OPEN
+ SOFTWEAR.

2nd edition
revised & expanded

OPEN + SOFTWEAR™

Fashionable prototyping and wearable computing using the Arduino.

Tony Olsson | David Gaetano | Samson Wiklund | Jonas Odhner

Prólogo & Prefacio

Primera parte: Aspectos Basicos

Segunda parte:

Ejemplos

Tercera parte:

Programación

Prólogo

Los textiles electrónicos quieren algo de nosotros.

In the Dando vida a los tejidos. Tradicionalmente los textiles son una parte fundamental de la vida humana. En forma de ropa que cubre nuestro cuerpo y a través de su diseño interior, son parte de nuestro día a día. ¿Que ocurriría si empezásemos a integrar electrónica en los textiles? Me gusta considerar que la electrónica tiene propiedades “frankestinianas”. Cuando conectas materiales tradicionales con la electrónica puedes dar vida a objetos inertes. La electrónica nos permite dar a los tejidos una rudimentaria forma de inteligencia, permitiéndoles sentir el entorno y responder a el.

Parece que estamos liderando un época de computación ubicua donde muchos de nuestros objetos, de una forma o de otra, son parte de nuestro intercambio digital de información. No podemos imaginar una vida sin tecnología. En consecuencia, la discusión sobre qué es tecnología y qué debería ser en el futuro es ahora esencial. Si decides empezar a crear textiles electrónicos y “vestibles”, estarás automáticamente formando parte de esta discusión. ¡Enhorabuena y bienvenido! Estamos deseando oír tu voz.

Este libro es un punto de partida genial y una magnifica referencia de experimentos en el campo de las ropas y textiles basados en Arduino. Aunque aquí encontrarás algo de teoría, en el fondo este libro es práctico. Creo que Open Softwear tiene la capacidad de educar artistas y diseñadores hasta el nivel donde pueden empezar a crear sus propios diseños interactivos para el futuro. Como diseñador de textiles electrónicos puedo decirte que es un campo complicado, pero muy divertido. El proceso requiere que constantemente estés cambiando tu atención entre el diseño y los problemas técnicos, así que no te vas a aburrir nunca. Un montón de caminos están todavía sin asfaltar y la aventura sigue estando ahí. Las oportunidades están para ser aprovechadas por gente con cerebro, imaginación y mucha perseverancia. Asegúrate de documentar tu trabajo y compartir tus diseños en internet con el resto. El campo de los textiles electrónicos crece con cada proyecto acabado.

Mi consejo para artistas y diseñadores principiantes en "vestibles": Se curioso y paciente; aprende por ti mismo diseño textil y electrónica; pide opinión; encuentra asesores que entiendan tu trabajo conceptual, visual y técnicamente; Contacta con otras personas que hagan "vestibles"; y por último, mantén a raya los niveles de voltaje y ¡diviértete!

Melissa Coleman

Melissa Coleman es una artista de La Haya (Holanda). Enseña sobre textiles electrónicos en la Royal Academy of Art de La Haya y es profesora en "Wearable Senses" en la Technical University of Eindhoven. Es co-fundadora de V2_'s E-Textile Workspace en Rotterdam y blogger invitada de Fashioning Technology

Retos de diseño con realidades inacabadas

Cuando creamos nuevos objetos y experiencias para otros, los diseñadores imaginamos que somos usuarios. Nos anticipamos a los aspectos ergonómicos del dispositivo, así como a los placeres asociados a explorar el mundo estéticamente a través de ellos. El proceso de probar físicamente formas, colores, materiales y, últimamente, patrones de interacción, es lo que llamamos "prototipar".

Imaginar cosas es fácil, hacerlas realidad es más complicado. La principal habilidad de un diseñador es ilustrar conceptos mediante cualquier cosa a su alcance. Invito a mis estudiantes a buscar a su alrededor artefactos que de alguna manera se parezcan al dispositivo al que aspiran. Les pido que lo desmonten, que miren en su interior, que modifiquen su funcionalidad para acercarse tanto como sea posible a lo que están buscando y que lo vuelvan a construir con su propio envoltorio. La mayoría del tiempo, las ideas se ven comprometidas, y empieza a ser parte del trabajo del estudiante encontrar la manera correcta de comunicar que es lo que están buscando en comparación con lo que lograron.

Las cosas no funcionan de forma diferente en el mundo real. Las empresas a veces crean prototipos que permiten a sus clientes probar un artefacto no existente. Hay un montón de dispositivos que pueden ser ajustados para proporcionar una falsa experiencia. Sabiendo como mezclar tecnología y artesanía es ahora más relevante que nunca. Estos híbridos llevan la tecnología digital a todos los aspectos de nuestras vidas.

La mezcla de circuitos digitales con ropa es lo que llamamos "computación vestible". Esto lleva la tecnología tan cerca como es posible al cuerpo sin ser ser intrusiva. Es un área emergente donde los diseñadores tienen mucho que decir. Hay un espacio para la exploración en el diseño, para enloquecer, y para crear aplicaciones en el mercado de las tecnologías existentes. Hay un montón de productos para ser imaginados, prototipados y fabricados.

La primera edición del libro Softwear fué creado por estudiantes de la Malmö University -no profesionales- buscando como mezclar electrónica y moda. Querían ayudar a otros a prototipar nuevas prendas con circuitos digitales. El arte de prototipar electrónica fue durante mucho tiempo reservado solamente a ingenieros. Herramientas como Arduino, Lilypad y otras muchas, simplifican el acceso a la tecnología para el prototipado.

Para aprender como navegar por este mundo de lo inacabado, es importante convertirse en un diseñador capaz de dar forma al futuro. Este libro es para aquellos que tienen curiosidad por ponerse manos a la obra con las tecnologías “vestibles”. Manten tus ojos abiertos, haz que ocurra.

David Cuartielles

David Cuartielles es el co-fundador de Arduino.cc, y es actualmente aspirante a Doctor en Diseño de Interacción en la Malmö University.

Prefacio

El contenido de este libro está inspirado en las enseñanzas del laboratorio de prototipado físico ("Physical Prototyping") de la escuela de arte y comunicación de la universidad de Malmö. El laboratorio de "Physical Prototyping" tiene algunos de los cursos más veteranos basados en la plataforma Arduino, donde Arduino ha sido un parte activa del temario de los cursos "Fashion, Body and Technology" y "Light Installation" así como en los cursos de interacción, tanto en las licenciaturas como en los masters desde 2005.

Hasta 2008 los estudiantes del curso "Fashion, Body and Technology" eran iniciados en el prototipado físico de una forma "dura" y pasada de moda donde antes se enfocaba en una mera transferencia de tecnología en el contexto de la moda y la computación "vestible". Sin embargo, en el verano de 2008 se dieron los pasos para implementar la tecnología en la moda de una forma más "suave".

Antes de esto, la experiencia de los profersores del laboratorio de prototipado físico era que los estudiantes con interés en moda y computación "vestible" tenían un duro tiempo para transladar los conocimientos del prototipado físico estándar al desarrollo de prototipos para "vestibles". Cuando buscábamos materiales adecuados en los que basar nuestro nuevo enfoque, pronto nos dimos cuenta que la información disponible era muy limitada y la mayoría de los materiales eran de tipo "arte y manualidades".

Somos firmes creyentes en el movimiento DIY o "Do-it-yourself" (en español, hazlo tu mismo) y seguimos pensando que hay mucho que aprender sobre los materiales de "arte y manualidades", pero un curso de nivel universitario debe ser capaz de ofrecer un enfoque más complejo. El problema que encontramos es que en ese momento solamente en una pequeña parte del material disponible se mostraba como comunicarlo con micro controladores.

Desde mi experiencia anterior asistiendo y enseñando cursos de prototipado normal en la escuela de Arte y Comunicación en la Universidad de Malmö, la plataforma Arduino es una de las mejores plataformas de prototipado actualmente disponibles por dos simples razones: el precio y la comunidad.

Ambas razones están relacionadas con la filosofía de Arduino. La plataforma Arduino es única porque tanto su software como su hardware son abiertos. El objetivo de Arduino era crear una plataforma de prototipado para diseñadores que fuesen capaces de realizar sus ideas por si mismos. La apertura en los diseños de Arduino significa que cualquiera es libre de hacer modificaciones tanto en el hardware como en el software e incluso fabricar y vender su propias placas. Esto a su vez significa que el precio de Arduino está en un nivel muy asequible, ya que cualquiera está autorizado a competir en la fabricación. El precio, a su vez es también una de las principales razones de su gran número de usuarios y son estos los usuarios que forman la comunidad Arduino. Esta es una comunidad que comparte un amor común por la creación de prototipos y comparte la filosofía de apertura de Arduino, lo que significa que se puede encontrar una gran cantidad de información y de ayuda. También significa que la comunidad Arduino es uno de los más actuales y rápidos actores para impulsar la evolución de los prototipos hacia adelante.

Con esto en mente la pregunta nunca ha sido alejarse de Arduino al acercarse al campo de la moda y la tecnología, sino más bien cómo abordar el campo de una forma mas “suave” usando Arduino. La mayor parte de los desarrollos en el curso están centrados en comparar proyectos de gente activa en el campo de la moda y la computación “vestible” con las enseñanzas de la Universidad para encontrar formas más “suaves” de ayudar a entender la tecnología en términos que se pueden considerar más naturales para gente que aborda el prototipado físico con un pasado mas textil.

Este libro no está únicamente orientado a gente con un

interés academico en el campo de la moda y la tecnología y la computación vestible, pero debe ser también considerado como una guía introductoría para cualquier persona con un interés general en el tema. Esta es una colección de partes y piezas que pretenden inspirar al lector y que empiece a desarrollar. Así que sinceramente espero que disfrutes del resto del libro y que recuerdes que si nosotros podemos hacerlo, tu también.

Tony Olsson

Añadido a la primera edición

Añadir que la primera versión de este libro fue escrita con bajas expectativas de interés fuera de la Universidad de Malmö. Sin embargo, una vez que la primera versión se terminó pensamos que seria una pena no compartirla.

En ese momento, la mejor idea que tuvimos fue hacer una página web sencilla y poner el libro para descargar. En dos meses tuvimos ocho mil descargas y después de unos meses el número fue lentamente creciendo hasta diez mil descargas. Para nosotros esto fue una increíble inspiración y fue el único factor que nos hizo decidir publicar una versión final del libro.

Durante el pasado año tuvimos también algunas incorporaciones al grupo de trabajo. Melissa Coleman, Derek Coleman, y Andreas Jiras han sido de gran ayuda y apoyo para el libro por lo que les damos nuestro más profundo agradecimiento.

Primera parte:

Conceptos básicos

Capitulo 1: Introducción

Prototipando con Arduino

La plataforma de prototipado Arduino está basada en el simple principio de utilizar entradas y salidas. Las entradas generalmente son algún tipo de botón, pulsador o sensor. Un botón solo tiene dos estados accionado o no accionado pero con un sensor se pueden medir un amplio rango de magnitudes del entorno. Sonido, movimiento, temperatura y luz pueden ser procesados por Arduino y si puedes pensar en alguna otra variable física que desees medir la cuestión es encontrar un sensor para ello.

De la misma forma que se pueden conectar una gran cantidad de entradas se pueden conectar una gran cantidad de salidas. Un salida puede ser desde algo tan sencillo como una luz, movimiento o calor a tipos de salidas más complejas como el envío de un SMS o apagar un aparato de TV en la otra punta del mundo. En muchos casos ya existe una solución técnica para su prototipo, solo hay que encontrarla.

Como Arduino es esencialmente un microcontrolador todo lo que se le conecte debe ser eléctrico. Lo bueno es que prácticamente todo puede ser convertido en una señal eléctrica. Por ejemplo, cuando un humano es tocado por algo se envía una señal al cerebro para que este perciba la sensación que se produce en cualquier parte de su cuerpo. De la misma manera podemos hacer una conexión eléctrica desde Arduino y hacer que vuelva a el. Si algo interrumpe esa conexión eléctrica una señal se envía de vuelta al cerebro del Arduino para decirle que no hay electricidad en la conexión.

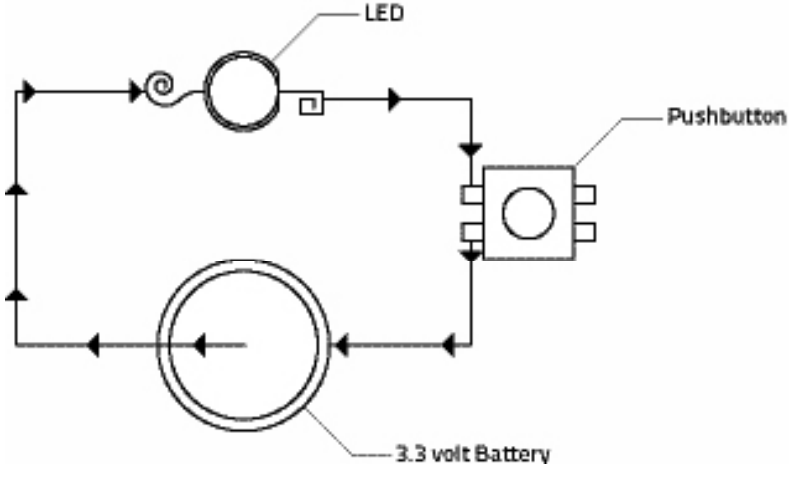
Heckear: ahorrar dinero, aprender más

Una vez que has comenzado a jugar un poco con la fabricación de prototipos electrónicos propios pronto te darás cuenta que tener una gran cantidad de componentes electrónicos cuesta una buena cantidad de dinero. Por ello la mayoría de la gente interesada en el prototipado

electrónico tiene un pie puesto en el hackeo de hardware. El hackeo de hardware, también conocido como cacharreo, describe la actividad de tomar productos electrónicos comerciales para “ver su interior”. El cacharreo no es una buena forma de aprender más sobre como funcionan los aparatos electrónicos, pero es una forma eficiente de ahorrar dinero. Un montón de componentes que puedes necesitar para tus proyectos los puedes encontrar en lo que normalmente se considera chatarra. Una impresora vieja tiene un motor que todavía puede ser funcional, un viejo teléfono tiene baterías y pequeños motores vibradores y los juguetes baratos electrónicos son, prácticamente, minas de oro. No hay una forma buena o mala de hacer hackeo o cacharreo, lo que significa que tampoco hay unas instrucciones oficiales sobre como hacerlo. Pero internet está llena de información y trucos sobre cacharreo, además la página web de Arduino, www.arduino.cc/playground, www.makezine.com y www.instructables.com son buenas formas de empezar.

Como funciona la electricidad

Hay algunas cosas que debes saber sobre la electricidad para hacer tus propios prototipos electrónicos. Lo primero es que la electricidad siempre necesita volver a su punto de origen para



hacer un circuito.

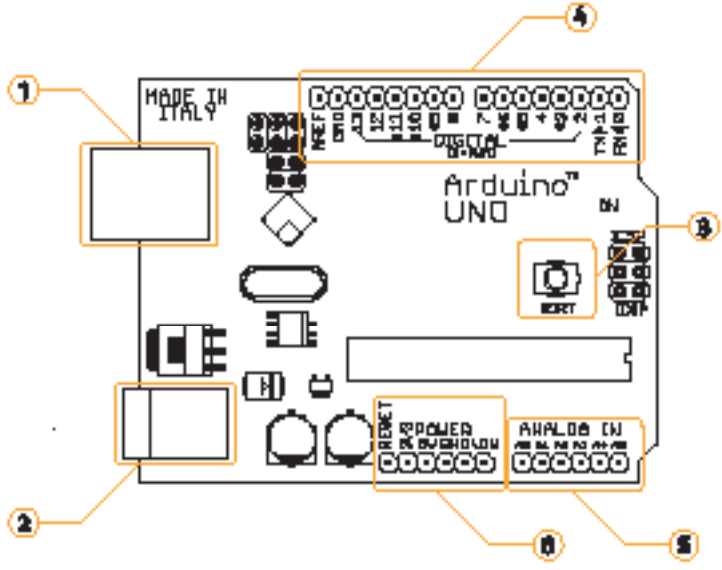
En el siguiente ejemplo tenemos conectado un LED a una batería con un pulsador: la energía de la batería viajará a través del cable hasta una patilla del LED y saldrá por la otra hacia el pulsador y volverá a la batería.

Un pulsador normalmente conecta una pequeña pieza de metal con otra cuando es accionado. Si el pulsador no es accionado no pasa nada, pero cuando se acciona se tocan las piezas metálicas y la energía de la batería puede volver allí de donde salió. Cuando regresa a la batería el LED se enciende. En el ejemplo anterior la batería es de 3.3v y el LED puede manejar 3.3v. Si conectamos una batería de 9v en su lugar el LED se quemará. Esto se debe a que la electricidad tiene diferentes voltajes y amperajes y estos viajan con resistencia. Imagina la electricidad como si fuese agua. La velocidad del agua puede ser lo mismo que el voltaje y la cantidad de agua que se mueve puede ser lo mismo que los amperios. Vamos a decir que nuestra agua está pasando por una manguera de jardín, que es la analogía más utilizada para entender la resistencia eléctrica. Entonces, conectando el LED a la batería de 9v viene a ser como bombear mucha agua a mucha presión por la manguera de jardín. Si la manguera de jardín no resiste el paso de toda el agua estallará. Es muy raro que las cosas exploten cuando hacemos prototipos con Arduino ya que la mayoría de prototipos utilizan corrientes tan pequeñas que no pueden ser siquiera consideradas peligrosas. Pero aún así nunca es una buena idea conectar más potencia de la que puede manejar a algo ya que hay muchas probabilidades de que se rompa. Por ello hay que seguir siempre las recomendaciones sobre limitación de potencia de nuestros componentes. Se pueden encontrar estas en el “datasheet” del componente. Un datasheet se puede encontrar online buscando el número del componente combinado con la palabra ‘datasheet’.

Capítulo 2: Hardware

Arduino

Arduino es una placa microcontroladora de diseño abierto utilizada para prototipado electrónico. Arduino puede recibir datos de sensores para recoger información de su entorno y se puede utilizar para controlar otros dispositivos electrónicos como luces, motores y mucho más. Hay diferentes versiones de Arduino disponibles, la más común es la última versión de la placa Arduino estándar, la cual es como esta:



Los componentes más importantes de la placa Arduino están subrralladas en naranja:
1: Conector USB.
2: Conector de corriente.
3: Pulsador de reset.
4: Pines digitales.
5: Pines analógicos.
6: Pines de corriente.

El conector USB (1) se utiliza para conectar Arduino al ordenador. La placa Arduino se alimenta a través del cable USB y mientras está conectada se puede subir código y establecer comunicaciones desde y hacia la placa Arduino.

El conector (2) se utiliza cuando no quieres alimentar tu Arduino a través del cable USB. En su lugar se puede utilizar un transformador normal (adaptador de corriente) en el rango de 6V

Nota: Aunque la placa Arduino tiene un regulador de voltaje, asegurate de no conectarlo a mas de 24 voltios, sino la placa se quemará.

a 20V. Sin embargo se recomienda el uso de uno que proporcione un voltaje de entre 7V y 12V, pues el uso de voltajes más altos puede causar daños en el regulador interno de la placa. Arduino también puede funcionar con baterías. En modelos antiguos como la Duemilanove es necesario cambiar manualmente un selector de alimentación. Esto se hace cambiando un puente (3) situado entre el conector USB y el conector de alimentación. Si quieres alimentar tu Arduino por USB sitúa el puente entre los dos pines más cercanos al conector USB, en caso de querer alimentarlo desde una fuente de alimentación externa sitúa el puente entre los dos pines más cercanos al conector de alimentación. En las versiones desarrolladas tras la Duemilanove, el Arduino conmuta automáticamente entre un tipo u otro de alimentación. En un Arduino hay 13 pines digitales (4) y estos pueden ser utilizados indistintamente como entrada o como salidas dependiendo de como las establezcamos mediante la programación: Los pines analógicos (5) funcionan únicamente como entradas pero pueden leer un rango de información más amplio que los pines digitales: A la izquierda de los pines digitales se encuentran los pines de alimentación (6). Desde aquí se pueden extraer 3.3V o 5V. El pin marcado como Vin da el mismo voltaje que entra por el conector de alimentación, esto es, si alimentas con 12V a través del conector de alimentación obtendrás el mismo voltaje en ese pin. Aquí se pueden encontrar dos pines GND.

Nota: GND representa masa.

El pulsador de reset (7) se utiliza para inicializar cualquier programa en Arduino y que vuelva a empezar. En Arduinos más antiguos que el Diecimila el botón de reset debía ser pulsado cada vez que se quería subir código.

Para los ejemplos de este libro se ha utilizado una Arduino estándar, ya que en nuestra opinión es la mejor placa para prototipar. Cuando estés cerca de finalizar tus prototipos puede ser útil migrar a alguno de los modelos más pequeños para ahorrar espacio. Estas otras placas trabajan de la misma forma y cualquier programa escrito para una Arduino estándar funcionará en cualquier otro modelo de Arduino.

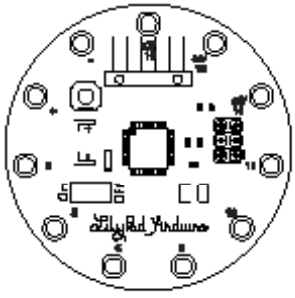
Los siguientes son dos ejemplos de placas Arduino disponibles.

Nota:
ATmega 168 es un microcontrolador producido por Atmel. Tiene las instrucciones básicas de Atmel AVR. Para más información sobre Atmel y la familia de componentes ATmega, visita la página web: www.atmel.com.

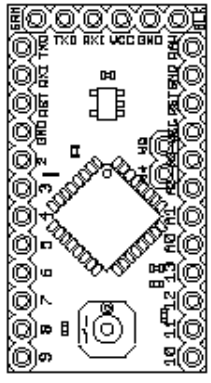
LilyPad

La Arduino LilyPad es un microcontrolador diseñado para "vestibles" y e-textiles. Puede ser cosida sobre tela y, utilizando hilo conductor, alimentarla desde una fuente de alimentación, conectarle sensores y actuadores. Esta placa esta basada en el chip ATmega328V (la versión de bajo consumo del ATmega328). La Arduino LilyPad fue diseñada y desarrollada por Leah Buechley y SparkFun Electronics. El + de la Lilypad acepta voltajes de 2.7V a 5.5V.

Nota: Para aprender más sobre el hardware de la placa estandard Arduino u otras placas Arduino visita la página oficial Arduino: www.arduino.cc.



Alimentarla con más de 5.5V puede destruir la Lilypad. Siempre hay que tener cuidado a la hora de conectar baterías, las cuales pueden dar voltajes iniciales superiores a los especificados en sus etiquetas. Es un hábito muy recomendable medir el voltaje de las baterías antes de conectarlas



Arduino Mini

La Arduino Mini es un pequeño micro controlador basado en el ATmega168, utilizado para su montaje en placas de prototipado y cuando el espacio está limitado. Tiene 14 entradas/salidas digitales (6 de las cuales puedes ser utilizadas como salidas PWM), 8 entradas a analógicas y un cristal oscilador de 16 MHz. Puede ser programado mediante un adaptador mini USB o con un adaptador de USB o RS232 a TTL serie (www.arduino.cc). Si se eliminan los pines macho de esta placa puede ser cosida

igualmente a un trozo de tela mediante hilo conductor.

Componentes electrónicos básicos para prototipado textil

• Hilo de coser conductor

Este tipo de hilo parece hilo de coser normal de color gris pero tiene la particularidad de poder transportar corriente eléctrica para alimentación o para señales. El hilo de coser conductor más común es de metal plateado, esto significa que una capa de metal se ha puesto sobre el hilo textil y esta capa es conductora. El hilo de coser conductor puede utilizarse igual que los cables normales y es muy apropiado para el prototipado “blando”.

Conductive threads come in different thicknesses and with
Los hilos de coser conductores vienen en diferentes grosores y con diferentes resistencias. La resistencia en el hilo baja la presión del voltaje en la fuente de alimentación que estés utilizando. Esto significa que menos corriente puede pasar a través del hilo. La resistencia del hilo puede ser calculada por metro o medida con un multímetro. Recuerda el simple principio que a mayor longitud mayor resistencia. Con hilos de coser conductores y tela el tamaño de la superficie también influye en la resistencia. Una gran superficie conductora tendrá una menor resistencia, esto significa que para la misma longitud un hilo de coser más grueso tendrá menor resistencia que uno más fino. Si la resistencia del hilo de coser conductor se hace demasiado alta para tu circuito y tu diseño requiere un gran recorrido puede ser una buena idea utilizar cinta de tela conductora o combinar múltiples hilos conductores en un solo hilo de coser más grueso.

Cuando uses hilo conductor con una máquina de coser siempre es más sencillo utilizarlo como hilo inferior - este es el llamado hilo de la canilla. Alguno hilos de coser conductores se pueden utilizar como hilo superior, pero asegúrate de ajustar correctamente la tensión. El uso de hilo de coser conductor no apropiado para máquina de coser/bordar también puede funcionar, pero el hilo

puede romperse y como resultado dar algunas imperfecciones y más trabajo para terminar que si el hilo de coser utilizado hubiese sido para máquina.

Algunos hilos de coser constan de un hilo metálico unido a un hilo textil resistente al calor o son puramente metálicos. Estos cables se pueden soldar directamente a una superficie metálica como un botón metálico o una cremallera. Este tipo conexiones son generalmente más rápidas y más fiables que las conexiones cosidas.

• **Resistencias**
Una resistencia es un componente electrónico diseñado para oponerse al paso de una corriente eléctrica produciendo una caída de voltaje entre sus terminales en proporción a la corriente. La resistencia siempre se mide en Ohmios y también puede ser representada por el símbolo Ω. Los múltiplos mas comunes para los cálculos de resistencias son: el Kilo Ohm (kΩ), el cual equivale a 1000 Ω.

Nota: El tercer dígito no se usa en las resistencias de 4 bandas de colores. En las resistencias con 4 bandas de colores, el tercero es el multiplicador y el cuarto es la tolerancia.

Mega Ohm (MΩ), lo que es igual a 1,000,000 Ω.

Las resistencias están codificadas con colores, pueden tener 4, 5 o 6 bandas de color y son estas bandas de color las que te indicarán su valor.

En una resistencia de 6 bandas de color las 3 primeras son dígitos que siguen el siguiente esquema de color:

negro 0 marrón1 rojo 2 naranja 3 amarillo 4

verde 5 azul 6 violeta 7 gris 8 blanco 9

Si las tres bandas fueran marrón, rojo y azul deberían interpretarse como 126. La cuarta banda es el multiplicador, multiplicas los tres primeros dígitos por esta banda para obtener

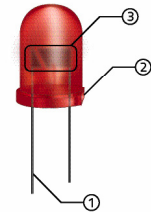
el valor total de tu resistencia. La cuarta banda sigue el siguiente esquema de color:

plata 0.01 oro 0.1 negro 1marrón10rojo 100 naranja 1k

amarillo 10k verde 100k azul 1M violeta 10M

La quinta banda es la tolerancia de la resistencia y la sexta banda se utiliza para su coeficiente de temperatura. Es difícil aprender a calcular resistencias mentalmente, por ello recomendamos el uso de calculadoras online para estar seguros, si buscas en google "calcular resistencia colores" encontrarás montones de ellos. Otra forma – más rápida – de leer el valor de una resistencia es leerla directamente con un multímetro.

• **LED**



Un diodo emisor de luz (LED por sus siglas en ingles) es un semiconductor que se enciende cuando una corriente eléctrica lo atraviesa en el sentido adecuado. Hay tres formas de encontrar el sentido adecuado de un LED, la primera es mirando sus patillas, una es más larga. Esta patilla más larga es la que se debe conectar a la fuente de alimentación y la patilla corta debe conectarse a un pin GND (1). La segunda forma es de averiguar el sentido de un LED es mirar el borde del encapsulado plástico (2), en el borde inferior del encapsulado debe haber una porción plana o dentado.

La patilla cercana al borde plano va a GND y la otra patilla va a power. La tercera forma es orientar el LED hacia una luz y mirar en su interior, hay dos terminales, uno pequeño y otro grande (3). La patilla que va al terminal pequeño debe ser conectada a power y la patilla del terminal grande va a GND.

Los LEDs más comunes se alimentan en torno a los 3.3V y los 20mA. Nótese que Arduino proporciona siempre 5V en los pines digitales, por lo que se hace obligatorio el uso de una resistencia de 220 Ω para reducir el voltaje y no quemar el LED. Para estar

seguro de los requerimientos de sus LEDs vea los correspondientes datasheets, esta información siempre está disponible para cualquier componente electrónico que compres. Si necesitas calcular la resistencia necesaria para tus LEDs te recomiendo que utilices un calculador online de resistencias.

(www.dannyg.com/examples/res2/resistor.htm).

• **Telas conductoras**
Las telas conductoras tienen la cualidad de conducir la electricidad. Normalmente son una combinación de metales altamente conductores y telas ligeras. Normalmente se utilizan como material de protección.

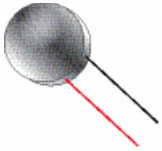
• **Sensor de inclinación**
Un sensor de inclinación puede detectar si un objeto se esta inclinando hacia un lado o hacia otro. El tipo más barato de sensor de inclinación también se puede utilizar para detectar de movimiento. El inconveniente de este tipo de sensor de inclinación es que funcionan como un pulsador, por lo que no puede decir en que dirección el objeto se está inclinando o cuanto, solo que se está inclinando. Dentro del sensor de inclinación hay una pequeña bola metálica en el interior de una carcasa, cuando la bola toca los laterales de la carcasa metálica cierra el circuito y podemos leer el movimiento de inclinación desde la placa Arduino.

• **Sensor LDR**
La LDR (por las siglas en ingles de Light Dependent Resistor) también llamada foto-resistencia. Una LDR esta hecha de un semiconductor de muy alta resistencia. Una LDR es similar a una resistencia normal con la excepción de que una resistencia normal tiene un valor fijo y la resistencia de una LDR depende de la luz que haya en sus alrededores. Es difícil determinar una cantidad exacta de luz en los valores proporcionados por una LDR, pero son buenas para detectar luz en un sentido más amplio - si está

oscuro o si hay luz.

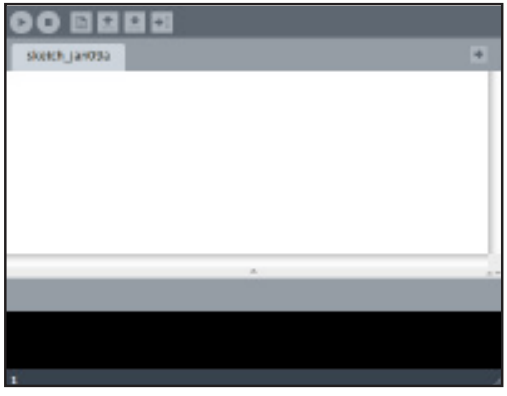
• **NTC sensor**
El sensor NTC (por las siglas de su denominación en inglés Negative Temperature Coefficient) también es conocido como termistor. Un termistor es un tipo de resistencia cuyo valor cambia de acuerdo con la temperatura. Es difícil medir temperaturas con precisión con un termistor, pero permiten determinar si algo está caliente o frío.

• **Motores**
Si quieres que algo se mueva probablemente necesitas algún tipo de motor. Un motor es más o menos un actuador que transforma la electricidad en movimiento. Cuando trabajas con prototipos "vestibles" ocultos los motores pueden ser un problema si necesitas mucha fuerza. La mayoría de los motores siguen un principio muy simple "grandes fuerzas, grandes motores", no obstante hay montones de pequeños motores que nos será útiles para nuestro tipo de aplicaciones y con un poco de creatividad se pueden integrar bien en nuestros prototipos.
Hay tres tipos principales de motores: los motores DC, los servos y los motores paso a paso. No hemos incluido ejemplos de motores paso a paso en este libro. Aunque los motores paso a paso son buenos para rotación completa y por pasos no son apropiados para su implementación en tecnología "vestible" ya que son muy pesados para su tamaño y muy voluminosos de forma. Si estás considerando utilizar un motor es recomendable que busques un motor DC o un servo que se adapte a tu prototipo.

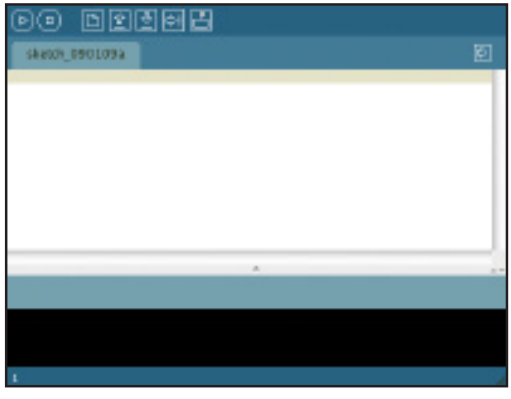


Capítulo 3: Software

El software usado para escribir programas para tu placa Arduino es llamado Arduino IDE (Integrated Development Enviroment). El Arduino IDE está basado en otro lenguaje de programación open-source llamado Processing, el cual es usado para programar imágenes, animaciones e interacciones con una computadora. El Arduino IDE parece muy similar al Processing IDE:



Arriba está el Processing IDE y debajo puedes ver el Arduino IDE.



El lenguaje de Processing forma el molde para el lenguaje de Arduino. El lenguaje de Arduino está basado en comandos fáciles de usar. Cada vez que tu presionas el botón de Upload en el Arduino IDE, éste traduce tu programa a código en C y la placa Arduino puede entender tu programa. El lenguaje está construido de esta forma por que la programación en C es difícil para quienes programan por primera vez.

Instalando el software

El software se obtiene en el sitio wwww.arduino.cc en el apartado downloads. Una vez que has encontrado la página de descarga, escoge la versión correcta para tu sistema operativo. Cuando hayas copiado el software, descomprime el paquete y coloca el directorio Arduino en tu escritorio.

Esta guía solo contempla la placa Arduino Uno. Si utilizas una versión más antigua pega un vistazo a la web de Arduino para obtener información sobre como instalarla.

Guía rápida:
1. Abre el menu, clica el botón derecho en Ordenador y selecciona Propiedades.
2. Selecciona Administrador de dispositivos.
3. Encuentra en la lista i clica con el botón derecho en Arduino UNO (COMxx).
4. Selecciona "Actualizar Driver Software".5. Elige " Buscar en mi ordenador el driver del software y redirecciona a /Arduino/ Drivers".

• **Para usuarios de XP**
Después de efectuar los pasos de arriba, toma tu cable USB y conéctalo a un puerto USB en tu computadora. Windows tratará de instalar el controlador pero fallará. No te preocupes, esto es normal.

Una vez que el proceso de instalación falló, abre el Menú Inicio y haz clic derecho en Mi PC y selecciona Propiedades. Esto abrirá la ventana de Propiedades del Sistema, selecciona la pestaña Hardware y en la parte superior derecha encontrarás la opción Administrador de Dispositivos. Haz clic en Administrador de Dispositivos y abrirá otra ventana con una lista de dispositivos en tu computadora. Navega a Puertos (COM y LPT), y debes encontrar un dispositivo llamado 'Arduino UNO (COMxx)'. Haz clic derecho en él y selecciona la opción 'Actualizar Controlador'. Esto abrirá una ventana de instalación donde tu deberás elegir la

opción Instalar desde una lista o ubicación específica'. Éste paso final consiste en dirigir el instalador hacia el directorio que está adentro del directorio Arduino en tu computadora. Asegúrate que el directorio seleccionado se llame Driver y no FTDI USB Driver que es otro directorio dentro de Driver. Presiona siguiente y windows finalizará la instalación. Algunas veces Windows muestra un mensaje de advertencia el cual puedes ignorar y elegir la opción de instalar el controlador.

• **Para usuarios de OSX**
Monta la imagen de disco (arduino-00xx.dmg) y una ventana aparecerá. En esta ventana verás dos iconos, uno que dice Arduino y una carpeta que dice Aplicaciones. Solo arrastra el icono de Arduino sobre el icono de Aplicaciones dentro de la ventana y haz terminado. No se necesitan controladores para las placas Uno. Después de que tu instalación está completa, deberías encontrar el software Arduino dentro de tu directorio de aplicaciones.

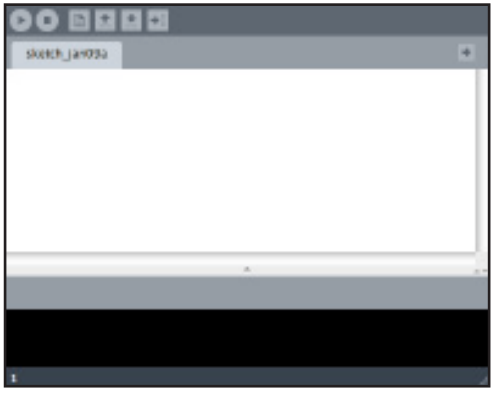
Guía rápida: 1. Monta la imagen del disco.2. Arrastra el icono Arduino a la carpeta Aplicaciones .3. Ya lo tienes.

• **Para usuarios de Linux**
El proceso de instalación para Arduino en Linux puede ser diferente de una distribución a otra. Por favor consulta la página web de Arduino y/o página web de tu distribución para información actualizada sobre la instalación para tu distribución.

Capítulo 4: Utilizando el IDE

The IDE buttons:

El IDE consiste en dos grandes espacios, uno blanco y otro negro. El espacio blanco es donde escribirás tu programa. Ten en cuenta que cualquier cosa que escribas ahí se considerará código si no la "comentas". Para aprender más sobre como esconder texto en tu código, vete a la página 80.



El espacio negro es donde recibirás los errores y mensajes de confirmación. Encima de el espacio blanco encontrarás los botones del IDE. Con estos botones controlas la mayoría de las acciones en el IDE.

Nota: el IDE hace un chequeo lógico y no puede determinar si el programa corresponde con lo que quieres que haga el programa. Una vez la compilación ha comenzado, no la puedes parar con el botón Stop.

El primero de ellos es el botón Compilar. Este botón comprueba tu programa para ver si hay cualquier error lógico en tu código. El segundo botón es el botón Parar. Este botón se utiliza para apagar el monitor del puerto serie. El compilador se tomará el tiempo que necesite, pero no te preocupes, compilar normalmente solo lleva unos cuantos segundos dependiendo como de grande sea tu programa.

El tercer botón es el botón Nuevo Sketch. En el IDE Arduino todos

los programas que tu abres o escribes se llaman sketches. El botón Nuevo Sketch abrirá un nuevo sketch, pero te preguntará primero si quieres guardar el sketch actual. El cuarto botón es el botón Abrir Sketch. Este botón abre la carpeta de sketches y ahí puedes elegir que sketch abrir de los que ya estaban guardados.

El quinto botón es el botón Guardar Sketch. Este botón guarda el actual sketch en una carpeta llamada sketchbook. El sexto botón es el botón Subir. Este botón subirá el programa actual a tu placa Arduino asumiendo que no hay ningún error en tu código. El botón Subir primero intentará compilar tu código y si encuentra cualquier error, parará de compilar y aparecerá un mensaje de error en la ventana negra del IDE, informándote cual es el problema. A continuación el IDE resaltará la línea de código que está causando el problema.

El ultimo botón es el botón Monitor de puerto Serie. Este botón abrirá el monitor de puerto serie en una nueva ventana, en la cual encontrarás una lista desplegable, un botón de enviar y un campo de texto. Para parar el monitor, utiliza el botón Parar. En la parte superior del IDE encontraras menús desplegables como en cualquier otro programa.

En el menú Archivo encontrarás todas las funciones de los botones, tu carpeta sketchbook y las preferencias. En el menú Edición encontraras las funciones y comandos para deshacer, rehacer, cortar, copiar, pegar, seleccionar todo, buscar, y buscar siguiente. En el menú Sketch puedes verificar/compilar tu código, parar, importar librerías, mostrar la carpeta sketchbook y añadir un nuevo archivo. Las dos funciones mas importantes del menú Herramientas son el modelo de placa y el puerto serie. La opción de modelo de placa es donde seleccionas el tipo de Arduino que vas a conectar. En puerto serie es donde seleccionas el puerto USB donde tienes conectado tu Arduino. La manera mas fácil de determinar en que puerto está tu placa, ya que puede aparecer mas de un puerto en el menú, es desenchufar tu placa y fijarte que puertos están conectados. Una vez hecho esto, conecta de

nuevo tu placa y el puerto nuevo que aparecerá en la lista es tu placa Arduino.

Subiendo código

Para comprobar si tu software está bien instalado abre el código de ejemplo blink que encontraras en:

Archivo/sketchbook/examples/digital/blink.

Una vez tengas el código, asegúrate de que tienes seleccionado correctamente el tipo de placa y el puerto serie en el menú Herramientas. Haz click en el botón Subir y si todo va sin ningún problema, el LED de la placa cercano al pin 13 deberá empezar a parpadear encendiéndose y apagándose con un retardo de un segundo..

Segunda Parte:

Ejemplos

Examples

Here we will present a collection of examples on how to create and use both input and output devices and how to interface them with an Arduino board. These examples are not in any way finished prototypes but should be considered as inspirational construction solutions and programming techniques that can be useful for fashionable and wearable prototypes.

This section will start off with simpler examples, gradually turning to more complex construction and techniques. Each example starts with a list of all components needed and uses components and materials that should be supplied by most electronic hobbyist stores. Some materials like conductive fabrics and thread can be tricky to find. Use the internet to find suppliers near you or compare online stores to find materials at the best prices.

Capítulo 5: Usando Pines digitales

Como se explicó en el capítulo 2, los pines digitales en el Arduino solo tienen dos estados, o están en On o en Off. El comando actual para estos dos estados en el Arduino es 1 para on y 0 para off, y esto es por lo que los llamamos pines digitales. Normalmente usamos las constantes HIGH y LOW ya que hacen más fácil leer el código comparado con usar 0 y 1. Recuerda que los pines digitales siempre dan una potencia de salida de 5V cuando están en HIGH y 0V cuando están en LOW. No conectes nada directamente a los pines digitales a menos que estés seguro que puede manejar 5V.

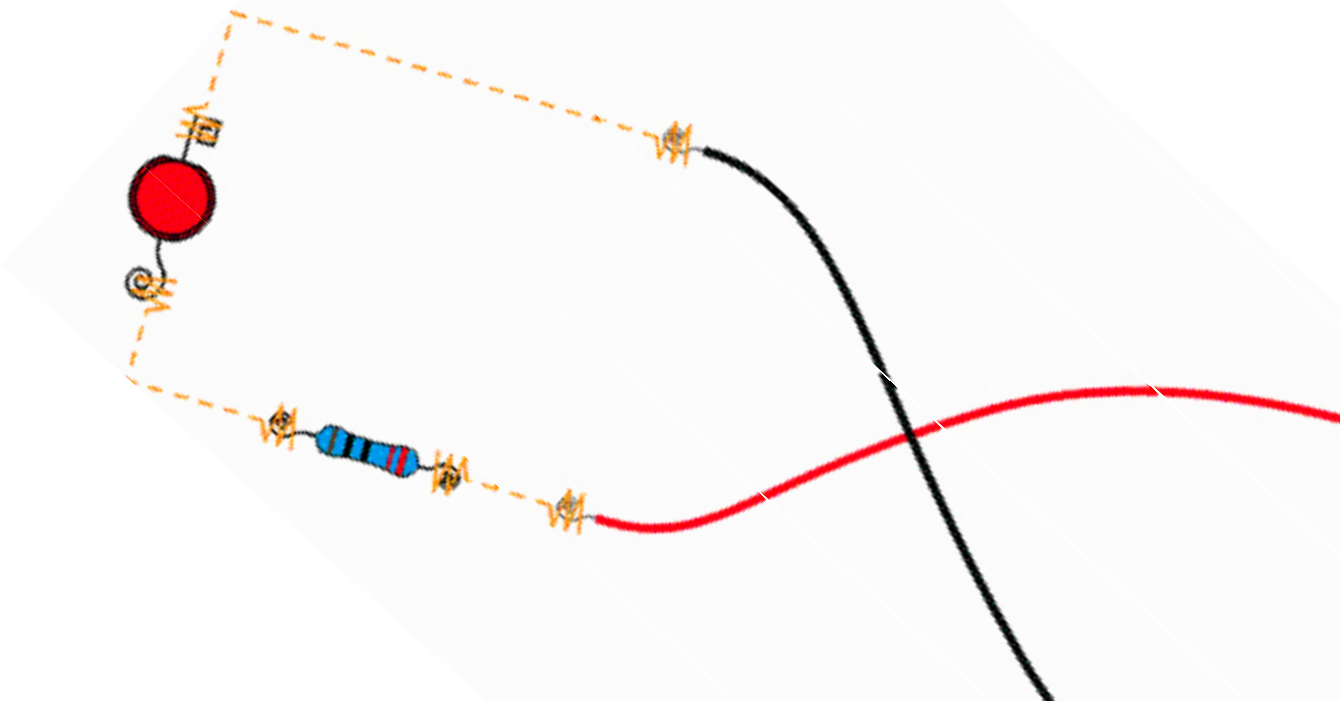
1: Prototipado textil con LEDs

Para este ejemplo necesitarás:

- Algo de hilo conductor.
- Un LED.
- Una resistencia de 220 Ω.
- Tela no conductora.

• **Cosiendo un LED**

Este dibujo ilustra como coser tu circuito a una pieza de tela. Tu puedes usar cualquier pieza de tela siempre que no sea conductiva:



En este ejemplo hemos hecho unos círculos en las patas de una resistencia y círculos y cuadrados en las patas del LED. Esto no es solo por razones cosméticas, es también práctico – hace más fácil coser los componentes en su sitio y las diferentes formas son también buenas para mantener la pista de las patas largas y cortas del LED. Para la conexión del voltaje y la tierra hemos usado cables, cosidos en su lugar con hilo conductor. Esta es una buena técnica para usar mientras pruebas tu prototipo. Para prototipos finales es recomendado el coser tu placa arduino en la tela. Dependiendo de tu diseño el camino del hilo conductor desde tu Arduino hasta tu LED puede ser tan largo que no necesites la resistencia de 220 Ω en tu circuito. Mide la resistencia de tu camino de hilo conductor y mira si es suficientemente largo para reemplazar a la resistencia.

• Programando un LED parpadeante

Antes de conectar tu Arduino tu puedes probar si tu código funciona y que no hay no daños a tu placa Arduino. El Arduino tiene una pequeño LED incorporado junto al pin digital 13 al que está conectado, así, el código siguiente debería hacerlo parpadear:

```
int ledPin = 13;
 /* una variable entero para el LED conectado al pin
digital 13 */

void setup(){
 pinMode(ledPin,OUTPUT);
 /* configura el ledPin como salida  */
}

void loop(){
 digitalWrite(ledPin,HIGH);
 /* enciende el ledPin  */
 delay(1000);
 /* espera un segundo */
 digitalWrite(ledPin,LOW);
 /* apaga el ledPin */
 delay(1000);
 /* espera un segundo */

}
```

Este programa encenderá el LED durante un segundo, entonces lo apagará durante otro segundo y volverá a comenzar. Una vez has escrito el programa en el IDE de Arduino, pulsa el botón de verificar y el mensaje “Done compiling” debería aparecer en la ventana negra.

Si no hay errores de compilación comprueba que tienes seleccionado el Puerto Serie y la placa correctos en el menu tools.

El paso siguiente es subir el código a tu placa Arduino pulsando el botón upload. Hay dos LEDs en el arduino: RX y TX. Cada vez que tu intentes subir un programa al Arduino estos dos LEDs deberían empezar a parpadear. Si no lo hacen, mira en el menu tools y comprueba que están seleccionados el Puerto serie y la placa correctos.

Si no hay errores en la subida, el programa está ahora almacenado en la memoria del Arduino y ahí se quedará hasta que lo reemplaces por otro nuevo y después de 5 segundos el programa empezará. Si el LED al lado del pin 13 de la placa empieza a parpadear es seguro asumir que el programa fue correctamente subido y que no hay nada mal en la placa Arduino. Este programa de LED parpadeante es bueno para usarlo si experimentas problemas mientras estás haciendo prototips y quieres excluir un fallo del hardware de la lista de posibles problemas. Ahora es seguro conectar el arduino a tu circuito textil. En el siguiente dibujo hemos usado rojo para el cable que va del pin 13 y negro para el cable que vuelve hacia el puerto GND en el Arduino.

Así es como comunmente codificarás con colores los cables mientras trabajas con electrónica. El rojo es usado para marcar un cable que conecta a la colrriente y el negro es usado para marcar los cables como tierra. Todos los siguientes ejemplos usados en este libro seguirán el mismo esquema de colores. Notar que

los cables rojos no siempre conectan a pines digitales. En este ejemplo estamos cambiando 5V on y off en los pines digitales y por eso hemos usado el rojo para marcar el cable.

• **Programando un LED desvaneciente**

En el ejemplo previo hemos cambiado un LED entre on y off. Como habrás notado, el led estuvo puesto al máximo en on (reducido a 5V con una resistencia) y luego a completamente off (0V). Como hemos mencionado antes, los pines digitales solo tienen dos modos, HIGH y LOW. Pero los pines digitales 3,5,6,9,10 y 11 tienen una función especial llamada PWM. Con el modo PWM podemos transformar los 5V de salida en un rango de 255 posibles niveles. Para aprender más acerca de la función PWM ir a la página 97. En el siguiente ejemplo vamos a probar como desvanecer un LED hacia arriba y abajo:

```
int ledPin = 5;
 /* conecta tu led al pin digital 5 */

void setup(){
 pinMode( ledPin , OUTPUT );
 /* declara el ledPin como OUTPUT */
}

void loop(){
 for(int i = 0; i < 255; i++){
 /* mientras que i sea menor que 255, incrementar i en
una unidad  */
 analogWrite( ledPin , i );
 /* desvanecer el led a  i */
 delay(30);
 /* pequeña pausa para que podamos ver cada paso */
}

for(int i = 255; i > 0; i--){
 /* mientras que i sea mayor que 0, decrementar i en una
unidad */
 analogWrite(ledPin,i);
 /* desvanecer el led a  i */
 delay(30);
 /* pequeña pausa para que podamos ver cada paso */
}
}
```

En el ejemplo anterior estamos usando el comando analogWrite(),

el cual es el comando para PWM en el LED. También usamos dos bucles for, el primero comienza contando desde 0 hasta 255. Para cada iteración se incrementa el voltage en el LED. En lugar de ir directamente de 0V a 5V, va de 0V a 5V en 255 pasos, lo que causa que el led se desvanezca. El segundo bucle for hace lo contrario, va de 5V a 0V en 255 pasos. El retraso de 30 milisegundos nos da tiempo a ver los pasos.

Ahora podemos conectar el led como hicimos en el ejemplo previo, subir el código a la placa arduino y disfrutar del desvanecimiento. Deberías poder ver como el LED se desvanece arriba y abajo una y otra vez.

2: Botón pulsador

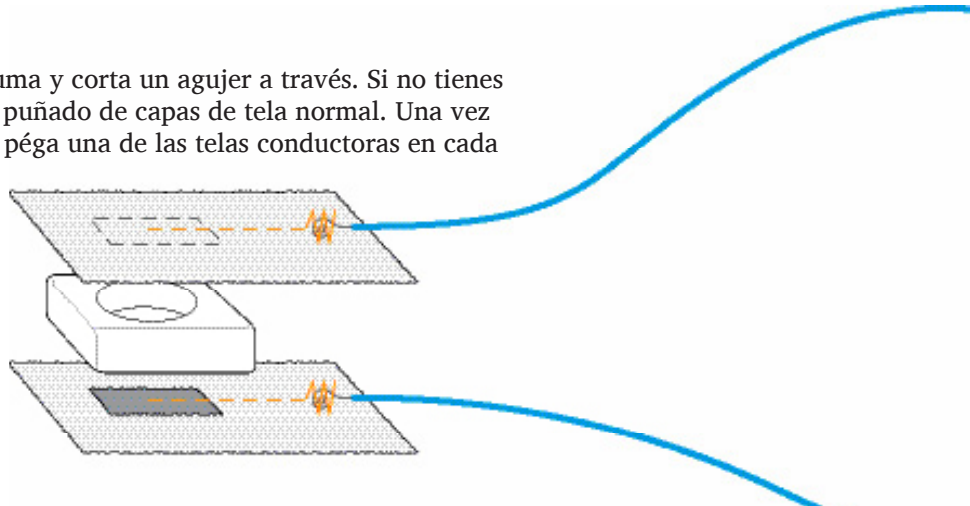
Hay diferentes formas de hacer tus propios botones pulsadores. Todas ellas están basadas en el principio de crear un circuito desde y hacia el Arduino con un punto de ruptura en alguna parte del circuito. Un punto de ruptura es donde podemos reconectar el circuito, pudiendo determinar si el botón pulsador están pulsado o no.

Para este ejemplo necesitarás:

- Tela conductora.
- Tela no conductora.
- Un trozo de espuma.
- Hilo conductor.
- Cables.
- Pegamento de tela.

Para hacer este botón pulsador, comienza cortando dos pequeños trozos de tela conductora y pégalos en dos piezas separadas de tela no conductora. Entonces cose un cable hasta el final de ambas piezas y cose una conexión a cada una de las telas conductoras.

Toma una pieza de espuma y corta un agujer a través. Si no tienes espuma puedes usar un puñado de capas de tela normal. Una vez el agujero está cortado, péga una de las telas conductoras en cada lado de la espuma:



Cuando tengas el botón pulsador preparado podemos comenzar con el programa:

Nota:
En la configuración conectamos 5V en el botón. Cada vez que pulsamos el botón redireccionamos la energía al GND y si leemos el estado del pin asignado al botón veremos que está a LOW ya que no hay energía en el pin en ese momento.

```
int myButton = 4;
  /* declara el pin digital 4 en el Arduino como myButton
*/
int ledPin = 13;
  /* declara el pin digital 13 en el Arduino como ledPin
*/

void setup(){
  pinMode(ledPin,OUTPUT);
  /* poner ledPin como OUTPUT */
  pinMode(myButton,INPUT);
  /* poner myButton como INPUT */
  digitalWrite(myButton,HIGH);
  /* activar la resistencia interna en el pin 4 */
}

void loop(){
  if(digitalRead(myButton) == LOW ){
  /*comprueba si el botón ha sido pulsado */
  digitalWrite(ledPin,HIGH);
  /* si el botón es pulsado, iluminar el led */
  }else{
  digitalWrite(ledPin,LOW);
  /* si el botón no es pulsado, apagar el led */
}
}
```

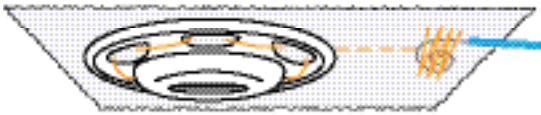
Ahora subimos el programa a nuestra placa arduino y conectamos el botón pulsador. Un final se conecta al pin digital 4 y el otro es conectado al puerto GND. El programa de arriba comprobará si el botón pulsador el pulsado y encenderá el LED al lado del pin 13. Si el botón no es pulsado el LED permanecerá apagado.

Para este ejemplo necesitarás:

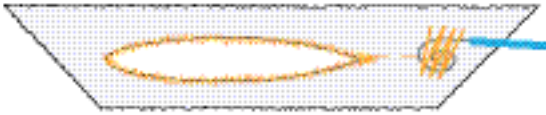
- Automático metálico para
- coser, botón metálico para coser, o botón de tejanos.
- Tela no conductora.
- Hilo conductor.
- Cables.

3: Botón pulsador oculto

Las siguientes son dos formas simples de hacer botónes pulsadores ocultos, los cuales son apropiados cuando necesitas una entrada discreta en una prenda. Puedes usar el mismo código de arriba para probarlo. El primero está hecho con dos botónes de metales que pueden ser comprados en cualquier mercería. Cose los botónes en el lugar usando hilo conductor y conecta dos cables al final.



Le último botón oculto es hecho como un botón normal . El truco es usar otra vez hilo conductor para coser el botón en el lugar y coser alrededor del agujero del botón usando el mismo hilo. La mayor parte de las máquinas de coser modernas tienen una función preprogramada para hacer agujeros de botón y funcionará perfectamente para esto. Solo carga tu máquina con hilo conductor y corta un agujero en la tela. Este ejemplo funciona igualmente con botón de coser o de jeans.



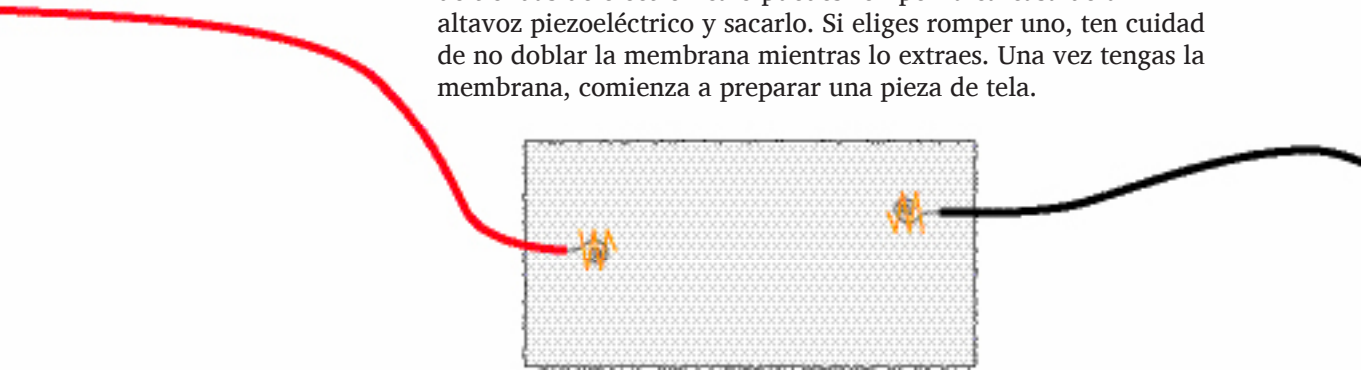
En estos ejemplos hemos puesto cables al final para hacer más fácil el testeo rápido. Para finalizar los prototipos cosemos cada botón con hilo conductor hasta nuestro Arduino.

Para este ejemplo necesitarás:

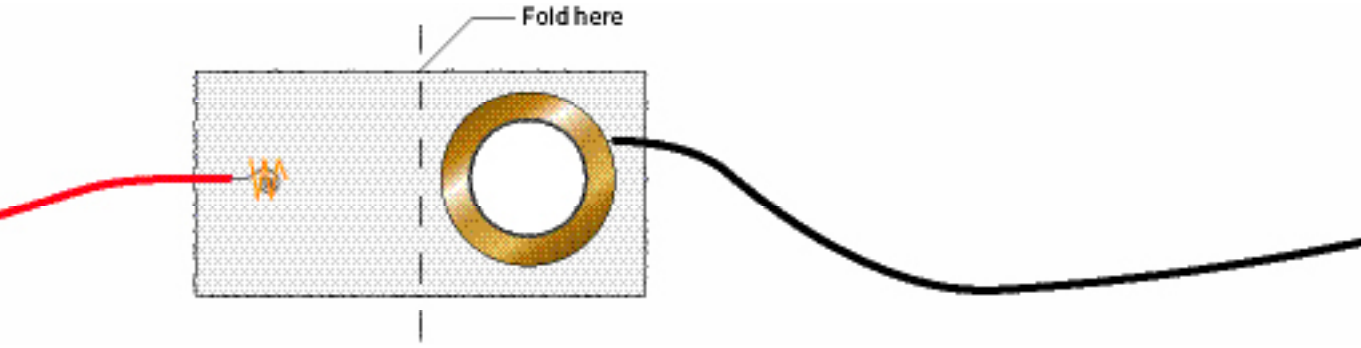
- Un altavoz piezo eléctrico.
- Tela no conductora.
- Hilo conductor.
- Dos cables.

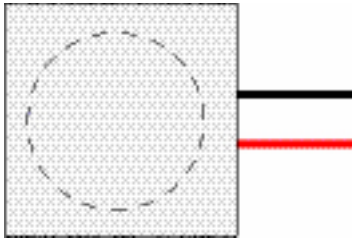
4: Sonido

La forma más sencilla de hacer sonido con el arduino es usar un altavoz piezoelectrico. La piezoelectricidad es la abilidad de los materiales de generar un campo eléctrico en respuesta a una excitación mecánica. Un altavoz piezoeléctrico consiste de dos placas metálicas, y cuando la electricidad es aplicada al altavoz piezoeléctrico, se causa que el metal se atraigan y se repelan, generando rápidas vibraciones las cuales se vuelven sonido. Los altavoces piezoeléctricos están disponibles en una amplia variedad de tamaños y formas. En el ejemplos solo vamos a usar la membrana del alatavoz piezoelectrico. Puedes comprar altavoces piezoeléctricos de membrana en la mayoría de tiendas de electrónica o puedes romper la carcasa de un altavoz piezoeléctrico y sacarlo. Si eliges romper uno, ten cuidad de no doblar la membrana mientras lo extraes. Una vez tengas la membrana, comienza a preparar una pieza de tela.



Cose dos cables de longitud 1/3 el ancho de cada lado de la tecla. Pega el final de el cable sin el plástico protector con el hilo conductor, y entonces cose la parte restante del cable con hilo normal. No uses hilo conductor cuando juntes el resto del cable con la prenda ya que puede crear un cortocircuito una vez que el altavoz piezoeléctrico esté en su lugar. Las uniones con hilo normal sirven para mantener los cables en su sitio. Coloca el elemento sobre un lado y dobla el otro lado sobre el, y une todo junto con hilo normal. Es aconsejable coser alrededor de la membrana, ya que necesita estár ajustado para conectar los cables en ambos lados de la membrana.





Ahora tenemos un altavoz piezo que es más flexible que uno normal, y podemos empezar con algunos programas para generar sonidos. Para hacer las vibraciones que generarán el sonido vamos a pulsar el piezo con tensión. Para hacer estos pulsos lo suficientemente rápido para generar vibrarciones no podemos usar el delay() normal, ya que la pausa no es suficientemente rápida.

En su lugar vamos a usar delayMicroseconds(). En el siguiente ejemplo vamos a generar una vibración que crea el tono A:

```
int piezoPin = 9;
  /* conecta el piezo al pin 9 */

void setup(){
  pinMode( piezoPin,OUTPUT);
  /* declara el pin del piezo como salida */
}

void loop(){
  digitalWrite(piezoPin,HIGH);
  delayMicroseconds(1136);
  /* el tiempo aquí determinará el tono */
  digitalWrite(piezoPin,LOW);
  delayMicroseconds(1136);
  /* el tiempo aquí determinará el tono */
}
```

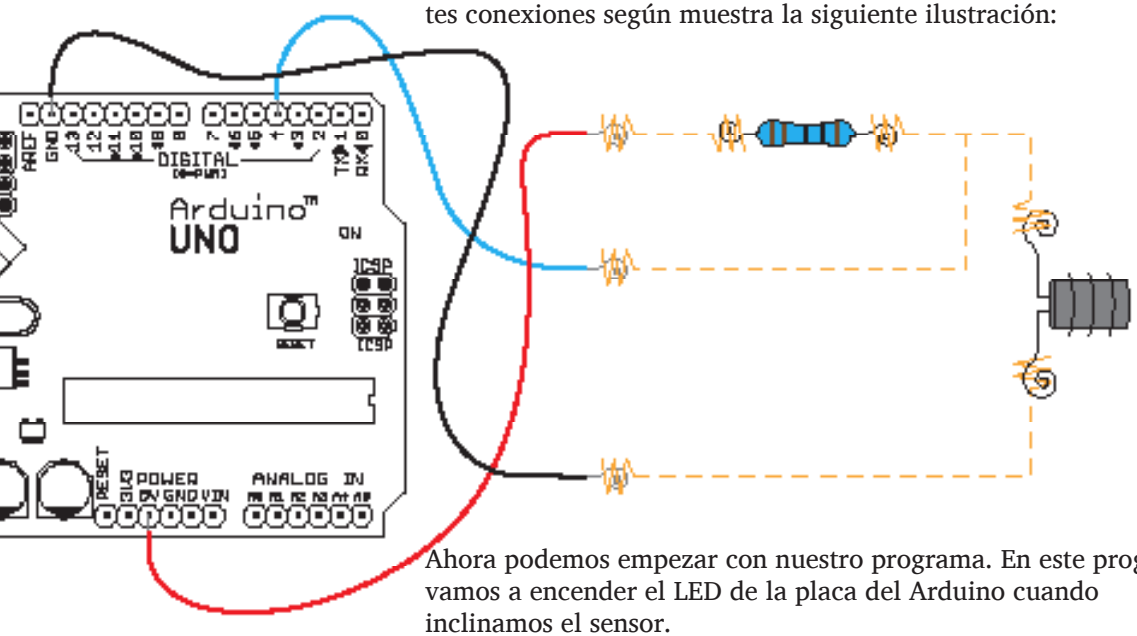
En el ejemplo anterior conectamos y desconectamos el piezo a 5V y entre medias ponemos una pausa de 1136 microsegundos. Esta pausa es lo que determina que tono es generado. Si por ejemplo

Para este ejemplo necesitarás
- Una resistencia de de 1KΩ.
- Un sensor de posición.
- Tela no conductora.
- Hilo conductor.
- Dos cables.

nosotros hacemos una pausa de 1911 microsegundos, podríamos en su lugar obtener el tono C.

5: Sensor de inclinación

Comienza conectando el sensor de inclinación y una resistencia de 1kΩ a una pieza de tela. Usando hilo normal, cose unas pocas vueltas sobre el sensor de inclinación para mantenerlo en su sitio. Une tres cables a la tela usando hilo conductor y haz las siguientes conexiones según muestra la siguiente ilustración:



Ahora podemos empezar con nuestro programa. En este programa vamos a encender el LED de la placa del Arduino cuando inclinamos el sensor.

```
int myTilt = 4;
  /* declarar el pin para el sensor de inclinación */
int ledPin = 13;
  /* declarar el pin para el led de la placa */
int tiltStatus = 0;
  /* declarar la variable para almacenar la inclinación
del sensor */

void setup(){
  pinMode(myTilt,INPUT);
  /* declarar el pin de inclinación como INPUT */
```

```
  pinMode(ledPin,OUTPUT);
  /* declarar el pin del LED como OUTPUT */
}

void loop(){
  tiltStatus = digitalRead(myTilt);
  /* leer el pin de inclinación y almacenar su valor */
  if(tiltStatus == HIGH){
  /* comprobar si el sensor de inclinación está inclinado
*/
  digitalWrite(ledPin,HIGH);
  /* encender el LED si el sensor está inclinado */
  }else{
  digitalWrite(ledPin,LOW);
  /* apagar el LED en cualquier otro caso */
}
}
```

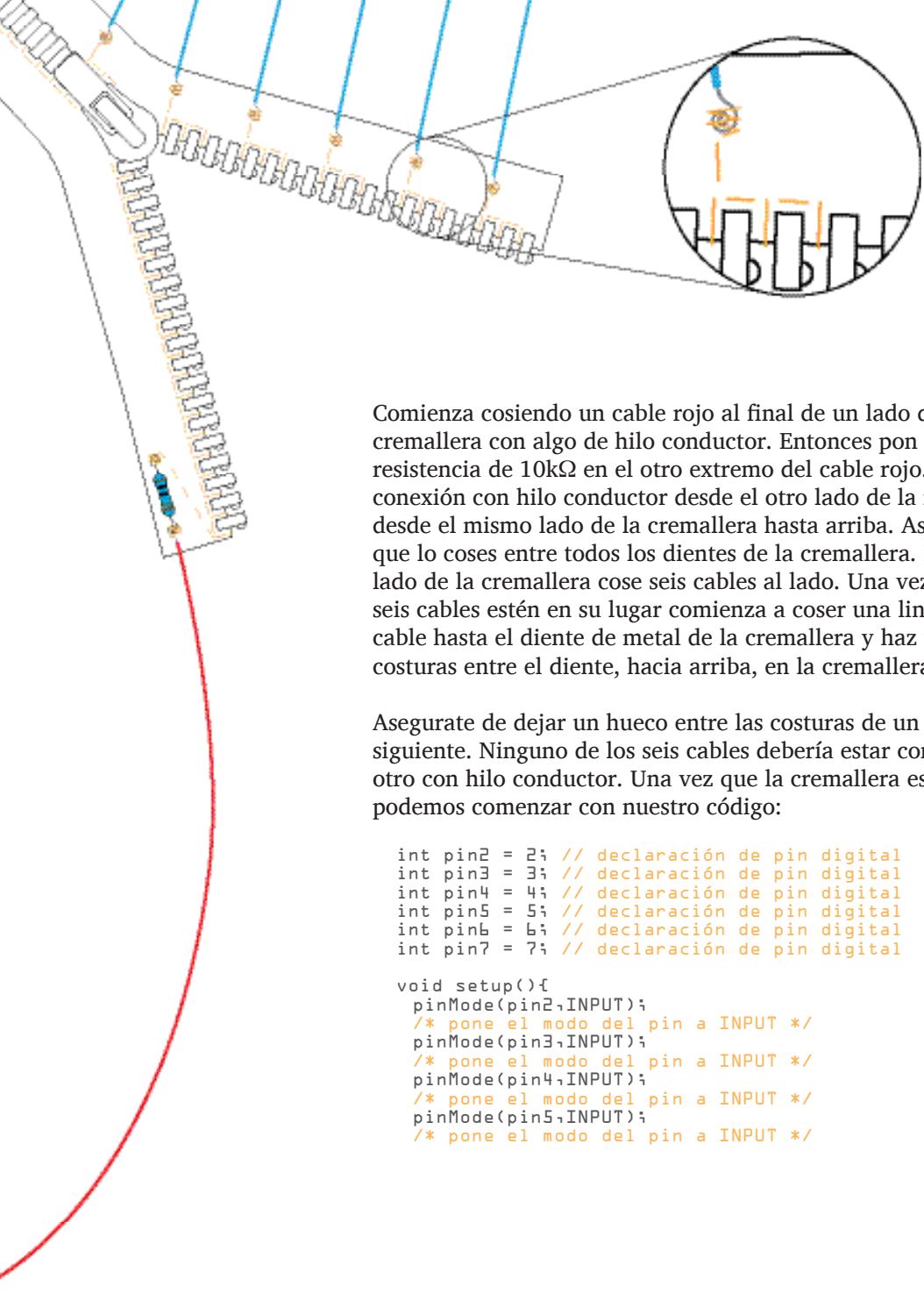
Una vez que has acabado con el código, subelo a la placa Arduino y conecta el sensor de inclinación como se muestra en la ilustración de la página anterior. Ahora cuando inclinas la tela el LED anexo al pin 13 de la placa Ardunio debería encenderse. Como se describe en la seción del sensor de inclinación del Capítulo 2 página 27, puedes usar este tipo de sensor de inclinación para detectar movimiento. Es una cuestión de estimar cuantos hits obtienes del sensor de inclinación durante un cierto tiempo. Digamos que el sensor de inclinación cambia de on a off 2 veces por segundo, entonces es seguro asumir que la prenda en la cual el sensor está puesto se está moviendo.

6: La cremallera digital

Hay dos formas en las que podemos usar una cremallera metálica como sensor de entrada. La diferencia entre ellas es como las leemos desde el Arduino. En este capítulo vamos a leer nuestra cremallera de forma digital. La ventaja de usar esta cremallera comparado con la cremallera digital de la página 51-54, es que la cremallera digital será más precisa en sus lecturas pero la desventaja es que tendremos un rango menor de valores para leer. El rango dependerá de cuandos pines digitales uses. En este ejemplo vamos a usar seis de los pines digitales de nuestro Arduino.

Para este ejemplo necesitarás:

- Cremallera metálica
- Hilo conductor.
- Resistencia de 10KΩ.



Comienza cosiendo un cable rojo al final de un lado de la cremallera con algo de hilo conductor. Entonces pon una resistencia de 10kΩ en el otro extremo del cable rojo. Haz una conexión con hilo conductor desde el otro lado de la resistencia y desde el mismo lado de la cremallera hasta arriba. Asegurate de que lo coses entre todos los dientes de la cremallera. En el otro lado de la cremallera cose seis cables al lado. Una vez que los seis cables estén en su lugar comienza a coser una linea desde el cable hasta el diente de metal de la cremallera y haz unas pocas costuras entre el diente, hacia arriba, en la cremallera.

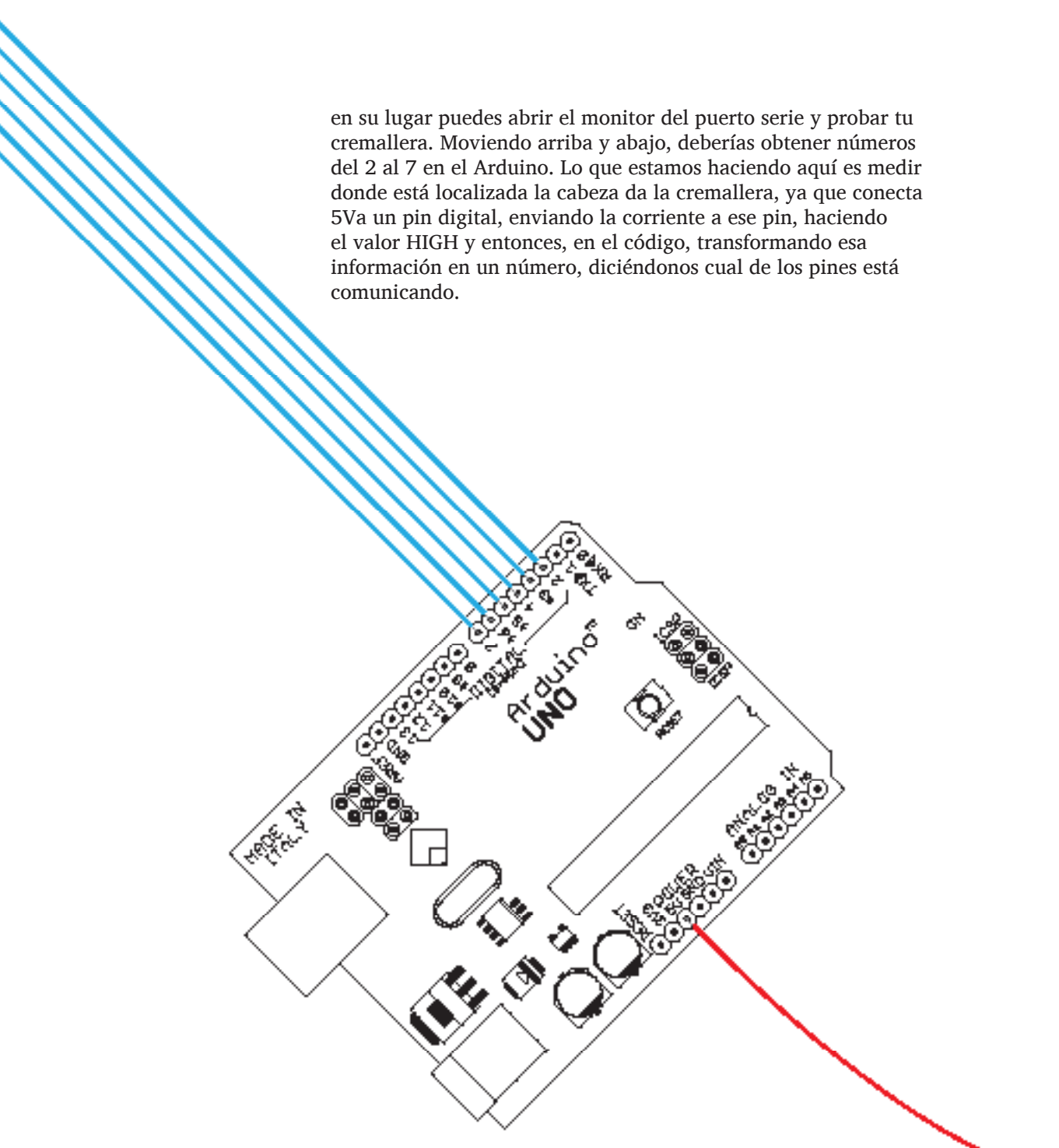
Asegurate de dejar un hueco entre las costuras de un cable al siguiente. Ninguno de los seis cables debería estar conectado a otro con hilo conductor. Una vez que la cremallera está cosida podemos comenzar con nuestro código:

```
int pin2 = 2; // declaración de pin digital
int pin3 = 3; // declaración de pin digital
int pin4 = 4; // declaración de pin digital
int pin5 = 5; // declaración de pin digital
int pin6 = 6; // declaración de pin digital
int pin7 = 7; // declaración de pin digital

void setup(){
 pinMode(pin2,INPUT);
 /* pone el modo del pin a INPUT */
 pinMode(pin3,INPUT);
 /* pone el modo del pin a INPUT */
 pinMode(pin4,INPUT);
 /* pone el modo del pin a INPUT */
 pinMode(pin5,INPUT);
 /* pone el modo del pin a INPUT */
```

```
  pinMode(pin6,INPUT);
  /* pone el modo del pin a INPUT */
  pinMode(pin7,INPUT);
  /* pone el modo del pin a INPUT */
  Serial.begin(9600);
  /* comienza la comunicación serie y establece la
velocidad en 9600 baud */
}
void loop(){
  if(digitalRead(pin2) == HIGH){
  /* comprueba si el pin está en HIGH */
  Serial.println(2);
  /* si está, envía el número de ese pin */
}

if(digitalRead(pin3) == HIGH){
  /* comprueba si el pin está en HIGH */
  Serial.println(3);
  /* si está, envía el número de ese pin */
}

if(digitalRead(pin4) == HIGH){
  /* comprueba si el pin está en HIGH */
  Serial.println(4);
  /* si está, envía el número de ese pin */
}

if( digitalRead(pin5) == HIGH){
  /* comprueba si el pin está en HIGH */
  Serial.println(5);
  /* si está, envía el número de ese pin */
}

if( digitalRead(pin6) == HIGH){
  /* comprueba si el pin está en HIGH */
  Serial.println(6);
  /* si está, envía el número de ese pin */
}

if(digitalRead(pin7) == HIGH){
  /* comprueba si el pin está en HIGH */
  Serial.println(7);
  /* si está, envía el número de ese pin */
}

delay(200);
  * hace una pausa */
}
```

Sube el código a tu placa Arduino y conecta la cremallera. El cable rojo va a el pin de 5V de la placa y el resto de los cables se conectan a los pines digitales 2 a 7. Una vez que todo esté

en su lugar puedes abrir el monitor del puerto serie y probar tu cremallera. Moviendo arriba y abajo, deberías obtener números del 2 al 7 en el Arduino. Lo que estamos haciendo aquí es medir donde está localizada la cabeza da la cremallera, ya que conecta 5Va un pin digital, enviando la corriente a ese pin, haciendo el valor HIGH y entonces, en el código, transformando esa información en un número, diciéndonos cual de los pines está comunicando.



Capítulo 6: Usando los pines analógicos

Como se mencionó anteriormente en este libro, el mundo real no es digital y, a veces, no puedes traducir los cambios en el entorno en lecturas digitales. Por ejemplo, la temperatura no cambia sólo de frío a caliente, cambia en un rango valores distintos y, normalmente, estos cambios ocurren muy lentamente. Éste es el motivo por el cual, a menudo, utilizamos sensores analógicos para leer los parámetros del entorno tales como, la temperatura, la luz, o el movimiento. Esta información resultante es almacenada como datos digitales secuenciales. Dado que Arduino no puede manejar la información como los humanos, necesitamos traducir los datos analógicos para que Arduino pueda entenderlos.

Los sensores analógicos pueden transformar los datos del entorno en un valor de voltaje comprendido entre 0V y 5V. Estos valores son distintos de los HIGH o LOW que utilizan los sensores digitales. Para los pines digitales, HIGH y LOW significan 5V y 0V respectivamente, y nada más. Sin embargo los pines analógicos pueden diferenciar cualquier valor intermedio.

La resolución entre los valores máximos y mínimos difieren de un microprocesador a otro. Arduino únicamente puede distinguir 1024 niveles en rango de 0V a 5V.

Nota:
Cuando hacemos una lectura analógica, Arduino comienza la cuenta en 0 por lo que nunca obtendremos valores superiores a 1023.

1: La cremallera analógica

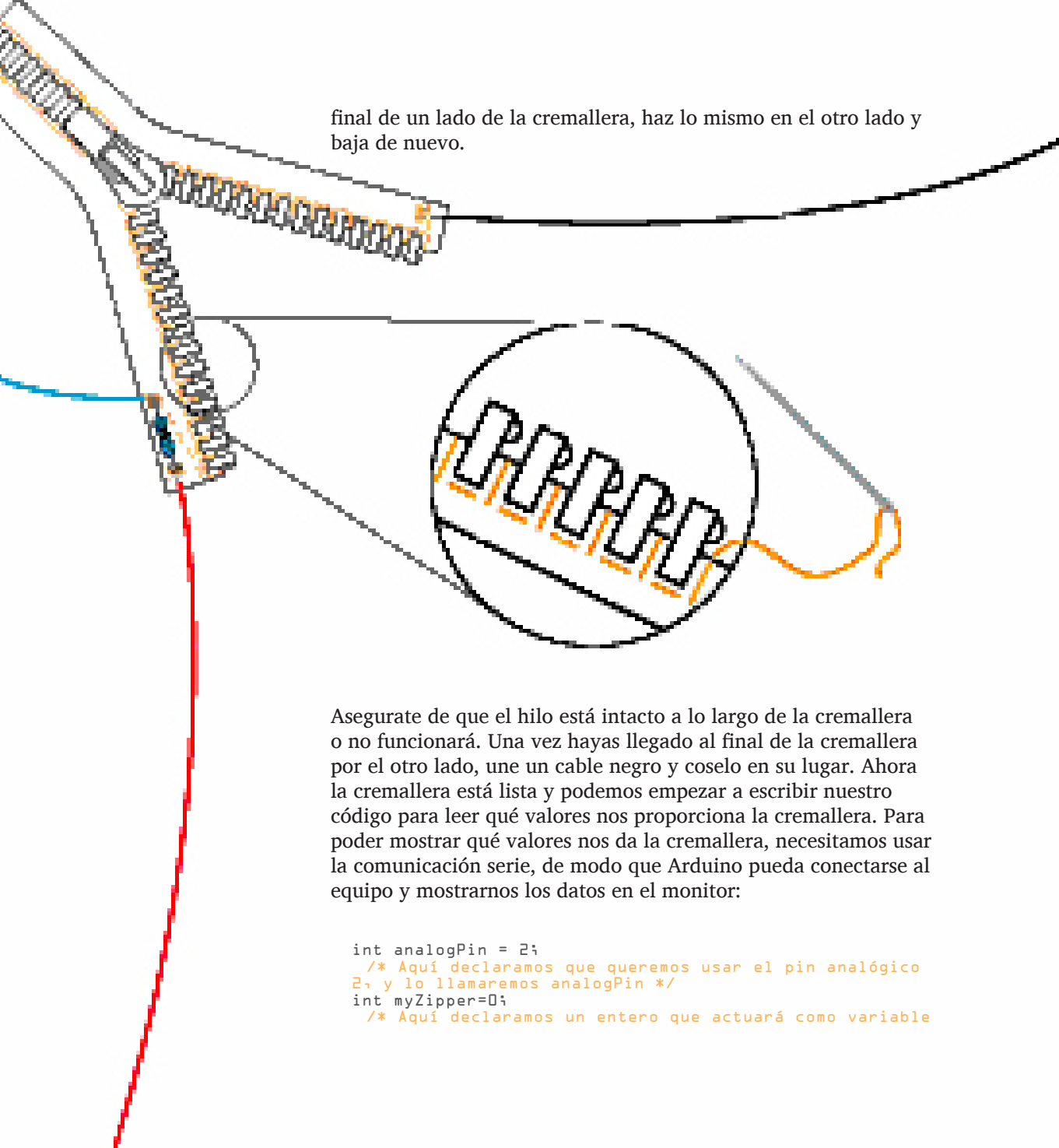
La cremallera analógica es diferente a la cremallera digital del capítulo anterior. La cremallera digital sólo podía estar HIGH o LOW (5V o 0V). sin embargo, la cremallera analógica, puede darte un rango de valores entre 0V y 5V.

Para hacer una cremallera analógica necesitas una cremallera normal, algo de hilo conductor y una resistencia de 10kΩ. Comience cosiendo la resistencia en su lugar. Después cose un cable rojo en uno de los terminales de la resistencia, y un cable azul en el otro terminal. Los colores de los cables son para que recuerdes dónde va cada cable. Desde donde empieza el cable azul, cóselo entre cada diente de la cremallera. Cuando llegues al

Para este ejemplo necesitarás:

- Una cremallera metálica.
- Hilo conductor.
- Una resistencia de 10kΩ.
- Dos cables.

final de un lado de la cremallera, haz lo mismo en el otro lado y baja de nuevo.



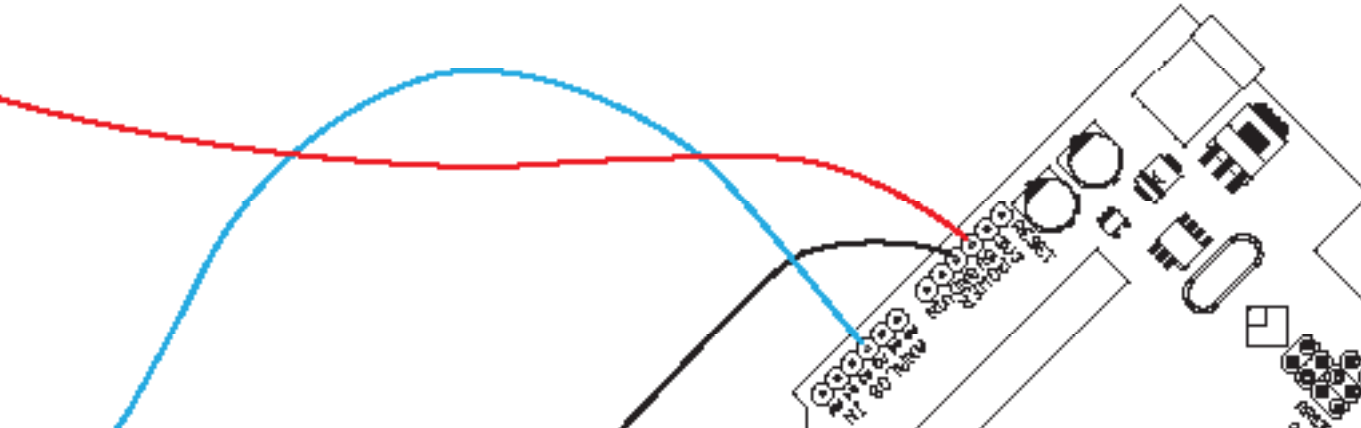
Asegurate de que el hilo está intacto a lo largo de la cremallera o no funcionará. Una vez hayas llegado al final de la cremallera por el otro lado, une un cable negro y coselo en su lugar. Ahora la cremallera está lista y podemos empezar a escribir nuestro código para leer qué valores nos proporciona la cremallera. Para poder mostrar qué valores nos da la cremallera, necesitamos usar la comunicación serie, de modo que Arduino pueda conectarse al equipo y mostrarnos los datos en el monitor:

```
int analogPin = 2;
  /* Aquí declaramos que queremos usar el pin analógico
2, y lo llamaremos analogPin */
int myZipper=0;
  /* Aquí declaramos un entero que actuará como variable
```

```
temporal, donde podremos almacenar los valores que nos
proporcione la cremallera */
void setup(){
  Serial.begin(9600);
  /* Esto configurará tu comunicación y la fijará a 9600
baudios (9600 bits por segundo) */
}

void loop(){
  myZipper = analogRead(analogPin);
  /* Aquí hacemos una lectura analógica de analogPin y
guardamos el valor en la variable myZipper */
  Serial.println(myZipper);
  /* Este comando enviará el valor almacenado en myZipper
a través del puerto serie y después hace una pausa de
200 milisegundos */
  delay(200); //pausa de 200 milisegundos}
```

Nota:
No hemos declarado el modo para el pin analógico tal y como deberíamos hacerlo si fuese un pin digital.

Lo primero que hacemos en void loop() es una lectura analógica del pin analógico 2. El valor es almacenado en la variable myZipper. Después imprimimos este valor por el puerto serie y, cualquier aparato conectado a Arduino (por ejemplo, un PC) recibirá el valor almacenado en la variable. Lo ultimo que hacemos es añadir una pausa en nuestro programa, ya que, aunque Arduino es pequeño, todavía se comunica más rapidamente que un ordenador normal, así que la pausa permite sincronizarse con Arduino. Ahora podemos subir nuestro código a la placa Arduino. Una vez que el código está subido, conecta el cable rojo en el puerto que indica 5V, el cable negro a uno de los puertos de GND, y el último cable al pin analógico 2. Si declaras tu analogPin en tu programa como cualquiero otro pin que no sea el 2, tendrás que colocar el cable azul en ese pin analógico:



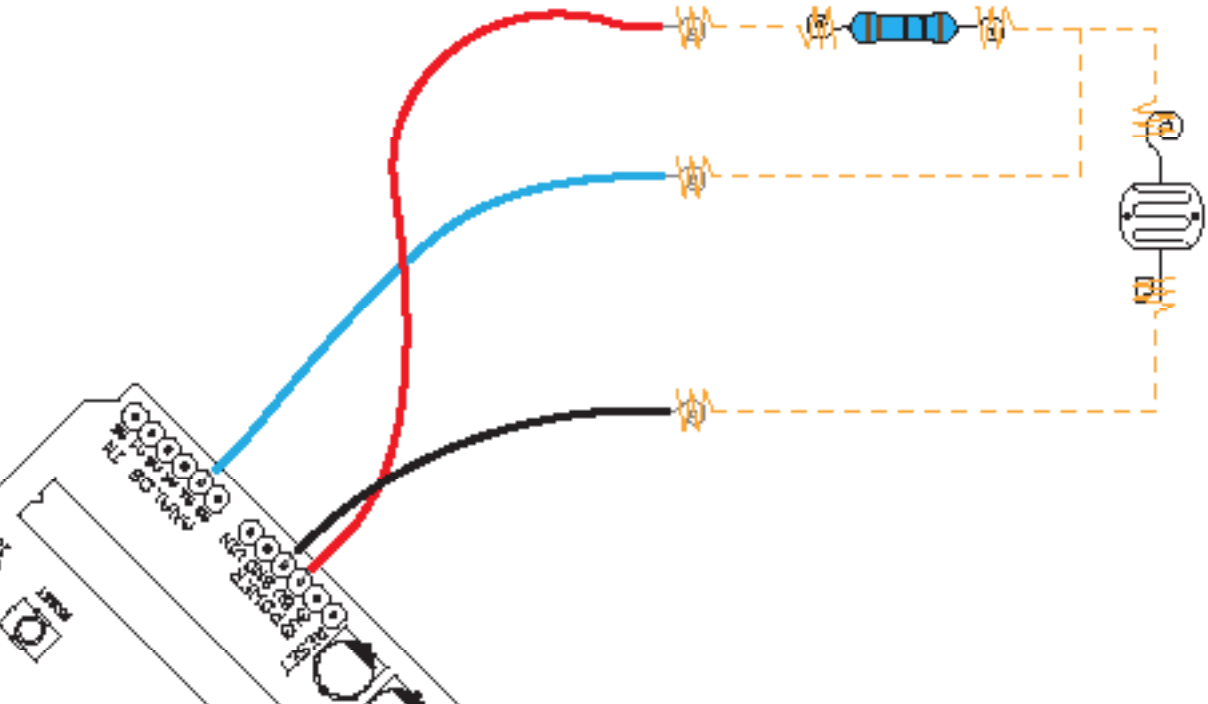
Una vez que hayas subido el código y conectado los cables, abre el monitor serie, que es el botón más a la derecha de tu IDE de Arduino. Ahora aparecerán los valores en el monitor y, si abres y cierras la cremallera, los valores irán cambiando. Si no puedes ver ningún número en tu monitor, asegurate que la velocidad de comunicación fijada en el IDE de Arduino es la misma que en tu programa (9600 en el ejemplo). Cada cremallera te dará unos valores diferentes, puesto que no es algo exacto; pequeñas diferencias afectarán en los resultados. Este tipo de sensor analógico casero es lo suficientemente bueno como para ser usado en la mayoría de prototipos que necesiten un dispositivo de entrada oculto.

Para este ejemplo necesitarás:

- Una LDR.
- Una resistencia de 10k.
- Hilo conductor.
- Tres cables.

2: Usando un sensor de luz LDR

El siguiente esquema muestra como coser una LDR y una resistencia en un trozo de tela y dónde deben conectarse los cables en el Arduino:



La resistencia usada en el ejemplo es lo que llamamos una resistencia “Pull up”. A veces usamos resistencias pull-up para asegurarnos que no recibe los 5V en sentido contrario el Arduino. Una resistencia Pull up es una resistencia normal pero usada para reducir el voltaje de salida de una fuente de tensión. La combinación de una resistencia con la LDR es necesaria por otra razón: sin ella no seríamos capaces de obtener valores analógicos de la LDR. Ésta configuración se conoce con el nombre de divisor de tensión y cambia el voltaje a la entrada del Arduino dependiendo de los cambios en la LDR. Conecta un cable rojo a una de las patillas de la resistencia y a los 5V de Arduino. Entre la resistencia y la LDR ponemos un cable que conecte a nuestro pin analógico. La misma conexión va a una patilla de la LDR. No importa que patilla conectes. De la otra patilla de la LDR cose un cable que conecte con el puerto GND del Arduino. Cuando hayas conectado todo prueba el siguiente código y observa los valores que proporciona la LDR:

```
int analogPin = 2;
 /* el pin analógico que usaremos en Arduino */
int myLDR = 0;
 /* variable temporal para almacenar los valores de la
LDR */
void setup(){
 Serial.begin(9600);
 /* configurando la conexión y la velocidad */
}

void loop(){
 myLDR = analogRead(analogPin);
 /* leemos el valor de la LDR y lo almacenamos */
 Serial.println(myLDR);
 /* mostramos el valor almacenado en myLDR */
 delay(200);
}
```

En el ejemplo anterior leemos el valor, lo almacenamos y lo mostramos en el ordenador. No olvides abrir tu monitor serie en el IDE de Arduino y configurarlo a 9600 baudios para poder ver los valores captados por Arduino. Una vez que todo funcione,

prueba a tapar la LDR con tu mano y comprueba que el valor que devuelve Arduino. Recuerda este valor y prueba el siguiente código de ejemplo.

```
int analogPin = 2;
 /* el pin analógico que usaremos en Arduino */
int myLDR = 0;
 /* variable temporal para almacenar el valor de la LDR
*/
int myDarkNumber = 100;
 /* el umbral de oscuridad, sustituir por el valor
obtenido tapando la LDR con la mano */
int ledPin = 13;

void setup(){
 Serial.begin(9600);
 /* configurando la conexión y la velocidad */
 pinMode(ledPin,OUTPUT);
 /* declaramos ledPin como Salida */
}

void loop(){
 myLDR = analogRead(analogPin);
 /* leemos el valor de la LDR y lo almacenamos */
 if (myLDR <= myDarkNumber){
 digitalWrite(ledPin,HIGH);
 }else{
 digitalWrite(ledPin,LOW);
}
}
```

En este ejemplo la variable “myDarkNumber” es el valor que obtuviste al tapar la LDR con la mano en el ejemplo anterior. En mi caso es 100, pero tú debes cambiarlo por tu valor. Este programa leerá el valor de la LDR y lo comparará con la variable del umbral (myDarkNumber) y si la LDR es menor o igual al umbral el LED interno de la placa Arduino se ecenderá, de no ser así permanecerá apagado.

Para este ejemplo necesitarás:

- Una NTC.
- Una resistencia de 10k.
- Hilo conductor.
- Tres cables.

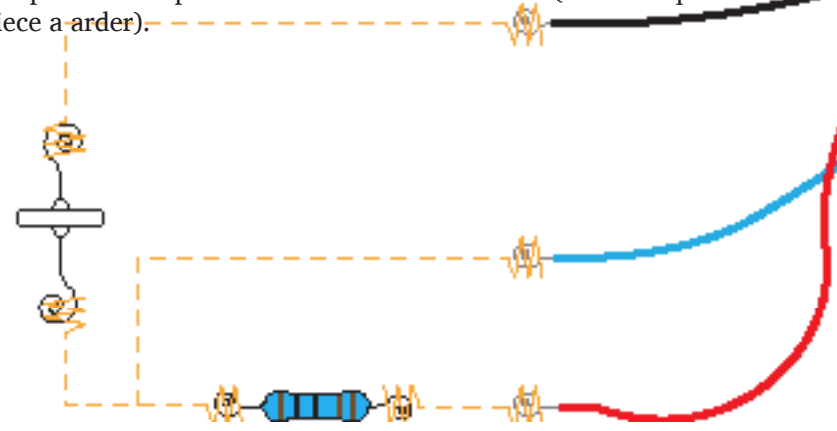
3: Usando un sensor de temperatura NTC

El código siguiente, para leer un termómetro, es el mismo que para la LDR. Ambos funcionan bien por tratarse de sensores analógicos:

```
int analogPin = 2;
 /* el pin analógico que usaremos en Arduino */
```

```
int myNTC= 0;
  /* variable temporal para guardar los valores del
termómetro */

void setup(){
  Serial.begin(9600);
  /* configurando la comunicación y la velocidad */
}

void loop(){
  myNTC = analogRead(analogPin);
  /* lee el valor del NTC y lo almacena */
  Serial.print(myNTC);
  /* muestra el valor almacenado en myNTC */
delay(200);
}
```

Ahora conecta todo como muestra la siguiente ilustración y carga el código en tu placa Arduino. Una vez que todo esté en su lugar, abre tu monitor serie e intenta calentar tu termómetro con las manos. Debes poder observar variaciones frente al valor original, cuando comenzó el programa. Aunque los termistores no sean los mejores sensores para obtener la temperatura exacta, son una manera fácil y barata de diferenciar entre frío y caliente. Si estás usando altas corrientes en tus prendas, o materiales que se calienten como “muscle wires”, los termistores son una manera de comprobar si tu prenda está demasiado caliente (antes de que empiece a arder).

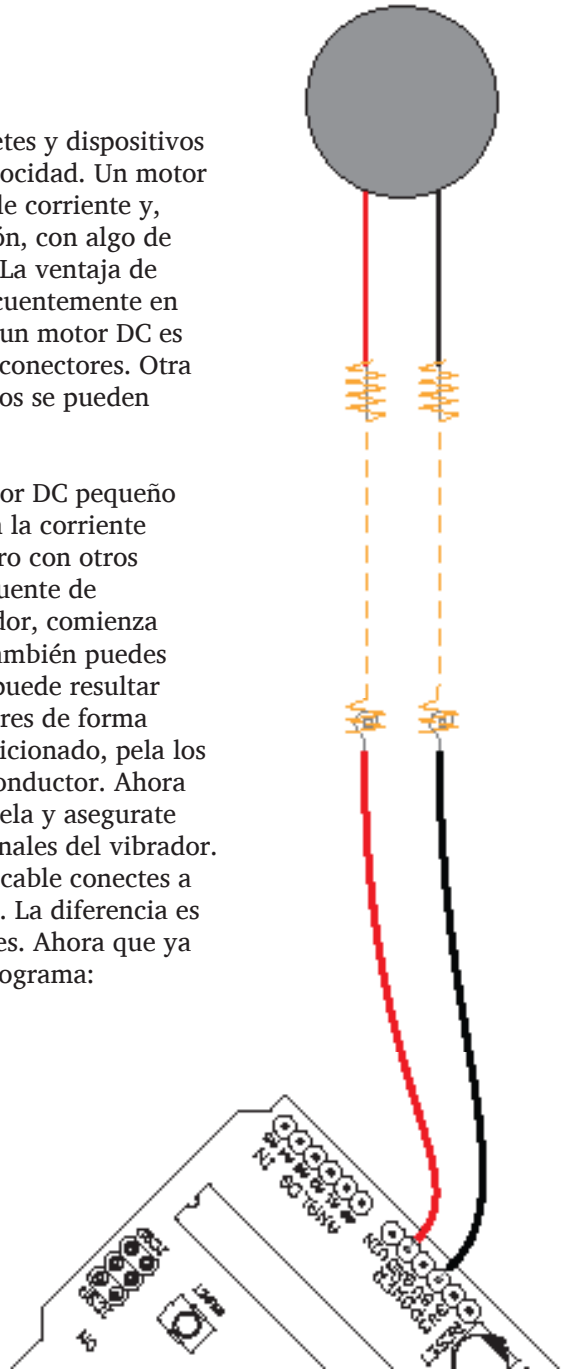


Capítulo 7: Moviendo cosas

• **Motores DC**
Es común encontrar este tipo de motor en juguetes y dispositivos que no necesitan gran precisión en ángulo o velocidad. Un motor DC (o de “Continua”) gira libremente al aplicarle corriente y, si bien es difícil controlar su velocidad y posición, con algo de circuitería extra se puede mejorar su precisión. La ventaja de un motor DC es su coste, por lo que se usan frecuentemente en aparatos electrónicos móviles baratos. Hackear un motor DC es bastante sencillo puesto que sólo consta de dos conectores. Otra ventaja de los motores DC es que algunos de ellos se pueden alimentar con menos de 2.5V.

El ejemplo siguiente muestra como usar un motor DC pequeño del tipo vibrador. En este caso puede bastar con la corriente propocionada por un pin digital de Arduino, pero con otros motores DC necesitarás de un transistor y una fuente de alimentación externa. Para coser un mini vibrador, comienza por unirlo con pegamento a un trozo de tela. También puedes coser sobre él usando hilo normal, si bien esto puede resultar complicado al ser la mayor parte de los vibradores de forma cilíndrica. Una vez se encuentre el vibrador posicionado, pela los cables y cose los contactos metálicos con hilo conductor. Ahora cose los dos cables, uno rojo y otro negro, a la tela y asegurate que cada uno de ellos va a uno de los dos terminales del vibrador. Como se trata de un motor DC, no importa que cable conectes a que terminal, ni la forma en la que lo alimentes. La diferencia es que girará en un sentido u otro según lo conectes. Ahora que ya tienes todo en su sitio, introduce el siguiente programa:

```
int motorPin = 5;

void setup(){
  pinMode(motorPin, OUTPUT);
}

void loop(){
  digitalWrite(motorPin, HIGH);
  delay(1000);
```



```
  digitalWrite(motorPin, LOW);
  delay(1000);
}
```

Este es un programa simple para encender el vibrador por un segundo y apagarlo por un segundo. Es también posible modificar la velocidad de vibración haciendo uso de PWM (modulación por ancho de pulsos):

```
int motorPin = 5;
  /* connecta tu motor al pin 5 PWM */

void setup(){
  /* nada pasa en el setup */
}

void loop(){
  for(int i = 0; i <= 255; i++){
  /* incrementa una unidad, mientras sea menor de 255 */
  analogWrite(motorPin,i);
  /* modifica la velocidad de vibracion con i */
  delay(30);
  /* haz una pequeña pausa para percibir el cambio */
}

for(int i = 255; i >= 0; i--){
  /* mientras i sea mayor de 0, decrementa una unidad */
analogWrite(motorPin,i);
  /* modifica la velocidad de vibracion con i */
  delay(30);
  /* haz una pequeña pausa para percibir el cambio */
}
}
```

El ejemplo anterior incrementará la velocidad del vibrador hasta el máximo para retornar luego al mínimo.

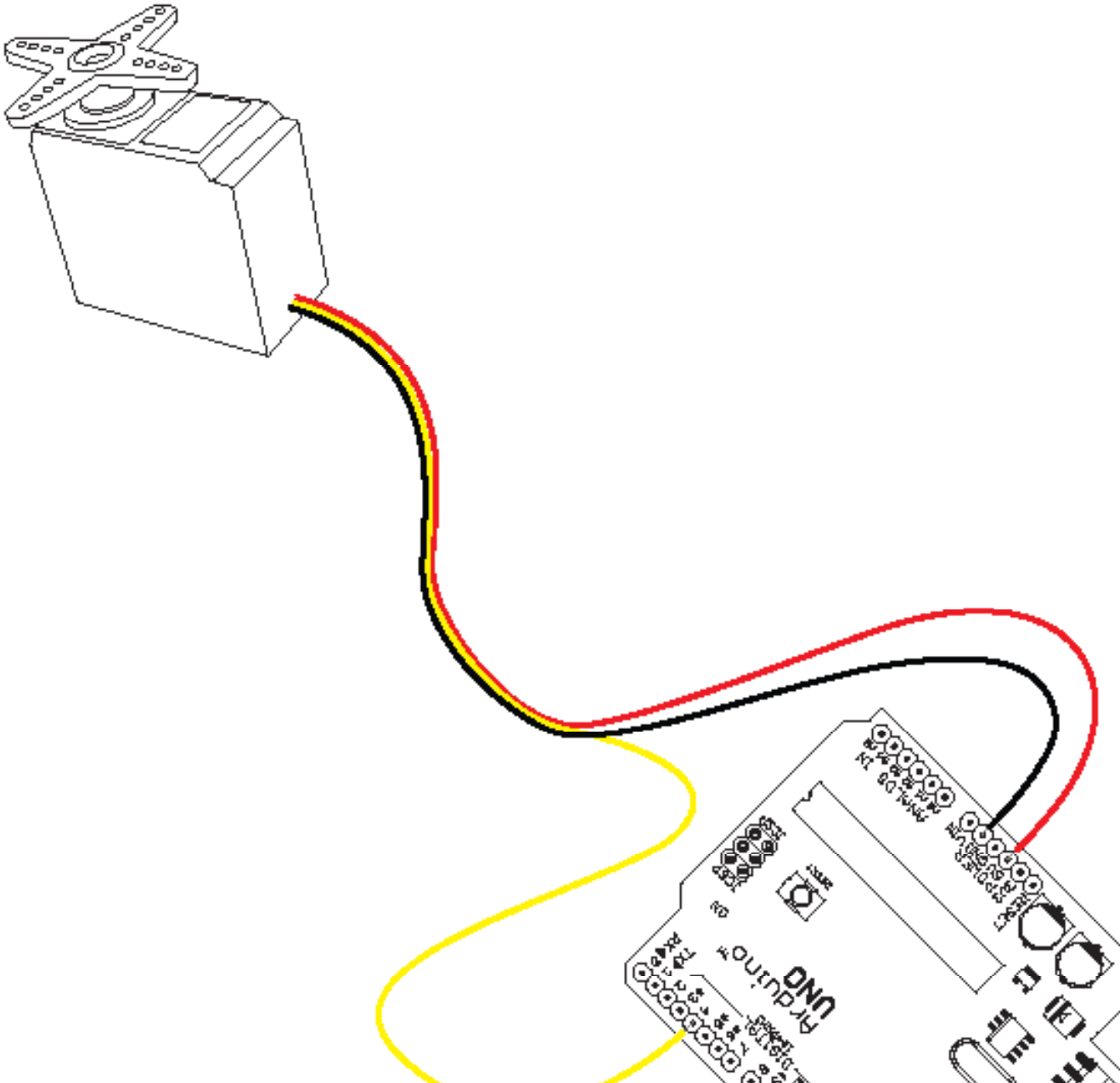
• **Servo motores**
Existed dos tipos de servo motores: normales y de giro continuo. Los servos normales se controlan con PWM y giran a una posición fija dependiendo en el ancho de pulso. Están limitados en movimiento a un rango entre 0 y 180 grados. Los servos de giro continuo se mantienen girando mientras que reciban pulsos PWM. Los más comunes, fabricados por Parallax, giran en una dirección cuando se les proporcionan pulsos por encima de 1500

microsegundos de ancho y en la contraria al darles un ancho menor al citado.
Dependiendo del tipo de proyecto que realices tendrás que usar tu imaginación para ocultar un servo motor. En el ejemplo siguiente mostraremos sólo como controlar el motor, puesto que no hay un método generalizado para utilizar servos con textiles. Es posible extender la longitud de los cables del servo hasta Arduino usando hilo conductor.

El primer ejemplo es un programa simple que ilustra como hacer rotar a un servo de giro continuo adelante y atrás:

```
int motorPin = 4;
 /* pin digital para el servo */
void setup(){
 pinMode(motorPin,OUTPUT);
 /* declara el pin digital como salida */
}

void loop(){
 for(int i = 0; i < 100; i++);{
 /* repite 100 veces para que veamos al servo moverse */
 digitalWrite(motorPin,HIGH);
 delayMicroseconds(1850);
 /* el delay aqui mostrado establece un ancho de pulso
*/
 digitalWrite(motorPin,LOW);
 delayMicroseconds(1850);
}

for(int j = 0; j < 100; j++);{
 /* repite 100 veces para que veamos al servo moverse */
 digitalWrite(motorPin,HIGH);
 delayMicroseconds(1250);
 /* el delay aqui mostrado establece un ancho de pulso
*/
 digitalWrite(motorPin,LOW);
 delayMicroseconds(1250);
}
}
```

Este programa hará que el servo de rotación continua gire 100 pasos en una dirección y 100 en la contraria. Una vez hayas subido el código a tu placa Arduino, conecta el motor tal y como se muestra en la siguiente ilustración.


Arduino
UNO

Capítulo 8: Ejemplos complejos

1: Oscilación con una cremallera

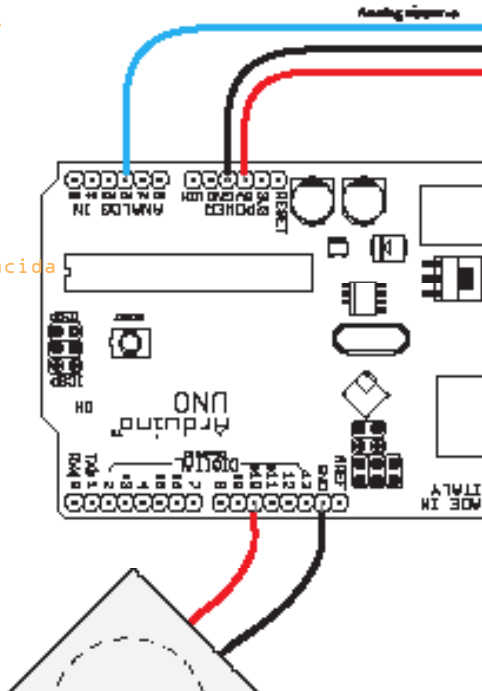
La oscilación es la variación en el tiempo de un valor central o entre dos o más estados. Con un oscilador podemos cambiar manualmente el tono de nuestro altavoz piezoeléctrico del ejemplo anterior. La cremallera analógica funcionará perfectamente como un oscilador dado que nos da un buen rango de valores que pueden ser reasignados a otro rango de tonos diferentes.

El siguiente programa leerá el valor de la cremallera, reasignará los valores y los usará para establecer el tono que será reproducido por el altavoz piezoeléctrico:

```
int piezoPin = 10;
  /* declara el pin al que el altavoz piezoeléctricos
está conectado */
int analogPin = 2;
  /* declara el pin al que la cremallera está conectada
*/
int myTemp = 0;
  /* declara una variable temporal para almacenar */
int myRemapValue = 0;
  /* declara una variable para el valor reasignado */

void setup(){
  pinMode(piezoPin,OUTPUT);
  /* declara piezoPin como OUTPUT */
}

void loop(){
  myTemp = analogRead(analogPin);
  /* lee y almacena el valor de la cremallera */
  myRemapValue = map(myTemp,100,300,0,2000);
  /* reasigna el valor de la cremallera para que coincida
en el rango de 0 a 2000 */
  digitalWrite(piezoPin,HIGH);
  /* envía 5V al altavoz piezoeléctrico */
  delayMicroseconds(myRemapValue);
  /* pausa con el valor reasignado */
  digitalWrite(piezoPin,LOW);
  /* envía 0V al altavoz piezoeléctrico */
  delayMicroseconds(myRemapValue);
  /* pausa con el valor reasignado otra vez */
}
```

Para este ejemplo necesitarás
- El altavoz piezo eléctrico de las página 44-45.
- La cremallera analógica de las página 51-54.

Nota:
Necesitas conocer el máximo y mínimo valor que obtenidos en la cremallera. En este ejemplo usaremos un rango ficticio entre 100 y 300. Puedes usar el ejemplo AnalogInSerial que viene con el software de Arduino para leer los valores de la cremallera.



Este ejemplo implementa la función map(). Para aprender más acerca de la función map() ve a la página 86-87. Tu cremallera podría dar valores que son suficientemente buenos para poner algunos tonos directamente, pero tendrían un rango menor que lo que queríamos, ya que un sensor analógico no da valores por encima de 1023.

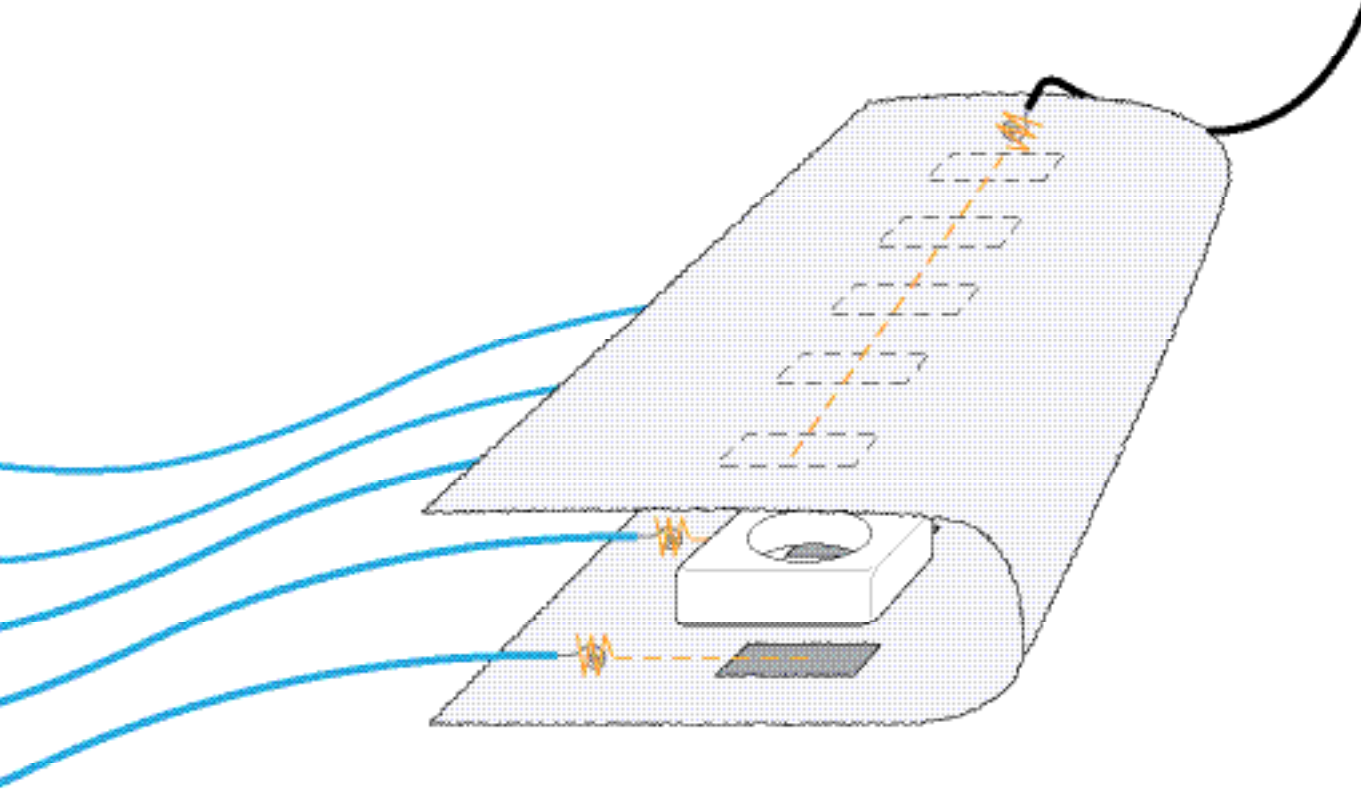
Una vez que tu código está completo, prueba a conectarlo todo a la placa Arduino como se muestra en la figura y prueba el oscilador cremallera.

Para este ejemplo necesitarás:

- Cinco pulsadores.
- Un altavoz piezo eléctrico..

2: El sintetizador flexible

Usa el mismo método para crear un botón pulsador que se muestra en el ejemplo de las páginas 41-42. Para este ejemplo usaremos una gran pieza de tela para hacer espacio suficiente



para todos los botones pulsadores.

En un lado de los botones conecta 5 cables y cose una conexión a una pieza de tela conductoradesde un cable a otro para todos los botones pulsadores. Desde el otra lado de los botones pulsadores, hasta la otra pieza de tela conductora, cose una conexión a un trozo largo de hilo y al final de este hilo conecta un cable negro. Esto actuará como conexión tierra para todos los botones, ya que no hay suficientes puertos GND en el arduino para todos los botones.

Ahora que tenemos listos los botones pulsadores para nuestro sintetizador flexible, podemos empezar con nuestro programa. Pon especial atención a la función de setup. Aquí configuramos los 5 botones a 5V. Podemos hacer esto porque el Arduino tiene resistencias internas de pullup. Esto significa que añadiendo estas líneas de código no necesitamos usar resistencias extra para poder leer los valores de los pines digitales:

```
int piezoPin = 10 ;
 /* declara el pin al cual el altavoz piezoeléctrico
está conectado */
int keyOne = 2;
 /* declara el primer botón pulsador */
int keyTwo = 3;
 /* declara el segundo botón pulsador */
int keyThree = 4;
 /* declara el tercer botón pulsador */
int keyFour = 5;
 /* declara el cuarto botón pulsador */
int keyFive = 6;
 /* declara el quinto botón pulsador */


void setup(){
 pinMode(piezoPin,OUTPUT);
 /* declara piezopin como output */
 pinMode(keyOne,INPUT);
 /* declara el primer botón pulsador como input */
 pinMode(keyTwo,INPUT);
  /* declara el segundo botón pulsador como input */
 pinMode(keyThree,INPUT);
 /* declara el tercer botón pulsador como input */
 pinMode(keyFour,INPUT);
 /* declara el cuarto botón pulsador como input */
```

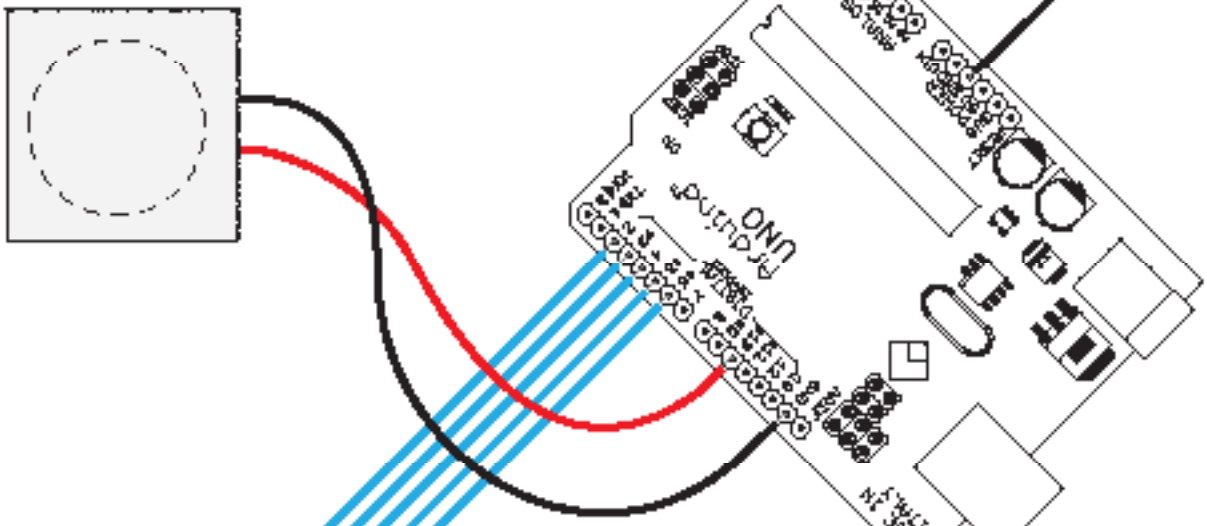
```
  pinMode(keyFive,INPUT);
  /* declara el quinto botón pulsador como input */
  digitalWrite(keyOne,HIGH);
  /* envía 5V al primer botón pulsador */
  digitalWrite(keyTwo,HIGH);
  /* envía 5V al segundo botón pulsador */
  digitalWrite(keyThree,HIGH);
  /* envía 5V al tercer botón pulsador */
  digitalWrite(keyFour,HIGH);
  /* envía 5V al cuarto botón pulsador */
  digitalWrite(keyFive,HIGH);
  /* envía 5V al quinto botón pulsador */
}

void loop(){
  if(digitalRead(keyOne) == LOW){
  /* comprueba si el primer botón pulsador es pulsado */
 digitalWrite(piezoPin,HIGH);
  /* envía 5V al altavoz piezoeléctrico */
  delayMicroseconds(1911);
  /* espera 1911 microsegundos (crea una C) */
  digitalWrite(piezoPin,LOW);
  /* para de enviar 5V al altavoz piezoeléctrico */
delayMicroseconds(1911);
  /* espera otros 1911 microsegundos */
}

if(digitalRead(keyTwo) == LOW){
  /* comprueba si el segundo botón pulsador es pulsado */
 digitalWrite(piezoPin,HIGH);
  /* envía 5V al altavoz piezoeléctrico */
  delayMicroseconds(1703);
  /* waits for 1703 microseconds (creates a D) */
  digitalWrite(piezoPin,LOW);
  /* para de enviar 5V al altavoz piezoeléctrico */
  delayMicroseconds(1703);
  /* espera otros 1703 microsegundos */
}

if(digitalRead(keyThree) == LOW){
  /* comprueba si el tercer botón pulsador es pulsado */
 digitalWrite(piezoPin,HIGH);
  /* envía 5V al altavoz piezoeléctrico */
  delayMicroseconds(1517);
  /* espera 1517 microsegundos (crea una E) */
  digitalWrite(piezoPin,LOW);
  /* para de enviar 5V al altavoz piezoeléctrico */
  delayMicroseconds(1517);
  /* espera otros 1517 microsegundos */
}

if(digitalRead(keyFour) == LOW){
  /* comprueba si el cuarto botón pulsador es pulsado */
  digitalWrite(piezoPin,HIGH);
  /* envía 5V al altavoz piezoeléctrico */
```

```
  delayMicroseconds(1432);
  /* espera 1432 microsegundos (crea una F) */
  digitalWrite(piezoPin,LOW);
  /* para de enviar 5V al altavoz piezoeléctrico */
  delayMicroseconds(1432);
  /* espera otros 1432 microsegundos */
}

if (digitalRead(keyFive) == LOW){
  /* comprueba si el quinto botón pulsador es pulsado */
  digitalWrite(piezoPin,HIGH);
  /* envía 5V al altavoz piezoeléctrico */
  delayMicroseconds(1276);
  /* espera 1276 microsegundos (crea una G) */
  digitalWrite(piezoPin,LOW);
  /* para de enviar 5V al altavoz piezoeléctrico */
  delayMicroseconds(1276);
  /* espera otros 1276 microsegundos */
}
}
```

Nota: Los cinco cables azules van del pin 2 al 6 en la placa de Arduino. El cable negro se conecta al GND. EL cable negro del piezo eléctrico se conecta también a uno de los pines GND pins y el cable rojo se conecta al pin digital 10.

Este programa comprueba todos los botones. Tan pronto como uno de ellos es pulsado comenzará a reproducir el tono de esa tecla. En este ejemplo vamos a usar los tonos C,D,E,F y G. Al final de este capítulo puedes encontrar una tabla de tonos y sus correspondientes tiempos de retardo.

Una vez que el programa está listo, súbelo a tu placa Arduino y conecta tu teclado flexibe y altavoz piezoeléctrico como se ve en la siguiente ilustración:



El altavoz piezoeléctrico podría estar también implementado en el teclado de tela para hacer tu diseño más estético.

Tabla de tonos y sus correspondientes tiempos de retardo:

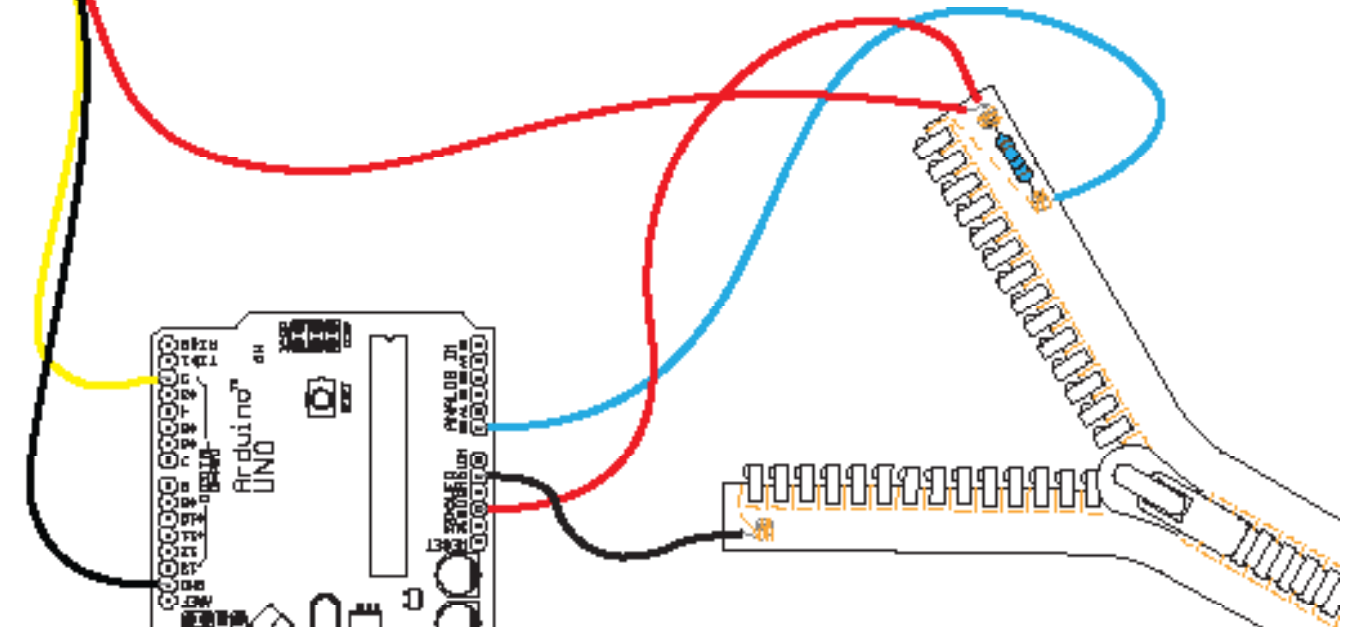
Tono	Retraso
C	1911
D	1703
E	1517
F	1432
G	1276
A	1136
B	1012
C	0956

Para este ejemplo necesitarás:

- La cremallera analógica de las páginas 51-54.
- Un servo motor normal.

3: Controlando un servo normal con una cremallera

En este ejemplo usaremos la cremallera analógica de las páginas 51-54 para controlar la rotación de un motor servo normal. La siguiente ilustración muestra como conectar ambos, el servo y la cremallera:



Para escribir el programa primero tienes que saber los valores máximo y mínimo de tu cremallera. Ve a las páginas 51-54 si no recuerdas como leer los valores de un sensor analógico.

Los valores mínimo y máximo de la cremallera serán remapeados en nuestro programa usando la función map() a los valores de rotación mínimo y máximo de nuestro motor servo:

```
int servoPin = 2;
  /* pin digital para el motor servo */
int servoMin = 500;
  /* posición mínima del servo */
int servoMax = 2500;
 /* posición máxima del servo */
int pulse = 0;
 /* cantidad para pulsar el servo */
long lastPulse = 0;
 /* el tiempo en milisegundos de el último pulso */
int refreshTime = 20;
 /* el tiempo necesario entre pulsos */
int myZipper = 0;
 /* el valor devuelto desde el sensor analógico */
int analogPin = 0;
 /* el pin analógico al que el sensor está conectado */

void setup(){
 pinMode(servoPin,OUTPUT);
 /* configura el pin del servo como output */
 pulse = servoMin;
 /* establece la posición del motor al mínimo */
 Serial.begin(9600);
 /* configura la velocidad de comunicación serie */
}

void loop(){
 myZipper = analogRead(analogPin) ;
 /* lee la entrada analógica */
 pulse = map(myZipper,0,1023,servoMin,servoMax);
 /* reasigna el valor de la cremallera al rango de
valores comprendido entre servoMin y servoMax */
 if (millis() - lastPulse >= refreshTime) {
 /* si milis (el cual es la cantidad de milisegundos que
el arduino ha estado encendido) menos el valor
desde "lastPulse" es mayor o igual a el valor en
"refreshTime" entonces esta parte de código es ejecutada
*/
 digitalWrite(servoPin,HIGH);
 /* enciende el motor */
 delayMicroseconds(pulse);
 /* la longitud del pulso establece la posición del
motor */
```

```
    digitalWrite(servoPin,LOW);
    /* apaga el motor */
    lastPulse = millis();
    /* guarda el tiempo del último pulso */
  }
}
```

Este programa leerá el valor de la creamllera y asignará este valor en el pulso que se usará para establecer la posición del motor. Una vez que hayas acabado con el código, súbelo a tu placa y prueba a mover la cremallera para mover el motor servo. En este programa tienes que añadir los valores máximo y mínimo de tu cremalleraen la función map(myZipper, your zipper min, your zipper max, 500, 2500);.

Para este ejemplo necesitarás:

- Three cables. Un chip QT118.
- Un condensador de 33n.
- Hilo conductor.
- Tres cables.

4: Sensor bordado táctil

En este ejemplo vamos a fabricar un bordado táctil, el cual hará que el bordado actue como un botón. Para hacer bordados necesitarás usar un chip extra. El chip QT118 es un sensor táctil que consta de un circuito integrado y es capaz de detectar proximidad o toque. Hay diferentes tipos de chips QT y el QT118 que vamos a usar somo puede manejar una entrada sensitiva. El chip QT funciona como un interruptor on/off y puedes la sensibilidad táctil desde el chip con cualquier material conductor como un cable o un hilo conductor. La inlustración siguiente muestra las patas de un chip QT118 y donde necesitan ser conectadas.

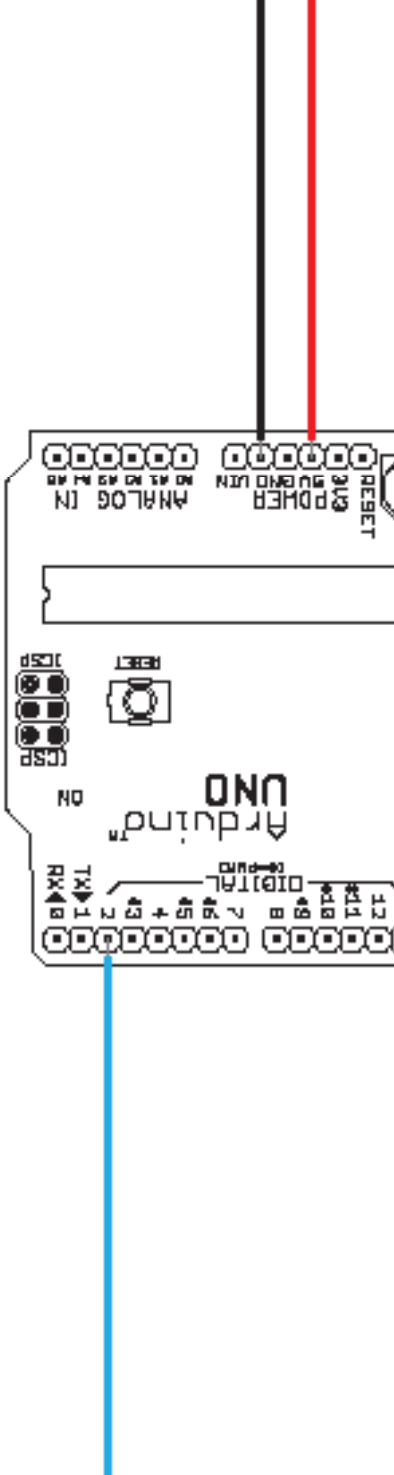
Para hacer un bordado sensible, necesitamos añadir un condensador. El condensador estabilizará la señal que viene del chip QT hacia nuestro bordado para permitirnos generar un valor de referencia. El chip QT usará este valor de referencia para comparar la señal cuando el bordado es tocado de cuando no. La ilustración en la izquierda muestra como conectar el condensador al chip y donde puedes empezar con tu bordado:

Todas las partes que quieras que sean sensibles al tacto tienen

que estar conectadas al hilo que conecta al pin del chip. Conecta los siguientes cables al chip. Una vez que todas las conexiones al chip hayan sido hechas, podemos empezar a escribir nuestro programa. El QT118 enviará una señal de 5V cuando sea tocado. Nosotros podemos leer esta señal en ambos pines, analógicos y digitales, pero si tienes pines digitales disponibles, no tiene sentido usar pines analógicos ya que la señal solo será leida como 0 (el chip no es tocado) o 1023 (el chip es tocado). El siguiente programa es un sketch simple de como encender el LED de la placa cuando el bordado es tocado:

```
int ledPin = 13;
  /* el LED en el Arduino */
int touchChip = 2;
  /* conecta la señal de salida del chip táctil al pin 2
*/


void setup(){
  pinMode(ledPin, OUTPUT);
  /* declara ledPin como OUTPUT */
  pinMode(touchChip, INPUT);
  /* declara touchChip como INPUT */
}

void loop(){
  if (digitalRead(touchChip) == HIGH){
  /* comprueba si el bordado es tocado */
  digitalWrite(ledPin, HIGH);
  /* si es así, encender el LED */
  }else{
  digitalWrite(ledPin, LOW);
  /* sino, apagar el LED */
}
}
```

Una vez acabado el código, podemos subirlo a la placa. Entonces conectamos el bordado al el chip como en la ilustración. El bordado será sensible en ambos lados, por eso, si quieres se capaz de llevar un bordado sensible necesitas poner algo de tela protectora dentro de tu prenda. Podrías experimentar con algunos tejidos para ver cual es suficientemente grueso para tu bordado y asegurarte que la tela que estás usando no es conductora



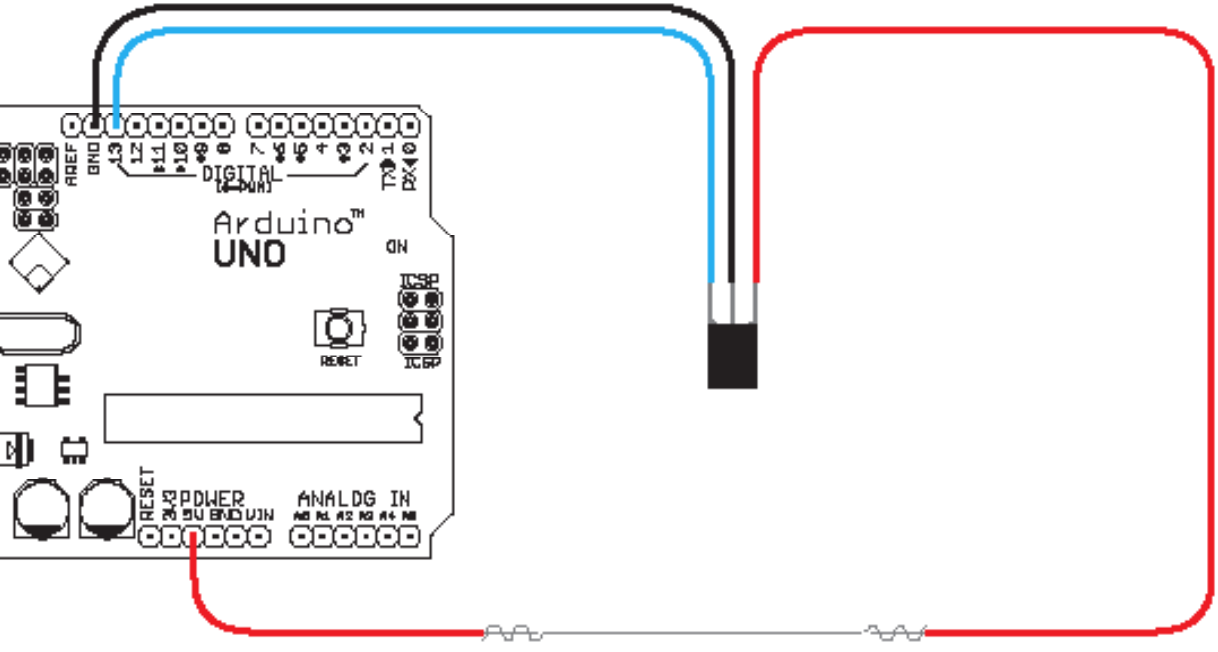
Para este ejemplo necesitarás:

- 25 cm de "trained muscle wire" de 0.15 mm de espesor.
- 2 cierres de presión.
- Un transistor 2N4401.
- 4 cables.

5: Músculo artificial

Usar músculos artificiales es una forma muy conveniente de crear movimiento en telas sin tener que incorporar pesados motores. Los músculos artificiales están hechos de un metal llamado Nitinol que puede recordar una forma. Cuando tu calientas el cable con un secador de pelo o con electricidad, él vuelve a al forma que se le hizo recordar. Mira si puedes conseguir cable que haya sido entrenado para contraerse en forma de espiral, porque esta forma será la que genere mayor movimiento.

Cose el músculo artificial a mano en la tela. Usa una tela fina, un pequeña pieza de tela o una pieza de tela que solo esté conectada en un lado, como un cordón. Esto asegurará que cuando el cables esté activado, tu todavía podrás el movimiento de la tela cuando estés un podo alejado de él. El músculo artificial no puede ser soldado, así que las anillas son usadas para conectar ambos lados del músculo artificial a los cables normales. Para hacer una conexión sólida pon una lado del músculo artificial y cable normal a través de una anilla y, usando unas tenazas planas, aprétalos juntos.



Comprueba que ambos cables no se escapan de la anilla chafada. Los músculos artificales usan mucha corriente, más que el Arduino puede suministrar. Para que el Arduino pueda encender el músculo artificial desde uno de sus pines tienes que usar un transistor. La ilustración muestra como concetar un músculo artificial a los cables normales, y los cables normales al transistor y al Arduino. Usa este código para hacer que la tela se mueva:

```
int musclePin = 13;

void setup(){
  pinMode(musclePin, OUTPUT);
}

void loop(){
  digitalWrite(musclePin, HIGH);
  /* enciende el pin para hacer que el músculo artificial
se caliente y se contraiga a su forma aprendida */
  delay(4000);
  /* primer retardo */
  digitalWrite(musclePin, LOW);
  /* apaga el pin para permitir al músculo artificial
enfriarse y contraerse */
  delay(4000);
  /* segundo retardo */
}
```

Los músculos artificiales con un grosor de 0.15mm solo tienen un sutil efecto en la mayoría de telas, porque no es muy fuerte. Un músculo artificial más grueso tiene un efecto mayor, pero también necesita más corriente. Si eliges usar un músculo artificial más grueso de 0.15mm asegúrate de que no lo dejas encendida mucho tiempo, ya que podrías sobrecalentarlo.

Los músculos artificiales de 0.15mm o más finos no tienen este problema, y pueden dejarse encendidos el tiempo que se quiera. Si quieres mover un músculo artificial más largo de 25 cm en este ejemplo, necesitas ajustar el primer retardo en el código. Una cable más larto necesita más tiempo para calentarse. Esto significa que en vez de 4000 milisegundos, el retardo podría necesitar ser de 6000 milisegundos. Busca el valor que necesitas para tu configuración probando diferentes retardos. Sabrás

que tienes el retardo correcto cuando veas el músculo artificial contraerse totalmente a su forma más pequeña..

Como el músculo artificial se moverá, no es convenientes hacer una conexión eléctrica con hilo conductor ya que el cable mismo puede moverse. Otra razón para no usar hilo conductor es que es resistivo. Un hilo conductor puede facilmente ser más resistivo que el músculo artificial. Esto significa que si tu usas hilo conductor en lugar de cables normales, podrías no conseguir calentar el músculo artificial, porque estarías calentando también el hilo conductor. En general necesitas tener cuidado con el hilo conductor y con grandes corrientes ya que podrías prender fuego a tus prendas.

Tercera Parte:

Codificación

Capítulo 9: Escribiendo Programas

Un programa en Arduino está hecho de código, escrito en un lenguaje de programación llamado C. Cuando escribimos un código es importante tener en cuenta que Arduino no actúa como un ser humano, no razona. Si alguien pronuncia una palabra incorrecta probablemente entendamos el significado – Arduino sin embargo no. Si tecleas incorrectamente un comando mientras escribes código en el IDE de Arduino, no va a entender lo que quieres hacer.

Otra cosa a tener en cuenta es que Arduino es lógico pero no racional. No sabe que quieres hacer, sólo hace lo que le digas hacer.

Estructura Básica

Cuando escribimos código en el IDE de Arduino usamos tres partes básicas en la estructura del programa.

- declaración de variables
- setup()
- loop()

Las funciones setup() y loop() son esenciales escribirlas para el funcionamiento del programa. A medida que tengas más confianza en la escritura de código para Arduino te darás cuenta que no es obligatorio para escribir código usar estos pasos. Sin embargo viene bien estructurar el programa para hacer fácil su revisión y ayuda a buscar errores en el código. El siguiente código es un ejemplo de un programa sencillo que hace parpadear el LED de la placa Arduino:

```
int ledPin = 13;

void setup(){
  digitalWrite(ledPin,OUTPUT);
}
```

```
void loop(){
  digitalWrite(ledPin,HIGH);
  delay(1000);
  digitalWrite(ledPin,LOW);
  delay(1000);
}
```

Nota: Siempre es bueno poner tus variables en el principio del código.

Variables

Una variable es un contenedor para guardar algún dato. Digamos que quiere leer un sensor de algún tipo. Este sensor te dará un valor en formato numérico. Si quiere usar este valor en otra parte del programa puede guadar este valor en una variable. Antes de almacenar el valor necesita declarar la variable, esto significa que le digas al programa que la variable existe y que tipo de variable es.

Tenemos que poner un nombre a la variable. El nombre podría ser cualquiera pero es una idea buena dar a sus variables nombres que sean lógicos. Cuando revisé su programa será fácil determinar qué tipo de valor se almacena en una variable nombrada “temSensor”. Puede ser más difícil recordar cual es el contenido en variables con nombres como “banana”, “peter” o “supercalifragilisticexpialidocious”. Más explicaciones en la sección Tipos y Declaraciones de Variables en las páginas 80-82.

Void setup

Arduino lo primero que hace cuando arranca es mirar la void setup(). Éste es uno de los pasos esenciales del funcionamiento del programa. La void setup() es la parte que inicializa las configuraciones de los diferentes elementos del programa de Arduino; por ejemplo la configuración de un pin o establecer la velocidad de comunicación. La void setup() sólo se ejecuta una vez en el inicio y no se volverá a ejecutar hasta que Arduino se apagué y vuelva a arrancar o se resetea.

El siguiente código es un ejemplo de una void setup que pone la configuración de un pin como salida.

```
void setup(){
 pinMode(pin,OUTPUT);
}
```

Void loop

La void loop() es el segundo paso esencial para el funcionamiento de un programa. Aquí es donde se ejecutan las acciones de su programa. Como su nombre indica, ésta parte se ejecuta una y otra vez. En un programa de Arduino todo el código se ejecuta línea a línea. Después de ejecutar la void setup() en el arranque continua con la void loop().

Entonces empieza a hacer todo lo que está en su código desde el comienzo hasta el final de la void loop(). Cuando llega al final de la void loop() simplemente vuelve a empezar con los cambios del programa. El siguiente código es la void loop() del ejemplo del principio del capítulo:

```
void loop(){
 digitalWrite(ledPin,HIGH);
 delay(1000);
 digitalWrite(ledPin,LOW);
 delay(1000);
}
```

Esta void loop() empieza con una línea de código que activa el pin cuyo número se indica en el nombre de la variable “ledPin”. La siguiente línea de código produce un retardo y después desactiva el mismo pin y a continuación realiza un nuevo retardo. Si ejecutamos este código en Arduino con un LED conectado al pin con el número indicado en la variable “ledPin”, el LED parpadeará encendiéndose y apagándose con un segundo de retardo hasta apagar la alimentación.

Marcas de puntuación. Paréntesis y llaves

Las marcas de puntuación se utilizan para definir el inicio y el final de ciertas partes del código. Hay dos tipos de marcas usados en la escritura de código para Arduino. Los primeros son los paréntesis izquierdo y derecho () y son normalmente llamados entre paréntesis. Estos soportes se utilizan para escribir funciones

dentro de nuestros programas. Se utilizan para intercambio de una variable en algún otro lugar dentro de nuestro programa.

También es posible tener una función con los paréntesis vacíos, pero es necesario poner después el nombre de la función o Arduino tendrá un error de compilación. Un ejemplo de funciones que utilizan paréntesis vacíos son la void setup() y la void loop().

El segundo tipo son las llaves {}. Estas se usan para indicar el principio y final de una función. Sin estas llaves Arduino no será capaz de saber dónde empieza y termina la función y que se considera como la siguiente parte del código. Un uso común de estas llaves se hace en la función void setup().

```
void setup(){
 //The code in the function goes in here.
}
```

Punto y coma

Los puntos y comas son uno de los elementos más importantes para escribir código para Arduino y uno de los que más fácilmente nos olvidamos. Se usan para separar las diferentes líneas de código en su programa e indica a Arduino donde termina su comando. El siguiente ejemplo se muestra como se declara una variable con el uso correcto del punto y coma.

```
int myNumber = 15;
```

El punto y coma termina el comando y hemos declarado una variable entera con el nombre “myNumber” y esta variable tendrá el valor 15. Si te olvidas un punto y coma en tu código el IDE de Arduino resaltará la línea de código indicando la falta de punto y coma.

Comentarios de código

A veces puede ser útil para poner notas o escribir comentarios dentro de su código para uno mismo o para otra persona. Si se escribe el texto dentro de su programa Arduino piensa que es de código e intenta ejecutar lo que está escrito. Si lo que está escrito es algo que el Arduino no entiende, le dará un mensaje de error.

Nota:
'//' solamente sirve para los mensajes y notas que no sobrepasen una línea.

Hay dos maneras de escribir mensajes en el código y ocultarlo a Arduino. La primera de ellas es el uso de una doble barra / / delante de cualquier mensaje. Esto oculta el mensaje a Arduino pero deja lo visible para los programadores. El siguiente es un ejemplo de un mensaje oculto dentro de loop():

```
void loop(){
  digitalWrite(ledPin,HIGH); // enciende el led
  delay(1000); // espera un poco
  digitalWrite(ledPin,LOW); // apaga el led
  delay(1000); // espera un poco mas
}
```

Si desea ocultar los mensajes de más de una línea tiene que utilizar / * y * /. Para marcar el inicio de un mensaje que se oculta, se usa / * y para marcar el final del mensaje, use el / *. Esto ocultará todo el mensaje. El siguiente código es un ejemplo de cómo ocultar un bloque de texto dentro de loop():

```
void loop(){
  /* este codigo encendera primero un led, despues
     esperara un tiempo, tras lo que volvera a
     apagarlo y de nuevo a esperar. */
  digitalWrite(ledPin,HIGH);
  delay(1000);
  digitalWrite(ledPin,LOW);
  delay(1000);
}
```

Tipos de variables y declaraciones

Dar a una variable un valor es también conocido como declarar una variable. Declarar una variable es definir un tipo, nombre y valor para la misma.

```
int myNumber = 14;
```

En el ejemplo anterior el “int” es el tipo de variable, “miNumero” es el nombre y 14 es el valor. Ten en cuenta que siempre tienes que darle un valor a la variable cuando la declaras. Suponemos que quieres guardar un valor de tu sensor en tu programa, pero no puedes leer el valor cuando declaras la variable fuera del void loop(), en el inicio del programa. Tú le das un valor temporal 0 a la variable cuando la declaras al inicio, como en el siguiente ejemplo:

```
int mySensor = 0;
```

Hay dos posibles maneras de declarar una variable. Si la declaras al principio de tu programa, antes del void setup() diremos que es una variable global. Una variable global es accesible desde cualquier parte de tu programa. Por otro lado tenemos las variables locales, que sólo pueden usarse dentro de la función en la que se declararon. El siguiente ejemplo muestra una variable global llamada ledPin:

```
int ledPin = 13;

void setup(){
  digitalWrite(ledPin,OUTPUT);
}

void loop(){
  digitalWrite(ledPin,HIGH);
  delay(1000);
  digitalWrite(ledPin,LOW);
  delay(1000);
}
```

La variable ledPin es visible en todo el programa y cuando es usada dentro de void loop(); será reconocida como una variable de tipo entero (integer) con el valor 13. Sin embargo, si escribes el mismo programa de la siguiente manera:

```
void setup(){
  int ledPin = 13;
  digitalWrite(ledPin,OUTPUT);
```

```
}

void loop(){
  digitalWrite(ledPin,HIGH);
  delay(1000);
  digitalWrite(ledPin,LOW);
  delay(1000);
}
```

Entonces el programa te dará un error diciendote que no puede encontrar la variable ledPin que intentas usar dentro de void loop();, ya que está declarada dentro (y sólo dentro) de la función void setup();.

En algunos programas puede ser útil usar variables locales, pero en la mayoría de los casos es mejor declararlas como variables globales y, por tanto, declararlas antes del void setup();

Una vez que tienes una variable con un valor, tu puedes asignarle otro valor, pero ten en cuenta que borrarás el que existía previamente.Supongamos que tenemos una variable llamada miNumero y la hemos declarado como:

```
int myNumber = 14;
```

Si a lo largo de tu programa quieres darle otro valor a “myNumero”, lo harás así:

```
myNumber = 56;
```

Esto borrará el número 14 de la variable “myNumero” y lo reemplazará por el número 56. Cuando reasignamos valores las variables, no tenemos que añadir “int” al comienzo de la linea de código como hicimos cuando la declaramos. Ésto es porque nosotros sólo declaramos el tipo de variable una vez en el programa. Es posible cambiar el valor de la variable, pero no su tipo.

Tipos

Hasta ahora hemos estado hablando de “int”, que es el la abreviatura de número entero (el tipo más común de variables en programas de Arduino). Las variables comunes son:

• **Int**

Los enteros se usan para guardar datos numéricos sin puntos decimales. Almacenan un valor de 16 bits en el rango de 32767 a -32767. Esto quiere decir que un entero (int) ocupa 16 bits en la memoria de Arduino y puede ser cualquier número entero entre -32767 y 32767.

```
int myNumber = 1234;
```

Nota:
Usar muchas variables tipo ‘long’ puede llenar la memoria de Arduino úsalo solamente cuando sea necesario.

• **Long**

En muchos casos el tamaño de un entero nos valdrá pero, en otros, necesitaremos almacenar variables más grandes que el tamaño que nos permite el tipo “int”, para ello usaremos el tipo “long”. El tipo “long” extiende el rango de valores enteros (sin decimales) a un valor de 32 bits, esto es un rango de 2147483647 a – 2147483647. Igualmente esto quiere decir que un número “long” ocupará 32 bits en la memoria de Arduino y nos permitirá usar valores entre – 2147483647 y 2147483647.

```
long myBigNumber = 90000;
```

• **Byte**

Para ahorrar espacio en la memoria de Arduino, es útil almacenar las variables como bytes. Un byte es un número entero de 8 bits con un rango de 0 a 255. Con un byte ocuparemos 8 bits de la memoria de Arduino y podremos representar cualquier número entero entre 0 y 255.

```
byte mySmallNumber = 150;
```

• **Float**

El único tipo de datos que puede guardar números con decimales

es “float”. Float tiene mayor resolución que los enteros (int, long, byte) y se guardan como un valor de 32 bits en el rango desde 3.4028235E + 38 a – 3.4028235E + 38. Esto quiere decir que puedes guardar un número decimal pero sólo en ese rango. Los números decimales (float) ocupan mucha memoria de Arduino. Usar decimales es mucho más lento que usar enteros, pues Arduino necesita más tiempo para realizar cálculos con éstos.

```
float mydecimalNumber = 2.33;
```

• **Arrays (Matrices)**
A veces puede ser útil guardar una colección de valores, entonces tendremos que utilizar una matriz. Todos los valores almacenados en una matriz se guardarán con un número índice, para acceder a cualquier valor lo harás referenciando su número índice. Las matrices se declaran de igual modo que las variables (con el tipo, el nombre y los valores). El siguiente ejemplo muestra como declarar una matriz de enteros con seis valores distintos:

```
int myArray[] = {1, 2, 3, 4, 5, 6};
```

Ten en cuenta que las matrices empiezan a contar desde 0. Esto significa que la primera posición de la matriz es 0. En el ejemplo anterior, el número 1 está guardado en la primera posición de la matriz, por tanto, si queremos utilizarlo lo haremos del siguiente modo:

```
myNumber = myArray[0];
```

Esto almacenará el valor de la primera posición de la matriz (el valor 1), en nuestra variable miNumero. Por otro lado, podemos almacenar valores en la matriz haciendo referencia a la posición del número que nos interesa:

```
myArray[0] = 23;
```

Esto guardará el número 23 en la posición 0 de la matriz (donde

antes estaba el valor 1).

Si sabes que vas a usar una cantidad de números y los quieres almacenar en una matriz, pero no sabes qué valores van a ser, puedes declarar la variable reservando el número de posiciones que quieras, del siguiente modo:

```
int myArray[5];
```

Esto creará una matriz de 5 posiciones, donde 4 será la última (Recordemos que empieza a contar de 0). No intentes almacenar datos en la quinta posición, esta posición será incorrecta, pero Arduino no te avisará a la hora de comprobar el código.

Haciendo cálculos

Como Arduino es un pequeño ordenador, puede hacer operaciones matemáticas. Puede realizar las operaciones matemáticas más habituales como son, la suma, la resta, la multiplicación y la división.

```
myValue = 1 + 1;
 /* esto guardara el número 2 en myValue */
myValue = 4 - 2;
 /* esto guardara el número 2 en myValue */
myValue = 3 * 4;
 /* esto guardara el número 12 en myValue */
myValue = 6 / 2;
 /* esto guardara el número 3 en myValue */
```

Si estás usando enteros para realizar las operaciones matemáticas no podrás obtener decimales, “float” es el único tipo de variable que puede obtenerlos. En otras palabras, si quieres dividir 10 entre 6, dará 1 como resultado. Hacer cálculos demasiado largos puede producir un desbordamiento en la memoria de Arduino, ya que todo tipo de variable tiene un tamaño máximo. Realizar cálculos con números grandes ralentizará el Arduino.

Puedes realizar cálculos entre variables, a continuación tienes los diferentes cálculos que se pueden aplicar a las variables:

```
x++
 /* incrementara en 1 el valor de x. Es lo mismo que
        escribir x = x + 1 */
x--
 /* decrementara en 1 el valor de x. Es lo mismo que
        escribir x = x - 1 */
x += y
 /* incrementara el valor de y al valor de x. Es
        lo mismo que escribir x = x + y*/
x -= y
 /* decrementara el valor de y al valor de x. Es lo
        mismo que escribir x = x - y */
x *= y
 /* multiplicara x por y. Es lo mismo que escribir x = x
* y*/
x /= y
 /* dividira x entre y. Es lo mismo que escribir x = x /
y  */
```

• **Mapeo**
Supongamos que tienes un sensor que solo proporciona valores entre 50 y 200, y que tú necesitas un rango desde 0 a 500. La función “map” puede ser útil. Ésta función remapea un rango de valores a otro rango de valores. Digamos, hace una regla de 3:

```
myVariable = map(mySensor,50,200,0,500);
 /* valorSensor, valorMinSensor, valorMaxSensor,
        minimoDeseado, maximoDeseado */
```

En el ejemplo anterior estamos usando la función “map” para guardar un valor en miVariable. El valor proviene de miValor y el 50 y el 200 marcan los valores mínimo y máximo de nuestro sensor. El 0 y el 500 es el rango deseado. La función “map” asignará el valor obtenido (dentro del rango del sensor, 50-200) al rango deseado (0-500). Observe que si quieres usar la función “map” debes conocer el rango que quieres “mapear”. Para encontrar más información sobre cómo leer valores vaya a las páginas 98-101.

• **Random(max) (Aleatorio(máximo))**
El comando “random” devolverá un valor aleatorio en el rango comprendido entre 0 y el valor que pongas entre paréntesis. Para poder usar ese valor tienes que guardarlo en una variable.

```
myVariable = random(5);
```

Ésto guardará un número aleatorio en miVariable, dentro del rango de 0 a 4. También puedes usar el comando “random” directamente mientras haces comparaciones:

```
if (3 == random(5)){
  doSomething;
}
```

El comando “random” sólo devolverá un valor entre 0 y el máximo establecido, nunca devolverá ese máximo.

• **Random(min,max)**
Si quieres un número aleatorio comprendido en un rango que comience en un valor distinto de 0, debes especificar dicho mínimo:

```
myVariable = random(200,300)
```

Comparaciones lógicas

Si quieres realizar comparaciones en tus programas debes usar alguno de estos comparadores. Estas comparaciones se pueden realizar entre variables o con constantes, obteniendo siempre valores Verdadero o Falso.

Nota:
Este ejemplo solamente devolverá un valor en el rango entre 200 y 300. Nunca devolverá 200 ó 300.

• **Igual a ==**
== es usado para comparar si un elemento es igual a otro. Una comparación usando == sólo será verdadero si ambos miembros de la igualdad son idénticos:

```
x == y

   /* x es igual a y */
```

• **Distinto de !=**
!= != se usa para comprobar si un elemento es distinto de otro.

Una comparación usando ! = sólo será Verdadero si un miembro es distinto del otro:

```
x != y
 * x es distinto de y */
```

• **Menor que** <
< se usa para comparar si un elemento es menor que otro. Una comparación usando < sólo será Verdadero si el miembro izquierdo es menor que el derecho:

```
x < y
 /* x es menor que y */
```

• **Mayor que** >
> se usa para comparar si un elemento es mayor que otro. Una comparación usando > sólo será verdadero si el miembro izquierdo es mayor que el derecho:

```
x > y
 /* x es mayor que y */
```

Nota:
Los símbolos 'mayor que' (>) y 'menor que' (<) pueden ser fácilmente confundidos. Recuerda siempre que la parte cerrada del símbolo apunta al valor más pequeño.

• **Menor o igual** < =
< = se usa para comparar si un elemento es menor o igual que otro. Una comparación usando < = sólo será verdadera si el miembro izquierdo es igual o menor que el derecho:

```
x <= y
 /* x menor o igual que y */
```

• **Mayor o igual** > =
> = se usa para comparar si un elemento es mayor o igual que otro. Una comparación usando > = sólo será verdadera si el elemento izquierdo es mayor o igual que el derecho:

```
x >= y
 /*x es mayor o igual que y*/
```

Operadores Lógicos

Los operadores lógicos se usan cuando necesitas dos o más elementos en la misma sentencia, y éstos pueden ser verdaderos o falsos. Se pueden utilizar tros comparadores lógicos.

• **Y &&**

Se usa para determinar si dos o más elementos son verdaderos. SI alguno de los elementos no es verdadero, la sentencia será falsa. Para que algo sea verdadero, todos los elementos deben serlo, para que sea falso, con que uno sea falso, la sentencia lo será:

```
x < y && y > 5
/* x is smaller than y and y is bigger than 5 */
```

• **O ||**

Se usa para determinar si alguno de los dos elementos es verdadero. Con que alguno de los elementos sea verdadero, la sentencia será verdadera:

```
x < y || y > 5
/* x es menor que y, o y es mayor que 5. Nota: Los dos
elementos pueden ser
     verdaderos */
```

• **Negacion !**

Se usa para determinar si algo no es verdad. Si no se cumple, la sentencia será verdadera:

```
!x==5
 /* x no es igual a 5 */
```

Constantes

Las constantes son palabras que Arduino utiliza y que tienen valores predefinidos. Se usan para simplificar la lectura del codigo de tu programa.

• **True and False (verdadero y falso)**

True y False son lo que llamamos constantes Booleanas y definen

si algo lo es, o no, a nivel lógico:

```
boolean myBoolean = true;
```

Cualquier número puede usarse como operador Booleano. Por ejemplo, 200 se puede usar como operador, y si una variable tiene ése valor y lo comparamos con 200, ésto nos devolverá Verdadero:

```
int myNbrBoolean = 200;
/* Asignamos el valor a la variable*/
myNbrBoolean == 200;
  /* Comparamos myNbrBoolean con 200, devolvera Verdadero
*/
```

También se puede escribir con un número

```
int myNbrBoolean = 1;
```

En el primer ejemplo necesitamos comparar miBooleano con otro Booleano, y en el segundo caso comparamos miOtroBooleano con otro entero.

• **High and Low (Alto y Bajo)**
HIGH y LOW se usan para determinar el estado de un pin digital, que sólo tiene esos dos estados. HIGH quiere decir lo mismo que ON (o que hay 5 voltios en tu pin digital). Es lo mismo que un 1 lógico. LOW quiere decir lo mismo que OFF (o que hay 0 voltios en tu pin). Es lo mismo que un 0 lógico:

Nota:
Cuando estés programando todas las constantes Arduino siempre se escriben con letras mayúsculas.

```
digitalWrite(ledPin,HIGH);
```

Esto se puede escribir también con números:

```
digitalWrite(ledPin,1);
```

• **Input and Output (Entrada y Salida)**
INPUT y OUTPUT se usan cuando declaramos el modo de funcionamiento de nuestro pin digital, sólo existen esos dos modos:

```
pinMode(12,OUTPUT);
```

Si ocurre algo y qué hacer

Supongamos que estás trabajando en un prototipo y estás midiendo una distancia. Cuando algo se acerque a una distancia del objeto, quieres que ocurra algo. Aquí es cuando la sentencia If es útil.

• **If (Si)**

Una sentencia If es como un test que Arduino puede hacer para determinar si algo es verdadero o falso. Una sentencia If sigue esta estructura:

```
if (myVariable>myOtherVariable){
 doSomething;
}
```

En este ejemplo preguntamos si miVariable es mayor que miOtraVariable. SI es así, el programa saltará dentro de la función If y ejecutará el código (hazAlgo). Si la comparación resulta ser falsa, el programa saltará esta parte de código. En este ejemplo hemos comparado variables, pero también podríamos comparar constantes:

```
if (buttonPin==HIGH){
 doSomething;
}
```

En este ejemplo preguntamos si botonPin está HIGH, en cuyo caso leeremos el código hacerAlgo, de lo contrario omitirá el código de la función.

No olvides usar = = cuando realices comparaciones. Si usas sólo un = no compararás un elemento con otro, sino que asignarás el valor del segundo elemento al primero.

```
if (buttonPin=HIGH){
 doSomething;
}
```

Nota:
'if' siempre se escribe con minúsculas al escribir código. No te olvides de las llaves para marcar el inicio y el final de la función 'if'.

El ejemplo anterior muestra la manera incorrecta de escribir una sentencia If. En este ejemplo estamos definiendo botonPin como HIGH en lugar de comprobar si botonPin está en HIGH. El compilador de Arduino no te avisará de este error.

• If else (Si .. si no ..)

Ahora supongamos que quieres hacer una comprobación, y que sabes qué hacer tanto si se da la condición, como si no. Lo que puedes hacer es conectar una sentencia Else (si no ...) a tu sentencia If:

```
if (myVariable>myOtherVariabel){
 doSomething;
 } else {
 doAnotherThing;
}
```

El ejemplo anterior trabaja del siguiente modo: Si miVariable es mayor que miOtraVariable, entonces realiza hazAlgo. Si miVariable no es mayor que miOtraVariable, entonces realiza hazOtraCosa.

No olvides usar otro par de llaves ({}) para delimitar el comienzo y final de la sentencia Else.

Puedes añadir tantas condiciones Else como quieras dentro de una sentencia If, pero si quieres añadir más de una, tienes que escribir todas ellas, menos la última, con un Else if seguido de una nueva condición:

```
if (myVariable>myOtherVariable){
 doSomething;
 } else if (myVariable<100){
 doAnotherThing;
 } else {
 doTheLastThing;
}
```

En el ejemplo anterior, si miVariable no es mayor que miOtraVariable, entonces pregunta si miVariable es menor que 100 y, si no es cierto, entonces realiza hazUnaUltimaCosa.

• **For (Durante)**
Los bucles Fot se usan cuando quieres repetir una parte del código un número concreto de veces. El bucle For siempre tiene tres parametros entre paréntesis, éstos son: la inicialización del contador, la condición para terminar el bucle y el incremento de tu contador.:

```
for (int i=0; i<200; i++){
 doSomething;
}
```

En este ejemplo “int i = 0;” es la inicialización del contador para el bulce For. Aquí decimos que queremos un contador con el nombre “i” y de tipo “int”, y que queremos empezar a contar desde 0. Cuando hayamos terminado de definirlo, terminamos con un punto y coma, y pasamos a la segunda parte. Definimos la condición, si i < 200 terminaremos el bucle. La última parte define nuestro incremento por cada pasada en el bucle. i + + incrementará el contador en una unidad por cada pasada. La primera vez, el contador estará a 0, la siguiente estará a 1 y así sucesivamente. Cuando i alcance el valor 200, la condición i < 200 ya no se cumplirá, por lo que saldremos del bucle y continuaremos con el código escrito después de la llave de cierre del bucle (}).

• **While (Mientras...)**
Un bucle While durará hasta que la condición entre paréntesis sea falsa. Si usas un bucle While, debes asegurarte que la condición debe poder variar de algún modo (incrementandose, mediante comparaciones..) de no ser así el bucle no terminaría nunca:

```
while (myVariable<100){
 doSomething;
}
```

Este ejemplo comprueba si miVariable es menor que 100. Si es así, entonces comienza el bucle, pero si dentro de mi bucle no hay nada que haga cambiar el valor de miVariable, el bucle no terminará nunca.

En el siguiente ejemplo hemos añadido la lectura de un sensor:

```
while (myVariable<100){
  doSomething;
  myVariable = readSensor;
}
```

Ahora cada vez que se se repita el bucle, primero realizará hazAlgo, y después guardará en miVariable, el valor obtenido en la lectura del sensor. Si el valor almacenado en miVariable es superior a 100, el bucle While se detendrá y continuará leyendo el código escrito despues de la llave de cierre. Asegurate de, si estás usando un sensor para salir del bucle, éste pueda proporcionarte un valor que cumpla tu condición del bucle.

Los pines digitales

Estos pines son los pines del 0 al 13 de tu Arduino y se llaman digitales porque sólo pueden manejar valores 0 o 1. Si quieres usar un pin digital, lo primero que tienes que hacer es configurar el modo de trabajo del pin. Ésto se hace siempre en el void setup().

El comando para configurar el modo de trabajo es pinMode() y se usa del siguiente modo:

```
pinMode(pin,OUTPUT);
```

En este ejemplo “pin” es una variable con el valor correspondiente al número del pin a utilizar. OUTPUT es el modo de trabajo que queremos definir. Un pin digital tiene sólo dos modos, OUTPUT (salida) e INPUT (entrada). Si declaras un pin como OUTPUT, sólo podrás usarlo para activarlo, aplicando 5V en el pin, o para desactivarlo, aplicando 0V en el pin. Si configuras el pin como INPUT, sólo podrás usarlo para leer si hay 5V o 0V en el pin:

```
digitalWrite(pin,valor);
```

Para encender o apagar tu pin digital debes usar el comando digitalWrite(). Entre paréntesis debes indicar qué pin modificar, y qué valor darle:

```
digitalWrite(pin,HIGH);
```

Ésto pondrá el pin en su estado HIGH, proporcionando 5V en él. Si escribes LOW en lugar de HIGH apagarás el pin, volviendolo a dejar en 0V.Ten en cuenta que hasta que definas el estado del pin como HIGH su valor por defecto será LOW. Si miras tu placa Arduino, verás que los pines digitales 0 y 1 están marcados como RX y TX. Estos pines están reservados para la comunicación serie y no deben ser usados, ya que pondrán a Arduino en modo de espera hasta que se reciba una señal.

Nota:
Si estás usando un pin como entrada asegúrate de que la señal nunca es mayor de 5V o quemarás la placa Arduino.

• **DigitalRead(pin) (Lectura de pin digital)**
El comando digitalRead() lee el estado de un pin y devuelve HIGH si está a 5V o LOW si hay 0V en él:

```
digitalRead(pin);
```

Para poder usar el valor del estado para algún fin debes guardarlo en una variable:

```
myVariable = digitalRead(pin);
```

Si quieres realizar una comparación puedes escribir el comando directamente en la sentencia:

```
if (digitalRead(pin)==LOW){
 doSomething;
}
```

Aunque LOW equivale siempre a 0V en una salida digital, en una entrada digital cualquier valor entre 0V y 1.5V se considerará LOW en el comando digitalRead(). Del mismo modo todos los valores entre 3.3V y 5v se considerarán como un valor HIGH.

Los pines analógicos

Los pines analógicos y digitales funcionan de diferente manera. Hemos mencionado que los pines digitales sólo manejan información en 1 y 0, lo que es lo mismo como ALTO (HIGH) y BAJO (LOW) o 0V y 5V. Sin embargo, en el mundo real no medimos sólo ceros y unos así Arduino tiene seis pines especiales analógicos que hacen un cálculo del voltaje en un rango codificado de 0 a 1023. Los pines analógicos no deben ser declarados, su configuración de I/O, ya que sólo son utilizados como entradas.

• **Analog read (pin)**

Para leer el valor de un pin analógico tienes que usar el comando analogRead() y poner la referencia del pin que deseas leer:

```
analogRead(pin);
```

Nota:
Si estás haciendo comparaciones con lecturas analógicas, el valor del pin analógico no puede ser mayor de 1023.

Al igual que con el pin digital, tienes que guardar este valor en una variable para poder usarlo

```
myVariable = analogRead(pin);
```

Puedes usar el comando directamentepara hacer comparaciones

```
if (analogRead(pin)>500){
 doSomething;
}
```

• **Analog write (pin,value)**

Los pines digitales sólo puede ser HIGH y LOW, que es lo mismo que tener 5V o 0V en los pines digitales. Sin embargo, para los pines digitales 3, 5, 6, 9, 10 y 11 tenemos una función especial llamada analogWrite (). Con esta función es posible enviar un valor pseudo-analógico a estos pines digitales especiales. Esto se denomina modulación por anchura de pulso (PWM, pulse with modulation):

```
analogWrite(pin,value);
```

El valor de este ejemplo puede ser desde 0 a 255. Si escribes un 0, esto sería lo mismo que el poner en el pin LOW, y 255 es lo mismo que HIGH. Pero con analogWrite() obtienes 255 pasos entre HIGH y LOW, así por ejemplo:

```
analogWrite(pin,127);
```

Esto sería similar a enviar 2.5V al pin digital. En comparación con el digitalWrite(), que pasa de 0 V a 5 V en un instante, con la analogWrite() puede hacer una transición más lenta de 0V a 5V. Tenga en cuenta que la analogWrite() sólo funciona en los pines digitales marcadas con PWM (3, 5, 6, 9, 10 y 11) y no en los pines de entrada analógica de a0 a a5.

Usando el tiempo

Arduino es una pequeña computadora, pero poderoso y puede realizar 1.000.000 operaciones por segundo. Cuando vaya a hacer prototipos que no quiera ejecutar a una velocidad tan rápida. Vas a tener que decirle a Arduino que pare de vez en cuando.

• **Delay (retardo)**

El comando de retardo se utiliza para establecer una pausa en su programa. Esto cuenta de comando en milisegundos y se introduce el tiempo deseado de pausa dentro de los paréntesis, como en el ejemplo siguiente:

```
delay(1000);
```

Este delay pondrá una pausa en su programa de un segundo,

• **Contar milisegundos**

Este comando millis() devolverá cuantos milisegundos han pasado desde que Arduino inició la ejecución del programa. Para poder utilizar este valor hay que guardarlo en una variable:

```
myVariable = millis();
```

Puede usarlo directamente para hacer comparaciones de tiempo:

```
if (myAlarmTime == millis()){
 ringAlarm;
}
```

Nota:
El valor de la función millis() se restablecerá a 0 después de cerca de nueve horas.

Comunicación con otros dispositivos

Para ser capaz de comunicarse con otros dispositivos electrónicos debe habilitar los puertos de comunicación de Arduino. Arduino puede comunicarse con el ordenador y con otros dispositivos electrónicos que utilizan el protocolo de comunicación serie.

Los pines digitales 0 y 1 de Arduino se reservan para comunicaciones en serie con otros dispositivos y debería evitar usar estos dos pines para otras tareas, ya que puede interferir con el funcionamiento de su programa.

• **Serial begin**
Para permitir la comunicación de Arduino se utiliza el comando Serial.begin(). Este comando se utiliza sólo en el void setup(). Dentro de los paréntesis se introduce la velocidad de comunicación deseada en bits por segundo, lo que también se conoce como baudios, y las velocidades disponibles son 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200:

```
void setup(){
 Serial.begin(9600);
}
```

Este código abrirá el puerto serie y pone la velocidad de comunicación a 9600 baudios.

• **Serial println**

Este comando va a imprimir lo que pones dentro de los paréntesis y añadir un final de línea. Para imprimir números enteros tiene que poner el tipo dentro de los paréntesis:

```
Serial.println(12345);
```

Sin embargo, los caracteres de impresión y cadena de caracteres debe ser citado. Si desea imprimir un solo carácter usa las comillas simples ‘ ‘:

```
Serial.println('C');
```

La línea de código anterior enviará el carácter C a través del puerto serie. Si desea imprimir una cadena de caracteres, como mensaje tiene que utilizar las dobles comillas “ “

```
Serial.println("Hello from Arduino");
```

• **Serial print**
El Serial.print() trabaja como el Serial.println() con la excepción que Serial.println() inserta un retorno de carro y salta a la siguiente línea. Si algo se envia por el puerto serie con Serial. println(1) el mensaje se muestra en el monitor de la siguiente manera:

```
1
1
1
1
1
```

El avance de línea añadido por el Serial.println() es lo mismo que poner al mismo tiempo un final de linea y empezar en una nueva. Si queremos enviar Serial.print(1) través del puerto serie se recibiría como:

```
11111111111111111111111111111111111111
```

Tenga en cuenta que si está comunicandose con otros dispositivos electrónicos asegúrese de imprimir los datos en el formato correcto. A veces, el retorno de carro y salto de línea añadido por Serial.println() puede interferir con la comunicación. La información sobre el protocolo de comunicación adecuado para los distintos dispositivos electrónicos se pueden encontrar en la hoja de datos del dispositivo a comunicar.

• **Casos especiales de impresión**

A veces será necesario enviar la información en diferentes formatos y entonces tiene que añadir este formato para su impresión:

Nota:
Una 'b' minúscula, que se utiliza en estos ejemplos, tiene el valor ASCII "79".

```
Serial.println(message,format);
```

Los formatos disponibles son decimales, hexadecimal, octal, binario y bytes, y se utilizan como en los ejemplos siguientes:

```
Serial.print(b,DEC);
 /* Escribe 79 como código ASCII en decimal que es "79"
*/
Serial.print(b,HEX);
 /* Escribe 79 como código ASCII en hexadecimal que es
"4F" */
Serial.print(b,OCT);
 /* Escribe 79 como código ASCII en octal que es "117"
*/
Serial.print(b,BIN);
/* Escribe 79 como código ASCII en binario que es
"1001111" */
Serial.print(b,BYTE);
 /* Escribe 79 como código ASCII en bytes que es "O" */
```

El ASCII (American Standard Code for Information Interchange) de codificación es una forma estándar de codificación de texto numérico. Cuando se escribe el número en su código, no está en realidad escribiendo el número, pero si el carácter representado los números, como "uno" es la palabra que representa 1.

Epílogo

Ahora hemos alcanzado el fin de este libro y con suerte el inicio de tu propio futuro en prototipos de tipo wearable y de moda. Esperamos que esto te haya dado un profundo entendimiento dentro del mundo de la wearable computing y que los ejemplos en el libro servirán como base para tu propio progreso en el campo. Sin embargo, podría ser difícil una transición directa de este libro a realizar tus propios prototipos. Pero por fortuna, no estás solo en en el mundo de la creación de prototipos electrónicos. Como dijimos al principio de este libro, Arduino no solo es hardware y software. También es una amplia comunidad de personas, profesionales y aficionados con un interés en la creación de prototipos electrónicos.

Una gran parte de estas personas se reúnen en en el Arduino playground (www.arduino.cc/playground) donde de igual forma se ayudan uno al otro con todo tipo de problemas o comparten ideas y exhiben nuevas construcciones.

HOW TO GET WHAT YOU WANT (www.kobakant.at/DIY) por Mika Satomi y Hannah Perner-Wilson es el mejor recurso en línea para técnicas de tecnología wearable. En este sitio puedes encontrar ejemplos de proyectos e información de soft circuits, tejidos conductores y resistivos, hilos conductivos y consejos sobre herramientas.

Instructables (www.instructables.com) es otra gran comunidad en línea para la gente DIY (do it yourself. hazlo tu mismo). Los usuarios no solo crean sus propias guías electrónicas DIY, también “como hacer” casi cualquier cosa que puedas pensar. Instructables es una de las mejores y más grandes fuentes de inspiración y sinceramente podemos recomendarla a cualquiera interesado en la wearable computing y la moda.

Otra buena contendiente a Instructables es la comunidad y blog Make (www.makezine.com) que también tiene un gran grupo de

gente llena de cualidades, fieles a la comunidad, y es un recurso que te mantiene actualizado sobre que está sucediendo en el campo de la creación de prototipos.

Si estás buscando por comunidades estrictamente dedicadas al campo de la moda y la tecnología, entonces el blog Fashioning Technology (www.fashioningtech.com) es un agradable lugar para iniciar. Ellos también tienen una comunidad creciente de usuarios con un foro donde comparten ideas y se ayudan mutuamente.

Aunque un libro sirve como un buen comienzo e introducción, nunca superará al Internet por su información extensa y actualizada. Nuestra esperanza es que este libro te proporcionará el conocimiento necesario para hacer uso de toda la información que puedas encontrar en línea.

Existen más buenos recursos en línea e incluirlos a todos en este libro probablemente requeriría unos cuantos capítulos más. Los ejemplos anteriores todos son buenos puntos de partida para tu propia biblioteca de información en línea.

La cosa más importante que debes mantener en mente al trabajar con prototipos electrónicos es divertirte y no tener miedo de probar cosas. Pero ten en cuenta que estás trabajando con electricidad y si no estás seguro en algo, sigue el principio sencillo de hacer una búsqueda apropiada hasta que estés seguro y conecta las cosas, podría ahorrarte dinero.

Ten seguridad en lo que haces, no comentas estupideces y ten una feliz creación de prototipos

Agradecimientos

Este libro fue inspirado por el pequeño libro Arduino programming notebook de Brian Evans, 1ra. edición, 2007 y Getting Started with Arduino, 3ra. edición, 2008 por Massimo Banzi.

Agradecemos también al laboratorio de prototipos físicos en la Universidad de Malmö, Facultad de Arte, Cultura y Comunicación (k3), los chicos en el estudio de diseño crítico 1scale1, Malmö y un agradecimiento especial al equipo de Arduino por su trabajo con la plataforma Arduino: Tom Igoe, Dave Mellis, David Cuartielles, Massimo Banzi y Gianluca Martino.

OPEN SOFTWEARTM - 2ª Edición por Tony Olsson, David Gaetano, Jonas Odhner y Samson Wiklund.

Copyright © 2011, 2009 Tony Olsson, David Gaetano, Jonas Odhner y Samson Wiklund.

Open Softwear: Prototipado electrónico textil y computación "wearable" usando Arduino, está publicado bajo Licencia Creative Commons Reconocimiento-NoComercial-SinObraDerivada 3.0 Unported (CC BY-NC-ND 3.0) (http://creativecommons.org/licenses/by-nc-nd/3.0/#)

Editor: Tony Olsson. Editor de Producción: Jonas Odhner. Traducción: ? Correctores: ?

Diseño Gráfico: David Gaetano y Jonas Odhner. Ilustración: David Gaetano y Jonas Odhner.

Tipografía: Charis SIL (Cabeceras y cuerpo del texto)
Titillium Text (Notas)
OCR A Std (Ejemplos de código)

Papel: Edixion Offset 120g. Impreso en: JMS Mediasystem, Vellinge, Suecia, 2011.

Historia: Primera Edición, Julio 2009 (Publicación online en PDF).
Segunda Edición, Abril 2011.

ISBN: 978-91-979554-0-9

Web: www.softwear.cc

Publicado por Blushing Boy Publishing. www.blushingboy.com

Aunque se han tomado todas las precauciones posibles para la elaboración de este libro, el editor y los autores no asumen ninguna responsabilidad por los errores o omisiones, o por los daños ocasionados por el uso de la información que en él contiene.