal un emisor y un receptor infrarrojos. Si el haz infrarrojo (que está modulado) impacta con un obstáculo, se dispersará en varias direcciones, pero también en la dirección donde está situado precisamente el receptor. Utilizando el cálculo de triangulaciones, a partir de la medida del ángulo de incidencia del haz infrarrojo sobre el receptor, se puede deducir la distancia del obstáculo. La salida de estos sensores será una tensión proporcional a la distancia calculada.

Este método tiene algunos inconvenientes. Por ejemplo: la luz del sol (que contiene infrarrojos), tanto directa como reflejada en grandes cantidades puede perturbar la lectura. Para minimizar este problema, es una buena práctica no utilizar lecturas individuales sino tomar un promedio de un número de valores como la lectura válida. Otro inconveniente que hay que tener en cuenta es el de la distancia mínima de nuestro sensor: existe un ángulo máximo más allá del cual los sensores infrarrojos ya no recibe correctamente el reflejo del haz, por lo que los valores medidos a distancias bajo ese mínimo tienen poco significado.

Elegiremos para su estudio concretamente el sensor GP2Y0A02. Este sensor lo podemos encontrar por ejemplo en Pololu con código de producto 1137. Su rango de medida está entre 20 y 150 centímetros y como conector incorpora uno de tipo JST de 3 pines, por lo que para enchufarlo a una breadboard necesitaríamos un cable tal como el producto nº 117 de Littlebird Electronics o el nº 8733 de Sparkfun, el cual consta por un lado de un zócalo JST de 3 pines y por otro están los tres cables libres sin terminal. Cada pin del sensor se corresponde con la alimentación (5 V), tierra y la salida analógica (a conectar a un pin-hembra de entrada analógica de la placa Arduino).

Este sensor devuelve un voltaje que es inversamente proporcional a la distancia: cuanta más distancia haya con el objeto, menos voltaje genera el chip (aunque la relación no es lineal). Este hecho lo podemos aprovechar para escribir sketches compuestos por un conjunto de “ifs” del estilo “si el voltaje medido está en un rango determinado de valores que ocurra algo y si está en otro, que ocurra otra cosa” si lo único que nos interesa es comparar diferentes distancias sin necesidad de saber su valor exacto.

Si lo que queremos, en cambio, es conocer la distancia real del obstáculo, podemos consultar en el datasheet del sensor un gráfico preciso del voltaje proporcionado por este en función de la distancia medida, por lo que a partir de los

valores mostrados en esta gráfica podríamos interpretar los valores de voltaje recibidos y mostrarlos como distancia.

Ejemplo 7.23: El siguiente sketch, no obstante, utiliza una fórmula para obtener la distancia. Esta fórmula se obtiene precisamente de los datos de la gráfica de su datasheet, pero no es exacta, así que hay que tomarla con precaución. Posiblemente sea necesario calibrar los datos numéricos que intervienen (en nuestro caso, el 65 y el -1,1) para acabar de perfilar las mediciones en nuestro caso particular.

```
float sensor = 0.0;
float distancia = 0.0;
void setup(){
       Serial.begin(9600);
}
void loop(){
      sensor = analogRead(0);
/*Convierto el dato obtenido por el conversor analógico-digital en un
valor de voltaje. Si el sensor se alimenta con 3,3V, hay que
sustituir el 5.0 por un 3.3 */
      sensor = sensor * 5.0 / 1024.0;
//Calculo la distancia a partir del voltaje obtenido
      distancia = 65*pow(sensor, -1.10);
      Serial.print(distancia);
      delay(250);
}
```

Una utilidad muy curiosa de los sensores de distancia es construir theremines caseros. Un theremin es un instrumento musical que posee un par de antenas metálicas que controlan el volumen y la frecuencia de la nota emitida, según nos acerquemos o alejemos de ellas. En lugar de dos antenas metálicas, utilizaremos un sensor de distancia; concretamente el G2PY0A02 (aunque lógicamente también podríamos haberlo implementado con otros modelos, tales como los ultrasónicos). La idea es obtener un valor interpretable como distancia a partir del valor de tensión leído de este sensor, y a partir de allí generar el sonido correspondiente.

Ejemplo 7.24: Una implementación concreta de theremin es el sketch siguiente, el cual reproduce diferentes notas musicales estándares según sea la distancia detectada (cuanto más cerca esté el obstáculo –nuestra mano–, más aguda será la nota). La onda de sonido se emite a través de un zumbador conectado al pin digital de salida nº 8 de nuestra placa Arduino. Ese sonido ha de durar un tiempo lo suficientemente breve (hemos puesto 125 milisegundos pero se puede cambiar) para

poder hacer un seguimiento en tiempo real de la detección del movimiento del obstáculo.

```
int lecturaSensor, nota;
//Array que guarda las frecuencias a reproducir
float frecuencias[] = {
  329.63, // Mi
  349.23, // Fa
  369.99, // Fa#
  392.00, // Sol
  415.30, // Sol#
  440.00, // La
  466.17, // La#
  493.83, // Si
  523.25, // Do
  554.36, // Do#
  587.33, // Re
  622.25, // Re#
  659.26, // Mi
};
byte numFrecuencias=13; //Número de elementos del array
void setup(){
       Serial.begin(9600);
       pinMode(8,OUTPUT); //Donde está conectado el zumbador
}
void loop() {
/*Cuanto más lejos esté el obstáculo, "lecturaSensor" menos valdrá
De todas formas, hay que saber que esta relación no es lineal*/
      lecturaSensor=analogRead(0); //Donde está conectado el sensor
/*Asocio "lecturaSensor" a una de las frecuencias predefinidas*/
      nota=map(lecturaSensor,0,1023,1,numFrecuencias);
/*Hago que a medida que el obstáculo se acerca, la nota es más
aguda*/
      tone(8,frecuencias[nota]);
/*Tiempo de reproducción mínimo de la nota */
      delay(125);
}
```

Los otros sensores GP2Yxxx son similares al GP2Y0A02: la mayor diferencia entre ellos es el rango de distancias que pueden medir y el nivel de voltaje aportado según esta, pero su funcionamiento es muy parecido. Tanto en Adafruit como en Sparkfun podemos encontrar por ejemplo el sensor GP2Y0A21 (con código nº 164 y nº 242, respectivamente) que puede medir distancias entre 10 y 80 cm.

El sensor IS471F

Este sensor no es un detector de distancia propiamente sino simplemente de presencia; concretamente detecta la existencia o no de un obstáculo entre 1 cm y 15 cm. Funciona emitiendo un haz infrarrojo y comprobando si le llega rebotado. Si es así, el sensor generará una señal LOW (que podrá ser leída por una placa Arduino conectada a él convenientemente) y si no se detecta ningún objeto, el sensor generará una señal HIGH.

El sensor consta de cuatro pines, los cuales son (si observamos su cara plana de frente, de izquierda a derecha): alimentación (5 V), detección de datos (a conectar a un pin de entrada digital de nuestra placa Arduino), tierra y emisión de señal infrarroja (a este lo llamaremos pin "X"). Se recomienda conectar también un condensador (de 0,33 µF) de tipo "by-pass" entre ese pin "X" y el pin de tierra.

Lo que más sorprende de este sensor es que no es capaz de emitir por sí mismo ninguna señal infrarroja, por lo que para que funcione es necesario conectar al pin "X" del sensor el cátodo un diodo LED infrarrojo externo (preferiblemente de 940 nm). La emisión de este LED es convenientemente modulada a 38 KHz por un modulador interno que contiene el sensor, de manera que este sea relativamente inmune a las interferencias causadas por otro tipo de luz como la del sol (el rebote es a su vez demodulado por un demodulador interno). El ánodo de ese LED infrarrojo deberá conectarse a la fuente de alimentación apropiada, generalmente a través de un divisor de tensión. Si ese divisor de tensión lo convertimos en un potenciómetro, podremos regular dentro de unos límites la distancia hasta la que se puede detectar el objeto, ya que cuanto más baja sea la resistencia de ese potenciómetro más intensa será la luz emitida por el LED y, por tanto, mayor será esa distancia.

Los sensores QRD1114 y QRE1113

El sensor QRD1114 (código de producto 246 de Sparkfun) en realidad no es más que un emisor de infrarrojos y un fototransistor bajo el mismo encapsulado, por lo que el principio de funcionamiento es similar a los sensores infrarrojos analógicos ya vistos: cuanta más distancia tenga el obstáculo, menos voltaje de salida obtendremos del sensor. Su característica más destacable es su rango de distancias, ya que solo es capaz de detectar objetos situados entre 0 y 3 cm de él. En realidad, este componente no está pensado para medir distancias exactas, sino tan solo para comprobar la proximidad de objetos por debajo de esos 3 cm.

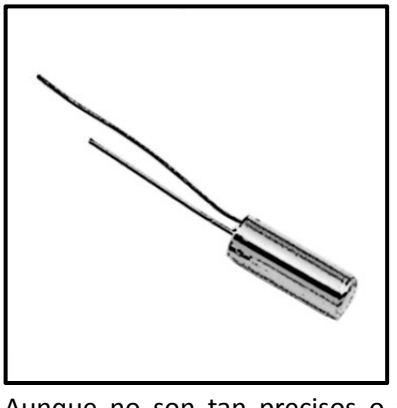
El código Arduino a utilizar con este sensor es idéntico al mostrado con el sensor LV-EZO: no se trata más que de obtener los valores de una entrada analógica. Sin embargo, las conexiones son algo más complejas: este sensor dispone de cuatro

patillas porque en realidad ya hemos comentado que se compone simplemente de un LED infrarrojo (ánodo y cátodo) y un fototransistor (colector y emisor). La patilla correspondiente al ánodo se ha de conectar a través de un divisor de tensión de 200 Ω al pin de alimentación de 5 V de nuestra placa Arduino, la patilla correspondiente al colector se ha de conectar a un pin de entrada analógica de nuestra placa Arduino y las otras dos patillas se han de conectar a la tierra común. Además, entre el ánodo y el colector se ha de conectar una resistencia de tipo "pull-up" cuyo valor puede oscilar entre 4,7 KΩ y 5,6 KΩ (según el que sea, los valores leídos por la placa Arduino cambiarán).

Este sensor también se puede utilizar además para detectar superficies blancas y negras (por lo tanto, podría ser utilizado en la construcción de robots seguidores de líneas) ya que las superficies blancas reflejan más luz que las negras, obteniéndose por tanto lecturas mayores. No obstante, existe un sensor más específico para realizar esta tarea concreta: el QRE1113.

El sensor QRE1113 es comercializado en forma de placa breakout por Sparkfun (producto nº 9453), la cual simplemente ofrece tres conectores: "VCC" (a enchufar a una fuente de 5 V), "GND" y "OUT" (a enchufar a una entrada analógica de nuestra placa Arduino). En realidad, este producto es la versión analógica de la placa breakout, ya que también se puede encontrar la versión digital. En la versión analógica, valores mayores para el voltaje leído significa que la luz emitida por el LED ha sido reflejada con mayor intensidad y por tanto que ha sido recibida mejor por el fototransistor (es decir, que se ha detectado una superficie clara).

SENSOR DE INCLINACIÓN



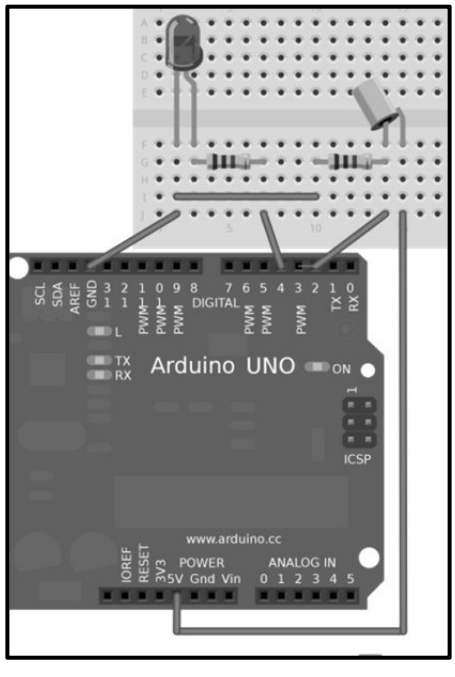
Los sensores de inclinación son pequeños, baratos, y fáciles de usar. Pueden trabajar con voltajes de hasta 24 V e intensidades de 5 mA. Constan de una cavidad y de una masa libre conductiva en su interior (como por ejemplo una bola metálica rodante); un extremo de la cavidad tiene dos polos conductivos de manera que cuando el sensor se orienta con este extremo hacia abajo, la masa rueda hacia los polos y los cierra. Por tanto, estos sensores actúan como interruptores, dejando o no pasar la corriente según la inclinación del circuito. Aunque no son tan precisos o flexibles como un completo acelerómetro, pueden detectar orientación o movimiento fácilmente.

Comprobar un sensor de inclinación es fácil: hay que poner el multímetro en modo de continuidad y conectar un cable cualquiera del multímetro (ya que los sensores de inclinación son dispositivos no polarizados) a cada borne del sensor. Seguidamente, hay que inclinarlo para determinar el ángulo en el cual el interruptor se abre y se cierra.

Si se conecta este sensor en serie a un LED (y su divisor de tensión preceptivo) y se alimenta el circuito, veremos cómo el LED se enciende o se apaga según la inclinación que le demos al diseño, tal como si estuviéramos utilizando un pulsador "invisible".

Si queremos usarlo con nuestra placa Arduino, debemos seguir la misma receta que empleamos cuando vimos los pulsadores: se pueden conectar utilizando una resistencia "pull-up" o "pull-down" (valores entre 200 Ω y 1 KΩ están bien).

Ejemplo 7.25: En el siguiente circuito mostramos un ejemplo básico, donde se puede apreciar que la lectura hecha por el sensor se recibe en un pin de entrada digital de la placa (hemos usado el nº 2) y se utiliza para controlar un LED conectado al pin digital de salida nº 4 (a través de su divisor de tensión), el cual se encenderá o no según el valor (HIGH o LOW) detectado por el sensor.



Y el código:

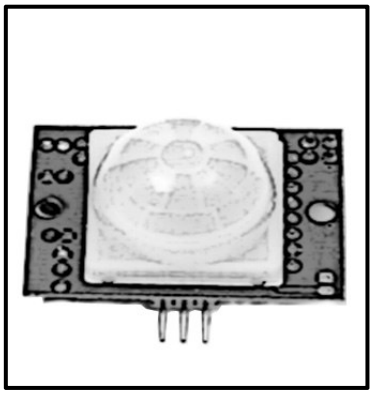
```
void setup(){
    pinMode(4,OUTPUT);
    pinMode(2, INPUT);
}
void loop(){
    digitalWrite(4,digitalRead(2));
}
```

Al sketch anterior se le podría añadir un control de “bouncing” como a los pulsadores, para aumentar la fiabilidad de las lecturas. Se deja como ejercicio.

SENSORES DE MOVIMIENTO

En este apartado hablaremos concretamente de los sensores “PIR” (del inglés “Pyroelectric passive InfraRed sensors”). La piroelectricidad es la capacidad que tienen ciertos materiales para generar un cierto voltaje cuando sufren un cambio de temperatura. Pero ojo, si su temperatura (sea alta o baja) se mantiene constante, ese voltaje poco a poco irá desapareciendo.

¿Y esto qué tiene que ver con el movimiento? Los sensores PIR básicamente se componen de dos sensores piroeléctricos de infrarrojos. Y todos los objetos emiten radiación infrarroja, estando además demostrado que cuanto más caliente está un objeto, más radiación de este tipo emite. Normalmente, ambos sensores detectarán la misma cantidad de radiación IR (la presente en el ambiente procedente de la habitación o del exterior), pero cuando un cuerpo caliente como un humano o un animal pasa a través del rango de detección, lo interceptará primero uno de los dos sensores, lo que causa un cambio diferencial positivo respecto el otro. Cuando el cuerpo caliente abandona el área de sensibilidad, ocurre lo contrario: es el segundo sensor el que intercepta el cuerpo y genera un cambio diferencial negativo. Estos pulsos son lo que en realidad el sensor detecta. Así pues, estos sensores son casi siempre utilizados para saber si un humano se ha movido dentro o fuera del (normalmente amplio) rango del sensor: alarmas de seguridad o luces de casa automáticas son un par de usos comunes para estos dispositivos.



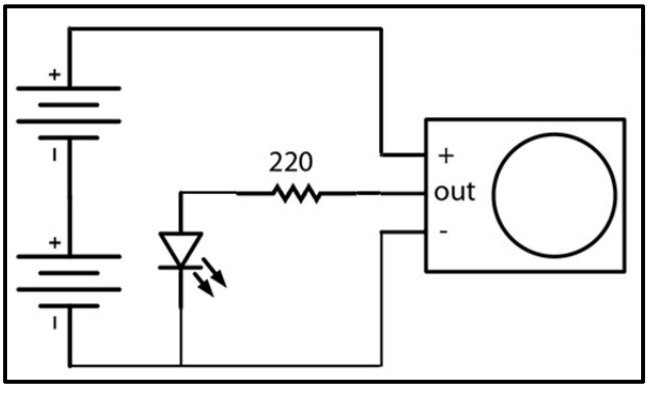
El sensor IR es básicamente un transistor FET con una ventana sensible a la radiación infrarroja en su cubierta protectora; cambios en el nivel de luz IR con una longitud de onda correspondiente al calor del cuerpo causan cambios en la resistencia fuente-drenador, que es lo que el circuito monitoriza. De todas formas, el gran truco de un sensor IR está en las lentes que incorpora: su función es condensar una gran área en una pequeña, proporcionando al sensor IR un rango de barrido mayor. De hecho, la calidad de las lentes es lo que básicamente diferencia un modelo de sensor PIR de otro, ya que la circuitería es bastante común a todos; cambiando una lente por otra se puede cambiar la amplitud y el patrón de sensibilidad del sensor.

Además de toda la circuitería ya comentada (sensores IR, lentes, etc.), un sensor PIR incorpora siempre un chip que sirve para leer la salida de los sensores IR y procesarla de tal manera que finalmente se emita un pulso digital al exterior (que es lo que nuestra placa Arduino recibirá).

La mayoría de sensores PIR vienen en plaquitas con 3 pines de conexión a un lado o al fondo: alimentación, tierra y señal de datos. El orden de estos tres pines puede variar según el modelo, así que hay que consultarlo en el datasheet (aunque la mayoría de veces cada pin ya tiene serigrafiada en la propia plaquita su función). La alimentación usualmente es de 3-5 V DC pero puede llegar a ser de 12 V, así que con una fuente de 5 V-9 V ya funcionan perfectamente.

Otras características comunes a la mayoría de modelos es que a través de su pin de datos emiten un pulso HIGH (3,3 V) cuando se detecta movimiento y emiten un pulso LOW cuando no. La longitud de los pulsos se determinan por los resistores y condensadores presentes en la PCB y difieren de sensor a sensor. Su rango de sensibilidad suele ser hasta una distancia de 6 metros y un ángulo de 110° en horizontal y 70° en vertical. La mayoría de modelos integran el chip BIS0001 para el control de la circuitería interna y el sensor IR RE200B; las lentes pueden ser muy variadas. Modelos concretos de sensores PIR son por ejemplo el producto nº 189 de Adafruit, el producto nº 8630 de Sparkfun o el producto 555-28027 de Parallax, entre otros.

Para probar el sensor PIR, podemos diseñar un circuito como el siguiente. Hay que tener en cuenta que el orden de los pines –alimentación, señal, tierra– es diferente según el modelo de sensor utilizado. Concretamente, el mostrado en el diagrama corresponde al modelo PIR de Adafruit, pero el orden de pines en el sensor 555-28027 es otro (concretamente, si lo miramos de frente, de izquierda a derecha tenemos los pines de señal, alimentación y tierra) y en el sensor de Sparkfun también (concretamente, si lo miramos de frente y de izquierda a derecha, tenemos los pines de señal, tierra, alimentación). En el sensor de Sparkfun es necesario además conectar también una resistencia "pull-up" de 10 KΩ entre su pin de alimentación y su pin de datos para que funcione correctamente; esto implica que, a diferencia de los anteriores, el sensor de Sparkfun envía una señal HIGH cuando no detecta movimiento y una señal LOW cuando sí.



La idea del circuito mostrado es que cuando el sensor PIR (suponiendo el modelo de Adafruit, que no usa resistencia pull-up externa) detecte el movimiento, el pin de datos enviará un pulso HIGH de 3,3 V y por tanto, iluminará el LED. Cuando el LED se apague, se puede probar de pasar la mano por delante, o directamente todo el cuerpo.

Hay que tener en cuenta que cuando se conectan las baterías, se ha de esperar entre medio minuto y un minuto a que el PIR se estabilice y comience a emitir datos fiables, así que al principio el LED puede parpadear un poco.

También hay que aclarar que el comportamiento del sensor PIR de Adafruit tiene la posibilidad de funcionar en dos modos según la posición de un jumper situado en el dorso de la placa. Si está colocado en la posición "L", el LED parpadeará a un ritmo de segundo a segundo mientras detecte movimiento, y si está colocado en la posición "H", el LED permanecerá encendido mientras detecte movimiento. El

sentido del modo “L” está en que (por poner un ejemplo) el sensor PIR puede estar conectado a algún otro dispositivo que se active cuando detecte un cierto número de parpadeos del LED.

Por su parte, si usamos el sensor PIR de Parallax, podemos especificar hasta qué distancia queremos abarcar (en dirección frontal al sensor) para detectar movimiento. Esto se hace mediante un jumper: si se coloca en la posición “S”, se detectarán movimientos que ocurran dentro de una distancia de 5 metros al sensor; si se coloca en la posición “L”, se detectarán movimientos que ocurran dentro de una distancia de 9 metros (en este modo, no obstante, pueden ocurrir falsos positivos).

Ejemplo 7.26: Conectar un sensor PIR (cualquiera de los mencionados) a nuestra placa Arduino es fácil: solamente hay que conectar el pin de datos del sensor a un pin-hembra de entrada digital, por ejemplo el nº 2 (además de a la alimentación de 5 V y tierra, obviamente). El sketch siguiente simplemente notifica por el canal serie cuándo se ha detectado movimiento.

```
int disparo= LOW;  //No hay movimiento al principio
int lectura = 0;
void setup() {
       pinMode(13, OUTPUT);
       pinMode(2, INPUT);
       Serial.begin(9600);
}
void loop(){
  lectura = digitalRead(2);
  if (lectura == HIGH) {
    digitalWrite(13, HIGH);
    if (disparo == LOW) {
      Serial.println("Movimiento detectado");
      //Solo queremos imprimir el mensaje una sola vez
      disparo = HIGH;
    }
  } else {
    digitalWrite(13, LOW);
    if (disparo == HIGH){
      Serial.println("Movimiento finalizado");
      //Solo queremos imprimir el mensaje una sola vez
      disparo = LOW;
    }
  }
}
```

El sensor ePIR

En Sparkfun (entre otros sitios) se puede adquirir con código de producto 9587 un sensor algo diferente llamado “ePIR”, del fabricante Zilog. La diferencia más importante entre este componente y los sensores PIR vistos anteriormente está en que el primero incluye además un microcontrolador propio dentro de su encapsulado. Esto permite una mayor flexibilidad a la hora de controlar el sensor y de gestionar los datos obtenidos.

Concretamente, se puede establecer comunicación con este componente de dos formas diferentes: en “modo hardware” y en “modo serie”. En el “modo hardware”, se puede ajustar la sensibilidad del sensor (es decir, a partir de qué valor detectado se considera movimiento) o el retardo (es decir, cuánto tiempo se esperará el sensor después de la detección de movimiento para volver a continuar detectando otro nuevo), entre otros parámetros.

El “modo serie”, por su parte, ofrece todas las funcionalidades del “modo hardware” pero además permite configuraciones más avanzadas mediante el envío de comandos específicos (como por ejemplo detectar movimiento en solo una dirección, ampliar el rango de detección de 3 m x 3 m a 5 m x 5 m y más). Por ejemplo, para forzar la detección de movimiento, deberemos enviar el comando “a”, el cual sirve para obtener la lectura del sensor en forma de carácter ASCII ‘Y’ (si hay movimiento) o ‘N’ (si no). De todas formas, para conocer todos los comandos disponibles y sus posibles usos, recomiendo consultar el datasheet del sensor.

Para seleccionar el “modo hardware”, en el momento de arrancar el sensor (o al salir de su modo de bajo consumo) se debe proporcionar una tensión menor de 1,8 V a su pin nº 4. Además, el valor concreto de esta tensión determina la sensibilidad del sensor, donde 0 V corresponde a la mayor sensibilidad y 1,8 V a la menor. Por lo tanto, si este pin se conecta directamente a tierra tendremos el “modo hardware” ya activado con la sensibilidad máxima posible. Si lo que se desea es regular dicha sensibilidad a mano, una opción sería conectar este pin a la patilla central de un potenciómetro (los extremos del potenciómetro en este caso deberían ir conectados a tierra y alimentación, respectivamente), utilizando los divisores de tensión pertinentes para obtener el rango 0-1,8 V deseado.

Para seleccionar el “modo serie”, en el momento de arrancar el sensor (o al salir de su modo de bajo consumo) se debe proporcionar una tensión mayor de 2,5 V a su pin nº 4. Una manera de conseguir esto es conectar una resistencia “pull-up” (generalmente de 100 KΩ) entre este pin nº 4 y la fuente de alimentación (que, atención, ha de ser de 3,3 V).

Los demás pines del sensor tienen una utilidad diferente según el modo de trabajo configurado. En el "modo hardware", las conexiones necesarias son:

Pin nº 1: este pin se ha de conectar a tierra.

Pin nº 2: este pin se ha de conectar a la fuente de alimentación. esta ha de ser de entre 2,7 y 3,6 V, por lo que el pin "3V3" de la placa Arduino es ideal.

Pin nº 3: este pin sirve para especificar el retardo del sensor (recordemos que es el tiempo que el sensor esperará desde que ha detectado movimiento hasta ponerse otra vez a detectar). En realidad, no es más que una entrada analógica que puede recibir desde 0V (correspondiente a un retardo de 2 s) hasta 2 V (correspondiente a un retardo de 15 m), por lo que se podría conectar por ejemplo a un potenciómetro para regular la tensión aplicada a mano. Si se conecta a tierra, el retraso será fijo de 2 s.

Pin nº 4: este pin sirve para seleccionar el tipo de modo de trabajo (hardware o serie) que se quiere utilizar. El procedimiento concreto se ha explicado en los párrafos anteriores.

Pin nº 5: este pin se ha de conectar a una entrada digital de nuestra placa Arduino. Mediante este pin se recibe un valor LOW si se detecta la existencia de movimiento, o un valor HIGH si no.

Pin nº 6: este pin no es más que una entrada analógica que debería recibir un valor de tensión proporcional a la cantidad de luz existente en el ambiente. La idea es activar la detección de movimiento solamente en entornos de poca iluminación (en horario nocturno, por ejemplo). Concretamente, cuando la tensión aplicada a este pin es menor de 1 V, la detección de movimiento está desactivada. Por eso, normalmente, este pin se conecta a un fotorresistor, aunque también se puede utilizar un potenciómetro para tener un control más directo. Si esta funcionalidad no se desea, bastará con conectar este pin directamente a la fuente de alimentación de 3,3 V.

Pin nº 7: este pin se ha de conectar a una salida digital de nuestra placa Arduino. Si se envía a este pin una señal LOW, el sensor pasará a un estado de bajo consumo ("sleep mode") y se inactivará. Este estado es útil cuando el sensor no se está utilizando. Cuando recibe una señal HIGH, vuelve a activarse.

Pin nº 8: este pin se ha de conectar a tierra.

En el caso de querer utilizar el "modo serie", las conexiones necesarias son:

Pin nº 1: mismo uso que en el "modo hardware".

Pin nº 2: mismo uso que en el "modo hardware".

Pin nº 3: este pin se ha de conectar al pin TX de la placa Arduino (o uno simulado con la librería SoftwareSerial). Sirve para recibir los comandos provenientes de esta.

Pin nº 4: además de servir para seleccionar el tipo de modo de trabajo (hardware o serie) que se quiere utilizar (el procedimiento concreto se ha explicado en párrafos anteriores), también sirve recibir las lecturas realizadas por el sensor, por lo que además se deberá conectar al pin RX de nuestra placa Arduino (o uno simulado con la librería SoftwareSerial).

Pin nº 5: este pin se ha de conectar al pin RESET de la placa Arduino.

Pin nº 6: mismo uso que en el "modo hardware".

Pin nº 7: mismo uso que en el "modo hardware".

Pin nº 8: mismo uso que en el "modo hardware".

Ejemplo 7.27: A continuación, mostramos un código de ejemplo de uso del "modo hardware", en el cual un LED (conectado a la salida digital nº 12 de nuestra placa Arduino) se ilumina cada vez que se detecta movimiento:

```
int sleepModePin = 4;    //Donde se conecta el pin n° 7 del sensor
int motionDetectPin = 2; //Donde se conecta el pin n° 5 del sensor
int lectura;
void setup() {
   pinMode(12, OUTPUT);
   pinMode(sleepModePin, OUTPUT);
   pinMode(motionDetectPin, INPUT);
   /*El pin "sleepModePin" ha de recibir una señal HIGH
   para activar la detección de movimiento */
   digitalWrite(sleepModePin, HIGH);
}
void loop() {
  lectura = digitalRead(motionDetectPin);
  if(lectura == LOW) {   //Se detecta movimiento
    digitalWrite(12, HIGH);
  } else {               //No se detecta movimiento
    digitalWrite(12, LOW);
  }
  //Me espero dos segundos para volver a detectar movimiento
  delay(2000);
}
```

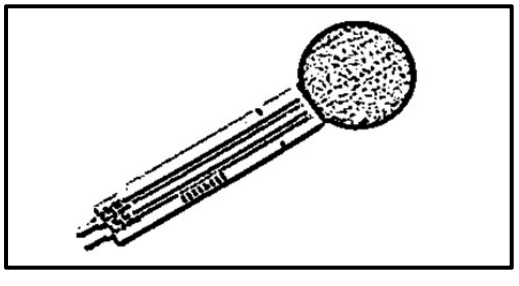
Ejemplo 7.28: A continuación, mostramos un código de ejemplo de uso del "modo serie", en el cual hemos sustituido el LED por un mensaje leído por el canal serie indicando si se ha detectado movimiento o no:

```
#include <SoftwareSerial.h>
int sleepModePin = 4;    //Donde se conecta el pin n° 7 del sensor
```

```
int lectura;
/*El pin 4 es el RX de la placa (conectado al n° 4 del sensor)
El pin 3 es el TX de la placa (conectado al n° 3 del sensor)*/
SoftwareSerial ePir = SoftwareSerial(4,3);
void setup() {
       pinMode(sleepModePin, OUTPUT);
       digitalWrite(sleepModePin, HIGH);
       /*La comunicación con el ePIR en modo serie
       ha de ser siempre a 9600 bits/s.*/
       ePir.begin(9600);
       Serial.begin(9600);
}
void loop() {
      //Ordeno la detección de movimiento
      ePir.print("a");
      //Mientras no se reciba respuesta del sensor, no hago nada
      while(!ePir.available()) {;}
      /*Muestro la respuesta del sensor:
      'Y' si hay movimiento y 'N' si no */
      Serial.print(ePir.read(););
      //Me espero dos segundos para volver a detectar movimiento
      delay(2000);
}
```

SENSORES DE CONTACTO

Sensores de fuerza



Estos sensores (también llamados FSRs, del inglés, "Force Sensitive Resistor") permiten detectar fuerza. Son básicamente un resistor que cambia su resistencia dependiendo de la fuerza a la que es sometido (concretamente, su resistencia disminuye a mayor fuerza recibida). Estos sensores son muy baratos y fáciles de usar pero no demasiado precisos: una misma medida puede variar entre un sensor y otro hasta un 10%. Así que lo que uno puede esperar de un FSR es conseguir medir rangos de respuesta; es decir: aunque los FSRs sirven para detectar peso, son mala elección para detectar la cantidad exacta de este. Sin

embargo, para la mayoría de aplicaciones sensibles al tacto, del tipo “esto ha sido apretado cierta cantidad” son una solución aceptable y económica.

Los FSRs están compuestos por una zona “activa” (de forma generalmente circular o cuadrada y de diferentes tamaños según el modelo), y dos terminales que, al ser este dispositivo una resistencia, no están polarizados. Normalmente, pueden soportar rangos de fuerzas de 0 a 100 newtons y el rango de resistencias que ofrecen van desde resistencia infinita cuando no detectan fuerza hasta aproximadamente 200 ohmios a máxima fuerza. En el datasheet del modelo concreto de FSR deberemos encontrar siempre cómo es esa relación “fuerza aplicada -> resistencia obtenida”, la cual no es exactamente lineal (ya que a pequeñas fuerzas hay una variación muy grande de la resistencia, y a fuerzas mayores la variación ya es menor).

Para comprobar su funcionamiento se puede utilizar un multímetro en modo de medida de resistencia. Apretando la zona sensible del FSR se deberá observar los cambios de resistencia.

Ejemplo 7.29: Si queremos utilizar este componente en un circuito con Arduino, debemos conectar un terminal a la alimentación y el otro a una resistencia “pull-down” (de 10 KΩ por ejemplo), la cual deberá ir a tierra. Además, un punto entre la resistencia “pull-down” (fija) y la resistencia FSR (variable) lo debemos conectar a una entrada analógica de la placa Arduino. Es el mismo montaje que ya vimos cuando hablamos de los LDRs y de los termistores, de hecho. La idea también es la misma que en los casos anteriores: leer el voltaje que hay en ese punto, que incrementa a medida que, por simple Ley de Ohm, la resistencia del FSR disminuye (es decir, a medida que se le aplica más fuerza).

A continuación, se presenta un sketch muy parecido a otros vistos anteriormente, el cual enciende de forma progresiva un LED según vayamos apretando más un FSR. En este ejemplo, ese LED está conectado (a través de su inseparable divisor de tensión) al pin de salida PWM nº 11. El FSR está conectado al pin de entrada analógica nº 0.

```
int lectura;
int brillo;
void setup() {
      pinMode(11, OUTPUT);
}
void loop() {
      lectura = analogRead(0); //Cuanta más fuerza, más voltaje leído
```

```
      brillo = map(lectura, 0, 1023, 0, 255);
      analogWrite(11, brillo); //A más fuerza, más brillo
      delay(100);
}
```

De forma muy similar podríamos realizar otro circuito muy interesante, consistente en un servomotor controlado mediante un FSR, de tal forma que el ángulo de giro de aquel fuera proporcional a la fuerza ejercida sobre este. El truco estaría en usar *map()* para mapear el rango de valores 0-1023 leídos por *analogRead()* al rango de valores 0-179, que es el admitido por *miservo.write()* (función que tendría que sustituir al *analogWrite()* del código anterior). Se deja como ejercicio.

Si lo que nosotros queremos saber en realidad es el valor concreto de la resistencia del FSR, podemos utilizar en nuestros sketches la siguiente fórmula: $R_{fsr} = (R_{pull} \cdot 1023/V_{convertido}) - R_{pull}$, donde $V_{convertido}$ es el valor leído por la entrada analógica de la placa Arduino una vez transformado por el conversor analógico/digital, R_{pull} es el valor de la resistencia "pull-down" (de valor fijo) y R_{fsr} es el valor de la resistencia del FSR que deseamos conocer. Esta fórmula es idéntica a las obtenidas anteriormente en el estudio de los fotorresistores y los termistores, ya que el razonamiento para obtenerla es el mismo.

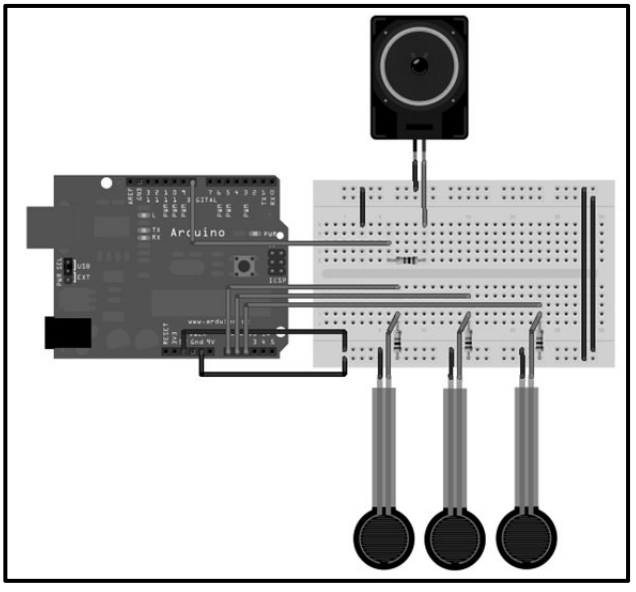
Conociendo R_{fsr} podríamos deducir la cantidad de fuerza recibida por el sensor, pero desgraciadamente no existe una fórmula analítica que relacione ambas magnitudes, por lo que no tenemos más remedio que consultar la gráfica del datasheet para conocer la relación exacta entre el valor de R_{fsr} calculado y el valor de la fuerza correspondiente.

Ejemplo 7.30: El siguiente sketch hace uso de la fórmula mencionada en el párrafo anterior para obtener la resistencia del FSR. El circuito es el mismo que el del sketch anterior, pero sin LED: usaremos el "Serial monitor" para leer los diferentes datos obtenidos.

```
int lectura;
const int rpull = 10000 //La resistencia pull-down es de 10KΩ
unsigned long fsr;  //Pueden ser valores muy altos
unsigned long cond; //Pueden ser valores muy altos
long fuerza;
void setup() {
       Serial.begin(9600);
```

```
}
void loop() {
      lectura = analogRead(0);
      if (lectura == 0) {
           Serial.println("No hay fuerza");
      } else {
           fsr=(rpull*1023/lectura) – rpull;
           Serial.print("Resistencia: ");
           Serial.println(fsr);
/*La gráfica del datasheet muestra la fuerza en función no de la
resistencia, sino de su inversa, la conductancia, medida en
microMhos.Por eso se ha de invertir convenientemente el valor de
fsr*/
           cond = 1000000 / fsr;
/*Y ahora usamos los valores de la gráfica para aproximar el valor
estimado de la fuerza. Dependiendo del modelo concreto de FSR los
valores en las condiciones de los "ifs" deberán ser diferentes */
           if (cond <= 1000) {
               fuerza = cond / 80;
               Serial.print("Fuerza en Néwtones: ");
               Serial.println(fuerza);
           } else {
               fuerza = (cond – 1000) / 30;
               Serial.print("Fuerza en Néwtones: ");
               Serial.println(fuerza);
         }
    }
    delay(100);
}
```

Ejemplo 7.31: Un ejemplo curioso de aplicación de FSRs (o de hecho, de cualquier otro sensor analógico) es el siguiente circuito. En él tenemos 3 FSRs conectados en paralelo a los pines de entrada analógica de la placa Arduino nº 1, 2 y 3 y a tierra a través de una resistencia "pull-down" de 10 KΩ. Además, tenemos un zumbador o altavoz conectado al pin digital de salida nº 8 (a través de un divisor de tensión de 100 ohmios).

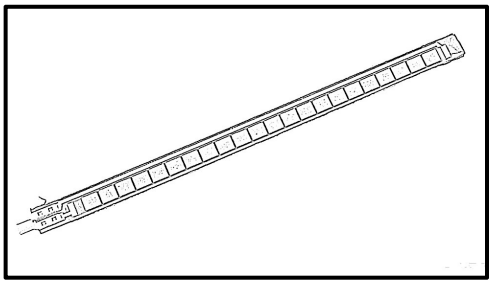


El código lo que hace es leer las lecturas de los tres sensores, cada uno de los cuales corresponde a una nota musical dentro de un array de notas. Si alguno de estos sensores se presiona más allá de un umbral, la nota correspondiente suena. Hemos implementado pues, un simple piano electrónico.

```
//Más allá del umbral sonará la nota
const int umbralminimo = 10;
//Cada nota corresponde a un sensor
int notas[] = { 220, 440, 880 };
void setup(){}
void loop() {
    int i;
    int lectura;
    //Voy recorriendo los sensores uno tras otro
    for (i = 0; i < 3; i++) {
         lectura = analogRead(i);
         //Si se presiona lo suficiente
         if (lectura > umbralminimo) {
               tone(8, notas[i], 20);
         }
    }
}
```

En Adafruit distribuyen un modelo de FSR con número de producto 166 y en Sparkfun venden varios con diferentes características: el nº 9375 y el nº 9376, pero todos son fabricados por Interlink (http://www.interlinkelectronics.com).

Sensores de flexión



Unos sensores parecidos a los FSR son los sensores de flexión (en inglés llamados "flex sensors" o "bend sensors"). Estos sensores están compuestos por una tira resistiva flexible solo en una dirección. Su resistencia cambia según cuánto sea arqueada: si están en equilibrio (es decir, sin combarse) su resistencia es mínima y cuanto más se flexiona más resistencia ofrece.

Al igual que las FSR, tienen dos terminales: uno podemos conectarlo dentro de nuestros circuitos a la fuente de alimentación (preferiblemente a través de un divisor de tensión) y el otro a una resistencia "pull-down" que va a tierra (también se podría usar la configuración alternativa usando "pull-up"s). En el punto donde se conecta el sensor a la resistencia "pull-down" se debe conectar una entrada analógica de la placa Arduino para leer el voltaje resultante en ese punto. Como ya sabemos, ese valor depende del valor de la resistencia del sensor, por lo que nos servirá para saber cuánto está flexionado.

Ejemplo 7.32: El siguiente sketch muestra un ejemplo muy básico de uso:

```
void setup(){
    Serial.begin(9600);
}
void loop() {
  int sensor, grados;
    sensor = analogRead(0);
/*Convierto el valor leído a grados de flexión. Los dos primeros
números de map() (768 y 853) son los valores leídos cuando el sensor
está completamente recto y cuando tiene una curvatura de 90 grados,
respectivamente. Estos valores los podemos haber consultado en el
datasheet o bien haberlos calibrado anteriormente. Los dos siguientes
números de map() son los grados a los que queremos mapear (0 grados y
ángulo recto)/*
  grados = map(sensor, 768, 853, 0, 90);
```

```
  Serial.print("Los grados de flexión son: ");
  Serial.println(grados,DEC);
  delay(100);
}
```

Sparkfun distribuye varios modelos de diferente longitud: el producto nº 10264 y el 8606 son dos ejemplos. Adafruit solo distribuye el segundo, con código 182. Además de la longitud, otras características importantes a tener en cuenta son su anchura y su peso, ya que muchas veces estos sensores se utilizan en proyectos textiles.

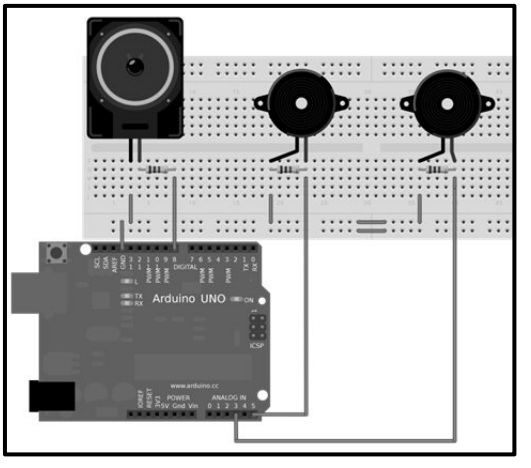
No confundir los sensores de flexión con los llamados sensores "Flexiforce", que es una marca registrada de Tekscan (http://www.tekscan.com). Los sensores "Flexiforce" son sensores FSR y Sparkfun ofrece unos cuantos que soportan diferentes rangos de fuerza (productos nº 11207, 8685, 8712 o 8713, entre otros).

Sensores de golpes

Debido a su constitución eléctrica interna, los zumbadores también pueden utilizarse, además de como emisores de sonidos, como sensores de golpes. El mecanismo es justo a la inversa del convencional: los golpes (suaves) recibidos por el zumbador se traducen en vibración de su lámina interna, la cual genera una serie de pulsos eléctricos que pueden ser leídos por una placa Arduino. De esta manera, podemos diseñar circuitos que respondan al tacto y que distingan incluso la presión ejercida. Como el zumbador es un dispositivo analógico, según lo fuerte que se golpee, la señal leída por la placa Arduino será de menor o mayor intensidad.

Ejemplo 7.33: Sabiendo esto, podemos diseñar el circuito de ejemplo mostrado a continuación, donde tenemos un altavoz y dos zumbadores. Tal como se puede ver, el altavoz está conectado a tierra y a la salida digital nº 8 (a través de una resistencia en serie de 100 ohmios), los terminales positivos de los zumbadores (si estos son polarizados) están conectados a las entradas analógicas nº 3 y nº 5, y los terminales negativos de los zumbadores (si estos son polarizados) a tierra. Además, cada zumbador está conectado en paralelo a una resistencia (de un valor recomendable de 1 MΩ), cuya función es actuar como resistencia "pull-down", manteniendo la entrada analógica a 0 V mientras el zumbador no sea presionado.

Si no tenemos ningún zumbador a mano, podemos conseguir el mismo efecto adquiriendo una lámina piezoeléctrica tal como el producto nº 10293 de Sparkfun, que no es más que un zumbador sin su recubrimiento, o adquiriendo el producto nº 10772, consistente en un kit de cuatro láminas como la anterior más varias resistencias de 1 MΩ. También podemos adquirir el "Sound & Buzzer Module" de Freetronics, que no es más que un zumbador incorporado a una cómoda plaquita breakout.

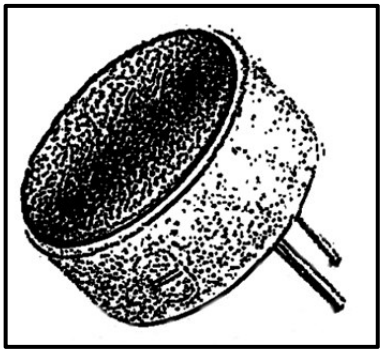


A partir de este circuito, podemos escribir un código como el siguiente. La idea es que según el zumbador que pulsemos, sonará por el altavoz una nota u otra. Pero además, cuanto más fuerte apretemos un zumbador, su nota sonará más tiempo. Esto es posible porque podemos conocer la presión ejercida sobre el zumbador: estos alcanzan una amplitud de vibración que depende de la magnitud del golpe recibido: a más presión, más amplitud. Y la amplitud se traduce proporcionalmente en voltaje, voltaje detectado por las entradas analógicas de la placa Arduino. Como precisamente el zumbador es un dispositivo analógico, la amplitud de su vibración al sufrir un golpe (es decir, el nivel de voltaje recibido) aumentará poco a poco, llegará a su máximo y volverá a decrecer hasta cesar. Por tanto, el truco para detectar la magnitud del golpe no está realmente en detectar el voltaje máximo obtenido (que puede ser un dato no muy preciso) sino en comprobar durante cuánto tiempo se mantiene el voltaje recibido por encima de un determinado umbral (decidido por nosotros): cuanto más tiempo transcurra entre la primera y última vez que se detecta un valor de voltaje, mayor de ese umbral, mayor habrá sido la magnitud del golpe.

```
int umbral = 100; //El zumbador se considera pulsado sobre ese umbral
int lectura1 = 0; //Lectura obtenida de un zumbador
```

```
int lectura2 = 0; //Lectura obtenida del otro zumbador
int tiempo = 0; //Tiempo en que las lecturas son superiores al umbral
int do = 1915;    //Semiperíodo de la onda de la nota "do"
int re = 1700;    //Semiperíodo de la onda de la nota "re"
void setup(){
        pinMode(8, OUTPUT);
}
void loop(){
        //Se detecta si está presionado un zumbador
        lectura1= analogRead(3);
        if (lectura1 > umbral) {
                tiempo=0;
/*Mientras se detecte que la presión continúa, se sigue leyendo la
entrada analógica para ver si aún se lee un valor por encima del
umbral. El tiempo transcurrido hasta que se lea un valor menor al
umbral marcará la duración de la nota a escuchar*/
                while (lectura1 > umbral) { tiempo++; }
/*Para evitar posibles ruidos no deseados y considerar el golpe
válido, se establece un tiempo mínimo de presión */
                if (tiempo > 100) { sonido(do, tiempo); }
        }
        //Se detecta si está presionado el otro zumbador
        lectura2= analogRead(5);
        if (lectura2 > umbral) {
                tiempo=0;
                while (lectura2 > umbral) { tiempo++; }
                if (tiempo > 100) { sonido(re, tiempo); }
        }
}
void sonido(int nota, int tiempo ) {
         unsigned long duracion;
         duracion = micros() + (35000 * tiempo);
/*Empiezo a contar desde el momento actual y añado un valor
arbitrario multiplicado por el valor de "tiempo"; así, cuanto más
fuerte se pulse el zumbador, más durará */
        //Mientras no se llegue al final
        while(micros() < duracion){
            digitalWrite(8, 255);
            delayMicroseconds(nota);
            digitalWrite(8, 0);
            delayMicroseconds(nota);
        }
}
```

SENSORES DE SONIDO



En realidad, un sensor de sonido no es más que un sensor de presión que convierte las ondas de presión de aire (las ondas sonoras) en señales eléctricas de tipo analógico; es decir, un micrófono. Existen muchos tipos de micrófonos según el mecanismo físico que utilizan para realizar esa conversión: los de tipo "inductivo" (también llamados "dinámicos"), los "de condensador", los piezoeléctricos, etc. Dependiendo del tipo, unos tendrán una mejor respuesta a un rango determinado de frecuencias de sonido que otros (es decir, que serán más "fieles" a la onda original), unos tendrán una mayor sensibilidad que otros (es decir, que ya generarán un determinado voltaje a menores variaciones de volumen detectadas), unos comenzarán a distorsionar a menores volúmenes que otros (es decir, que ofrecerán una THD menor para un determinado voltaje), unos serán más resistentes y duraderos que otros, etc.

No obstante, en nuestros proyectos con placas Arduino UNO la variedad de micrófonos a elegir se reduce drásticamente. Arduino UNO no es una plataforma pensada para el procesamiento de audio: ya hemos visto que (aunque existen proyectos destacables en el ámbito de la síntesis) la generación y emisión de sonido es ciertamente limitada. Y lo mismo ocurre con la recepción de sonido: para empezar, los pines-hembra de la placa no son capaces de recibir corriente AC (que es lo que son las señales de audio). Además, el conversor analógico-digital tarda como mínimo 100 microsegundos en realizar una lectura de una entrada, por lo que la máxima frecuencia de muestreo posible es de 10 KHz (es decir, una calidad relativamente baja). Además, el procesamiento de una señal acústica (compuesta en realidad de un conjunto de múltiples señales analógicas de diferentes frecuencias y amplitudes) es mucho más complejo de lo que el ATmega328P y su limitada memoria es capaz de realizar con solvencia.

Por eso, los micrófonos que se utilizan junto con las placas Arduino UNO en la mayoría de los casos son utilizados solamente como simples detectores de presencia y/o volumen de sonido (o como mucho, conectándolos a chips especiales como el MSGEQ7 de Mixed Signal Integration, podrían ser usados como detectores de frecuencias de sonido, pero esta posibilidad no la trataremos).

Concretamente, nosotros utilizaremos una variante de micrófono de tipo condensador llamado "micrófono electret". Este tipo de micrófono es

omnidireccional (es decir, detecta el sonido de todas direcciones y no solo el proveniente de un punto en particular) y por lo general tienen una sensibilidad buena (es decir, que el voltaje generado varía bastante acordemente según lo que varíe la intensidad del sonido). Además, son baratos y de un tamaño reducido. Tienen mejor respuesta a frecuencias de rango medio-alto, por lo que son mejores para comunicaciones de voz que para música de sección rítmica importante, por ejemplo. En muchos dispositivos domésticos como teléfonos móviles, computadores o auriculares los micrófonos que vienen incorporados son de ese tipo.

Sea cual sea la aplicación práctica que le demos a un micrófono, en todo caso su uso implica necesariamente el uso de un pre-amplificador. Esto es debido a que la señal generada por un micrófono tiene una amplitud demasiado pequeña (generalmente entre 0 y 100 milivoltios) para poder ser aprovechada por nuestra placa Arduino. Por tanto, no podremos conectar un micrófono directamente a una entrada analógica de nuestra placa Arduino tal como hemos venido haciendo con otros sensores analógicos, sino que deberemos incluir un pre-amplificador entre el micrófono y la placa Arduino. En este sentido, hay que distinguir entre pre-amplificación (conversión de la señal generada por el micrófono para llevarla a un nivel usable por el circuito (en este caso, la placa Arduino) y amplificación (conversión de esa señal usable internamente en nuestro circuito a un nivel lo suficientemente audible para poder emitirlo a través de altavoces).

De hecho, este proceso de pre-amplificación y amplificación también es necesario cuando el micrófono lo conectamos a un sistema de audio doméstico o profesional: todos los dispositivos intermediarios (desde los reproductores portátiles de ficheros “mp3” o similares hasta cadenas de alta fidelidad pasando por televisores, reproductores de DVD, mesas de mezclas, etc.) trabajan a unos niveles de tensión superior al aportado por cualquier micrófono (el llamado “nivel de línea”, diferente a su vez del utilizado por Arduino) por lo que la señal generada por este ha de ser pre-amplificada para poder ser utilizada por todos estos dispositivos de audio. Una vez esta señal eléctrica ya está a nivel de línea, para transformarla en sonido real y poderlo emitir por algún sistema de altavoces es necesario entonces amplificarla hasta alcanzar la potencia deseada.

Plaquitas breakout

Existen plaquitas breakout que incorporan un micrófono electret y un pre-amplificador todo en uno, de manera que podamos empezar a utilizar el kit completo micrófono+pre-amplificador al instante. Un ejemplo es el “Microphone Sound Input Module” de Freetronics. La plaquita consta de cuatro conectores: “VCC” (a conectar a

la alimentación de 5 V proporcionada por nuestra placa Arduino), “GND” (a conectar a tierra), “MIC” (salida analógica a conectar a una entrada analógica de nuestra placa Arduino) y “SPL” (otra salida analógica a conectar a otra entrada analógica de nuestra placa Arduino). En nuestros proyectos podremos utilizar las señales recibidas por ambos canales (“MIC” y “SPL”) pero lo normal es conectar y utilizar solo uno de ellos, según lo que nos interese: el canal “MIC” proporciona una señal amplificada lo más fiel posible a la señal acústica, por lo que es más útil cuando se desea realizar un procesado del audio; en cambio, el canal “SPL” (de “Sound Pressure Level”) simplemente ofrece un voltaje que es proporcional al volumen de sonido recibido, por lo que nos servirá para detectar fácilmente la existencia de sonido y su “cantidad”, que es lo que generalmente desearemos.

Ejemplo 7.34: El siguiente sketch lee el valor SPL proveniente de esta plaquita cinco veces por segundo y muestra la lectura por el canal serie: a más voltaje leído, más volumen de sonido.

```
//La salida SPL está conectada al pin de entrada analógico 0
const byte splSensor = 0;
void setup() {
 Serial.begin(9600);
}
void loop() {
  Serial.println(analogRead(splSensor));
  delay(200);  //Evito la sobrecarga del canal serie
}
```

Una plaquita similar a la anterior pero que solo ofrece una salida de tipo “SPL” es el producto nº DFR0034 de DFRobot. Otra plaquita similar que también ofrece solamente una salida de tipo SPL es ZX-Sound de Inex Robotics.

Por otro lado, también existen plaquitas cuya salida es digital: si detectan un sonido superior a un umbral (generalmente definido a través de un potenciómetro), envían una señal HIGH a la entrada digital de la placa Arduino donde estén conectadas, y si el sonido detectado es inferior a ese umbral, envían una señal LOW. Estas plaquitas son útiles cuando no se desea monitorizar el volumen del entorno, sino tan solo determinados sonidos con un volumen superior al resto (una aplicación práctica es detectar colisiones en robots, por ejemplo). Un ejemplo de plaquita de este estilo es el producto nº 29132 de Parallax; tan solo consta de tres conectores (5 V, GND y señal) y mediante un potenciómetro que lleva incorporado podemos calibrar la sensibilidad del sensor, de manera que el sonido umbral que separa el envío de una señal HIGH (se detecta ruido) de una LOW (no se detecta ruido) se

amolde a nuestras necesidades. Otra plaquita muy similar es la llamada "Sound Sensor TTL output" de Cutedigi.

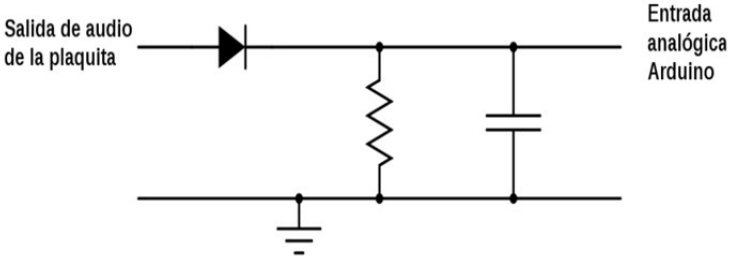
Ejemplo 7.35: A continuación, presentamos un código de ejemplo de uso muy sencillo para ambas plaquitas:

```
int lectura = 0;
void setup() { Serial.begin(9600); }
void loop() {
  //La plaquita está conectada a la entrada digital n° 2 de Arduino
  lectura = digitalRead(2);
  if (lectura == HIGH){ //Si se detecta un sonido más allá del umbral
    Serial.println("Sonido detectado");
    delay(100);
}
```

Una plaquita que tan solo ofrece una salida de tipo "MIC" (es decir, una señal eléctrica que reproduce el comportamiento de la onda acústica) es el producto nº 9964 de Sparkfun. Una vez conectada esta salida "MIC" a la entrada analógica de la placa Arduino (además del conector "VCC" a una fuente de 5 V y el conector "GND" a la tierra común), si observamos los valores recibidos por el "Serial monitor" veremos que cuando no se detecta sonido se mantienen estables alrededor del valor 512 y cuando hay ruido fluctúan por encima y por debajo de ese valor central (siendo mayor esta desviación –tanto por arriba como por abajo– cuanto mayor sea el volumen del ruido detectado). Por tanto, si quisiéramos obtener el volumen (es decir, utilizar esta plaquita como un simple sensor SPL), deberíamos utilizar la expresión `volume=abs(analogRead(0)-512);` (suponiendo que la entrada analógica utilizada es el pin-hembra nº 0). Pero ¿por qué este comportamiento?

Porque lo que hace esta plaquita es "trasladar" el valor de la salida correspondiente a 0 V (el valor central de la onda) a un valor central de 2,5 V y con él, el resto de valores en bloque sin alterar por tanto la forma de la onda. En otras palabras: añade un "colchón" de 2,5 V (DC) sobre el cual viaja la señal de audio (AC). Esto es lo que se llama tener una señal "descentrada" (o "biased", en inglés). El objetivo es hacer que la placa Arduino pueda detectar tanto los valores positivos de la onda acústica como los negativos. Si no existiera el descentramiento de la señal, los valores negativos estarían realmente por debajo de 0 V y entonces la entrada analógica de la placa Arduino (que solo admite corriente DC y no AC) no podría detectarlos correctamente. Con la señal descentrada, el pico positivo podrá llegar a un máximo de 5 V (es decir, podrá tener una amplitud máxima de 2,5 V a partir de la señal base descentrada de 2,5 V) y el pico negativo a un mínimo de 0 V (es decir, podrá tener una amplitud máxima de -2,5 V a partir de la señal base descentrada de 2,5 V).

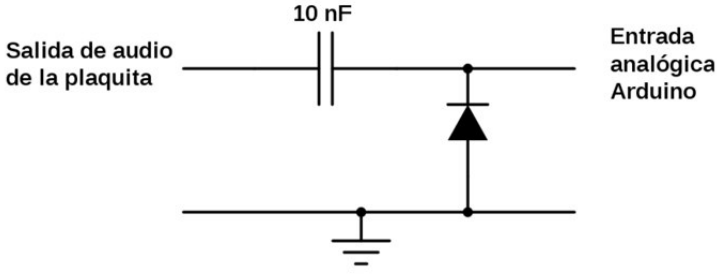
Otro truco que podemos utilizar con una salida de tipo “MIC” (tanto de la plaquita de Freetronics como la de Sparkfun) es distinguir si el sonido detectado es un sonido puntual (un portazo, por ejemplo) o bien un sonido continuado (como una conversación, por ejemplo). Debido a que la placa Arduino UNO es relativamente lenta a la hora de tomar muestras, es posible que no detecte todas las variaciones de volumen en tiempo real y se “deje por detectar” picos de sonido que aparezcan entremedio de dos lecturas. Para solventar esta cuestión, se puede acoplar a la salida “MIC” un circuito llamado genéricamente “detector de envolvente” (en inglés, “envelope detector”), mostrado a continuación.



El objetivo de este circuito es hacer llegar a la placa Arduino tan solo los valores extremos de cada pico positivo, uniéndolos entre sí de una forma suave, sin tener en cuenta por tanto toda la vibración de pico a pico. La figura de la señal recibida (la “envolvente”) ayudará a la placa Arduino a “ir siguiendo” las variaciones de volumen: si la envolvente contiene picos estrechos habremos detectado un ruido brusco, y si es al contrario, estaremos detectando un sonido más o menos continuo. Pero ¿por qué este circuito funciona así?

En los picos positivos, el diodo deja pasar la señal, cargando el condensador hasta (casi) el valor del pico. En los picos negativos el diodo está polarizado inversamente, no permitiendo que fluya la corriente, por lo que el condensador reaccionará descargándose lentamente a través del resistor (la entrada de Arduino está diseñada para atraer una cantidad de corriente despreciable). Cuando vuelve a aparecer un pico positivo, el diodo vuelve a dejar pasar la señal y el condensador se vuelve a cargar rápidamente, y así todo el rato. La velocidad de descarga del condensador viene definida por el producto R·C (llamado “constante de tiempo”), y su valor dependerá del tipo de proyecto que deseemos realizar, pero un valor típico para empezar a probar es 0,1.

Otra cosa que podemos hacer con la salida “MIC” es convertirla en una salida “SPL”. En el caso de la placa Freetronics no nos será necesario porque ya tenemos los dos tipos de salidas, pero en el caso de la plaquita de Sparkfun, el truco está en acoplar a la salida “MIC” un circuito llamado genéricamente “fijador de nivel positivo” (en inglés, “positive unbiased clamper”), mostrado a continuación.



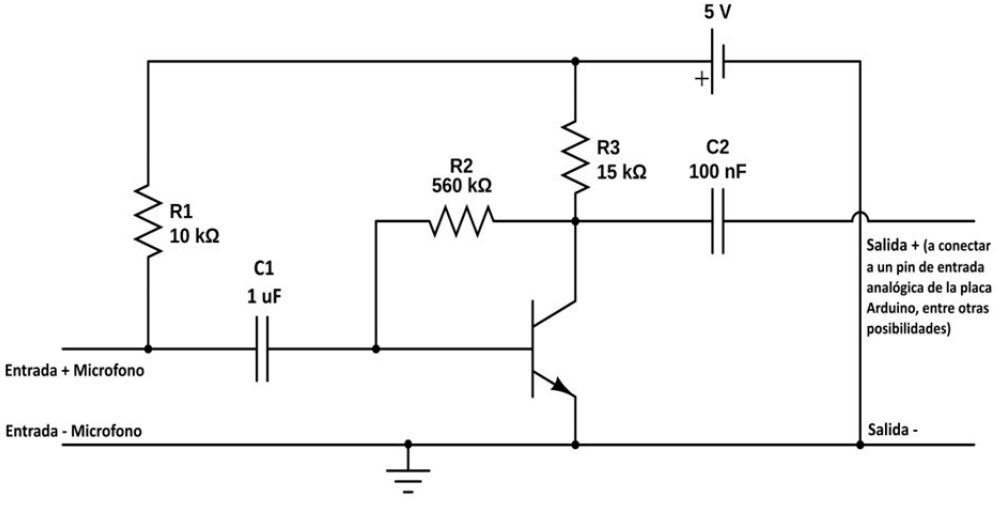
El objetivo del circuito anterior es descentrar la señal AC de tal forma que, sin modificar la forma de su onda, los extremos de los picos negativos tengan siempre justo un valor mínimo de 0 V. Si el volumen de sonido cambia, la longitud de los picos de la señal también, pero como lo que está fijado es que el valor mínimo sea siempre 0 V, lo que ocurrirá es que el valor central se deberá desplazar para cumplir esta condición, con lo que cambiará el valor extremo por el otro lado, por el pico positivo: cuando el volumen aumente, el valor extremo del pico positivo aumentará y cuando el volumen disminuya, el valor extremo del pico positivo disminuirá también. Y esto es lo que mediremos. Se recomienda que el diodo sea de tipo “germanio” (el 1N34A sería un buen ejemplo) y en este caso, se recomienda también alimentar la plaquita con 3,3 V en vez de 5 V. Si se hace así, los valores recibidos por la entrada analógica de la placa Arduino oscilarán entre 0 y 750/800 (a más volumen, mayor valor máximo recibido). Podremos entonces escribir un sketch que reaccionara a sonidos por encima de un umbral determinado, por ejemplo, simplemente observando los valores analógicos máximos recibidos.

Circuitos pre-amplificadores

Es posible que solo dispongamos de un micrófono electret en vez de toda una plaquita breakout. Si queremos conectarlo a nuestro circuito, lo primero que debemos saber es que los micrófonos electret (como el producto nº 8635 de Sparkfun, por ejemplo) deben ser alimentados. Son además dispositivos polarizados, en los cuales su terminal negativo suele estar marcado mediante una muesca o señal. En principio, este terminal negativo debería ser conectado a tierra y el terminal positivo debería ser conectado a una fuente de alimentación, la cual puede ser cualquiera capaz de aportar un voltaje de entre 2 V y 5 V (si es necesario, a través de

un divisor de tensión). La señal de audio recibida la obtendríamos del terminal positivo, siempre a través de un condensador (de entre 0,1µF y 1µF) actuando como filtro pasa-altos para eliminar cualquier colchón DC de la señal AC obtenida.

Desgraciadamente, conectar un micrófono electret no es tan sencillo porque ya sabemos que es necesario pre-amplificar la señal recibida para hacer que esta sea usable. Por tanto, deberemos construir un circuito accesorio alrededor de él que realice esta función. Un ejemplo muy sencillo es el siguiente, en el cual tan solo hemos utilizamos unas cuantas resistencias y un transistor NPN (como por ejemplo el 2N3904), cuya función es precisamente amplificar la señal, tal como estudiamos en el capítulo anterior dentro del apartado de generación de sonidos mediante *tone()*):



La resistencia R1 ejerce como divisor de tensión del micrófono electret y el condensador C1 sirve, tal como ya hemos comentado, para eliminar el posible colchón DC de la señal AC recibida. A partir de aquí, el funcionamiento lo deberíamos conocer: la señal recibida por el micrófono se envía a la base del transistor, el cual dejará fluir más o menos corriente entre colector y emisor según sea la intensidad de aquella. Si esa corriente generada la tomamos como salida del circuito, tendremos una representación calcada de la señal original pero con mayor amplitud.

La función de las resistencias R2 y R3 es descentrar la señal AC recibida por la base del transistor proveniente del micrófono. Dicho de otra forma: añaden a esa señal un "colchón" DC constante que permite tener a todos los valores de la onda AC

(incluyendo los picos negativos) por encima de 0 V. En caso de ausencia de sonido, la base del transistor recibirá una determinada corriente DC que mantendrá el transistor en un estado de conducción intermedio permanente llamado "punto-Q"; cuando se detecte sonido, las oscilaciones de la señal fluctuarán alrededor de ese estado de conducción intermedio, sin llegar nunca ni al modo de saturación por un lado ni al modo de corte por otro (debido a que la señal se autorregula: si la corriente por el colector incrementa, decrece la que circula por la base, y por tanto automáticamente la corriente por el colector pasa a reducirse). Por otro lado, debido a la aparición de este nuevo colchón DC, una vez obtenida la señal ampliada es necesario eliminarlo en la medida de lo posible mediante el condensador C2.

Los valores de R2 y R3 deberían elegirse con cuidado, porque de ellos depende fundamentalmente la amplitud de la señal amplificada, la cual deberá ser una u otra según lo que nos interese (nivel de línea, tensión de trabajo de Arduino, etc.). De hecho, el uso más habitual de este circuito (y de todos los pre-amplificadores) es adecuar las señales al nivel de línea para que puedan derivarse a circuitos amplificadores de audio propiamente dichos.

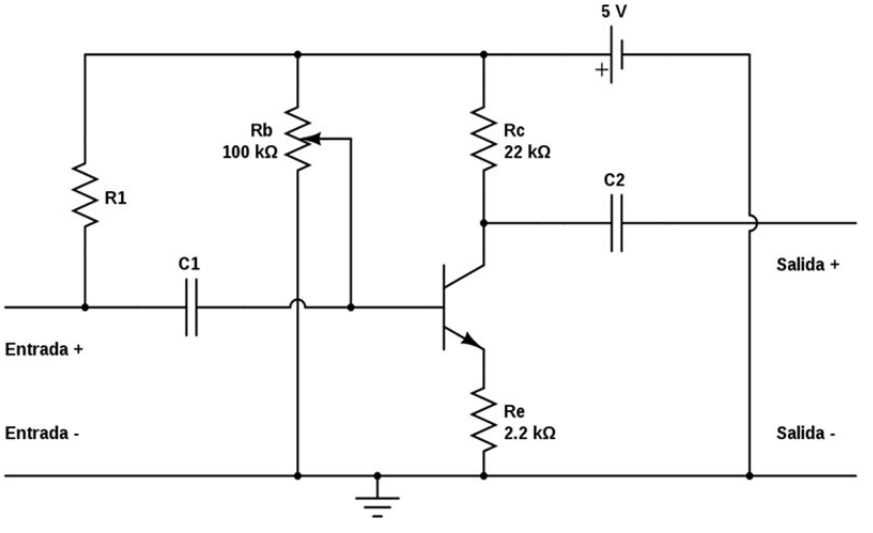
Si conectamos este circuito pre-amplificador a nuestra placa Arduino, y leídos los valores de R2 y R3 mostrados en el diagrama anterior, podremos observar mediante el "Serial monitor" la presencia de un colchón DC de 320 (sobre 1024) y picos de hasta 950 en sonidos muy fuertes. Estos valores ofrecen un rango útil (dentro del rango 0-1023 admitido por las entradas analógicas de la placa Arduino) bastante aceptable, pero si se desea, se pueden probar otros valores de R2, R3 y C2 para mejorarlo (es decir, para reducir más el valor del colchón y aumentar más el rango útil dentro del admitido).

Ejemplo 7.36: Para probar el circuito pre-amplificador anterior podemos ejecutar el código siguiente, el cual nos puede servir, en un entorno silencioso, para observar las fluctuaciones presentes en la señal aun cuando en teoría debiéramos tener una señal constante. Para evitar este fenómeno, podemos utilizar el recurso de calcular la media de los últimos valores leídos (tal como hemos visto en ejemplos anteriores) y así suavizar los picos no deseados.

```
void setup(){
 Serial.begin(9600);
}
void loop(){
 int mini = 1024; //Iré reduciendo el valor de "mini" hasta el real
 int maxi = 0;    //Iré reduciendo el valor de "maxi" hasta el real
```

```
 for(int i=0;i<1000;i++) { //Otra posibilidad: while(millis() < 1000)
       int valor = analogRead(0); //El micro está conectado en pin 0
       mini = min(mini, valor);
       maxi = max(maxi, valor);
 }
 Serial.print("Ruido=");
 Serial.println(maximum-minimum);
}
```

El circuito pre-amplificador presentado no es ni mucho menos el único que existe. Hay literalmente cientos. Otro circuito común (tal vez incluso más que el anterior), es el de la figura siguiente. En él se utiliza un potenciómetro como divisor de tensión para ajustar la corriente deseada a la base del transistor. Lo más práctico es calibrarlo en un entorno silencioso para obtener un voltaje de salida de 2,5 V correspondiente al punto-Q.



Tanto este circuito como el anterior tienen la ventaja de amplificar la señal sin distorsión, pero tiene el inconveniente de que aporta constantemente un colchón DC a la base del transistor, con lo que no es un circuito eficiente energéticamente.

Finalmente, no quisiera dejar de comentar que en muchos foros y blogs de Internet se sugiere utilizar como pre-amplificador un chip bastante popular, el LM386. No es una buena elección, ya que en realidad, el chip LM386 es un amplificador que trabaja a nivel de línea. Es decir, está pensado para aportar, a partir de una entrada

de audio ya pre-amplificada, una potencia de hasta 1 W (en sus versiones más capaces) a un sistema de altavoces. Si se conecta directamente a un micrófono, obtendremos demasiado ruido; una alternativa mejor sería utilizar en todo caso el chip LM358.

Tampoco es buena idea (como a menudo se propone) usar el LM386 como amplificador conectándolo directamente a una salida de audio de nuestra placa Arduino, ya que, como acabamos de decir, este chip está pensado para tener entradas a nivel de línea, las cuales admiten voltajes menores que los 5 V que ofrece un pin-hembra de la placa Arduino.

Reconocimiento de voz

Nuestra placa Arduino es capaz de responder a órdenes expresadas mediante voz gracias al "EasyVR Shield" de Veear (distribuido entre otros por Sparkfun con producto nº 10963), el cual incorpora un módulo de reconocimiento de voz diseñado y fabricado por la misma empresa Veear (también disponible de forma autónoma). Este shield también incluye un micrófono, una salida para conectar un altavoz de 8 ohmios y un zócalo jack de 3,5 mm para conectar unos auriculares.

Este shield se comunica con la placa Arduino a través del canal serie (preferiblemente mediante dos pines RX y TX definidos por software) a una velocidad de 9600 bits/s. Aunque podríamos controlar este shield enviando los comandos adecuados desde un sketch, para gestionar su funcionalidad y comportamiento es mucho más sencillo utilizar la librería que el propio fabricante ofrece en http://www.veear.eu/downloads. La documentación necesaria para aprender su uso viene incluida dentro del paquete descargado.

Este shield incluye de fábrica un conjunto de órdenes predefinidas disponibles en varios idiomas (entre ellos el español), ideales para realizar controles básicos, pero también admite la definición de hasta 32 comandos propios totalmente personalizados, incluyendo contraseñas. Para definir estos comandos y/o configurar las órdenes pregrabadas es necesario utilizar el software gráfico (llamado "EasyVR Commander") que el propio fabricante ofrece (aunque tan solo para sistema Windows) en http://www.veear.eu/downloads. La información necesaria para aprender su uso la podemos consultar en la Guía de Usuario, disponible en el mismo sitio de descargas.