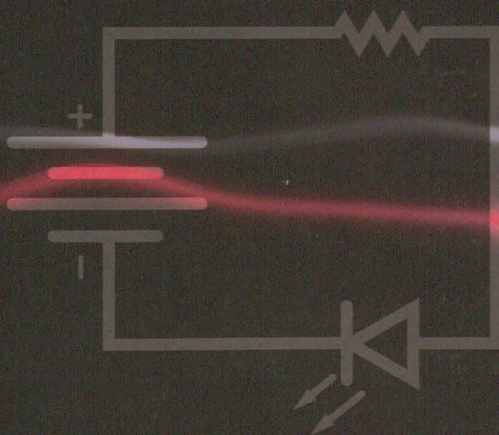


Make:



Wearable Electronics

**Design, prototype, and wear your
own interactive garments**

Kate Hartman

Make: Wearable Electronics

Kate Hartman



Make: Wearable Electronics

by Kate Hartman

Copyright © 2014 Kate Hartman. All rights reserved.

Published by Maker Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Maker Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreil.ly.com>). For more information, contact O'Reilly Media's corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editors: Brian Jepson and Emma Dvorak

Production Editor: Kara Ebrahim

Copyeditor: Jasmine Kwityn

Proofreader: Amanda Kersey

Indexer: Ellen Troutman

Cover Fashion Designer: Angella Mackey

Interior Designer: David Futato

Illustrator: Rebecca Demarest

Cover Photographer: Henrik Bengtsson

Cover Model: Jenny Andresson

Technical Editors: Rob Faludi, Erin Lewis, Pearl Chen, and Lynne Bruning

Custom Illustrator: Jen Liu

Research Assistant: Hillary Predko

August 2014: First Edition

Revision History for the First Edition:

2014-08-05: First release

2014-09-22: Second release

2015-10-16: Third release

2018-08-03: Fourth release

2021-09-03: Fifth release

See <http://oreil.ly/9781449336516> for release details.

The Make logo and Maker Media logo are registered trademarks of Maker Media, Inc. [Make: Wearable Electronics](#) and related trade dress are trademarks of Maker Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Maker Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-33651-6

[LSI]

Making Electronics Wearable

5

There are many challenges you don't anticipate in designing wearable electronics, which is why it is so important to actually wear your prototypes—early and often. In this chapter, I explore what makes something wearable and how to incorporate that into your own designs.

Why Wear It

When embarking on a wearable electronics project, the first and most important question you should ask yourself is this:

"Why does this need to be wearable?"

There are many possible answers to this question, but it's important that you have at least one in mind. Here are a few:

- You are sensing the body in such a way that the sensor needs to be placed on the body.
- You have a display or feedback mechanism that needs to stay with a person at all times.
- Your project needs to travel with the user and not stay in one place.

- You want to create a particularly intimate or immersive experience for the wearer.
- Your project is specifically clothing-oriented, such as a costume or fashion piece.
- You're interested in the future of wearable electronics and want to use making as a way to think about what's next.

Once you have a reason in mind, you can use it to guide the decisions that follow as you design your project.

What Makes Something Wearable

There are many factors to consider when converting an electronic circuit to a wearable form. What works on a breadboard or in a project box doesn't always translate so well to the dynamic, unpredictable, and rugged context of the human body.

Sensors

7

Simply stated, a sensor is an electronic component that measures some aspect of the physical world and converts that measurement into varying electrical characteristics, namely voltage or resistance. Sensors can sense things like light, movement, temperature, and touch. They are exciting because they make the physical world perceivable by computers—even tiny computers like microcontrollers.

As you get deeper into the realm of interactivity, it's worth considering the wide range of sensors available to you. In this chapter, you'll take a moment to consider how you can best listen to what's happening in, on, and around the body through technological means. You'll encounter an assortment of body-centric sensors and look at some simple ways to work with the data they produce.

Working with Sensors

There are both conceptual and technical factors to consider when working with sensors.

Getting to Know Your Sensor

Looking at a sensor that is new to you can be intimidating, exciting, or overwhelming. It is easy to look at the name of a newly released sensor, think,

"That's exactly what I need!", order it, get it home, and realize that it is incompatible with your project.

Just like in any good relationship, it's worth sniffing out your prospective sensor before making the big commitment. You can always, at the very least, have a virtual introduction to it through data-sheets, product descriptions, reviews, forums, and tutorials. And if you're part of a hacker, maker, or educational community, you may very well know someone who has the sensor you're considering. Ask if you can borrow it and take it out for a spin before taking the plunge and getting one of your own. While electronic sensors are significantly cheaper than they used to be, it's still an investment, so it's worth doing your research.

When encountering a sensor for the first time, whether it be online, in a catalog, or in real life, here is the type of information you should be looking for:

Connector type

Depending on the manufacturer, breakout board, and intended use, sensors will have different connection types (Figure 7-1). If there are headers, are they male or female? And do they use breadboard spacing or something else? Is it a standardized plug and socket set such as JST? Or a proprietary connector that is

specific to the manufacturer? Does it have sew tabs because it is intended for use with conductive thread? These factors will likely determine what other materials you will need to build your circuit.



Figure 7-1. Sensors with different connectors; left to right: legs, terminals, JST connector, pins, hook, male headers

Sensitivity

What is the range of values your sensor senses? For instance, smaller force-sensing resistors (FSRs) can sense as little as 2 grams of force ([Figure 7-2](#)), whereas larger ones have a sensing range of 100 grams to 10 kilograms ([Figure 7-3](#)).

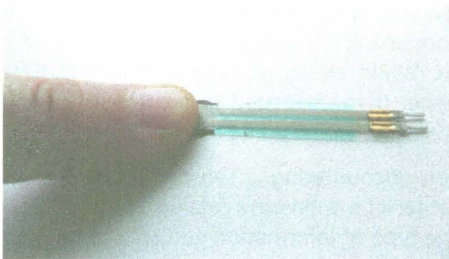


Figure 7-2. This smaller force-sensing resistor is sensitive to even the lightest touch of a finger

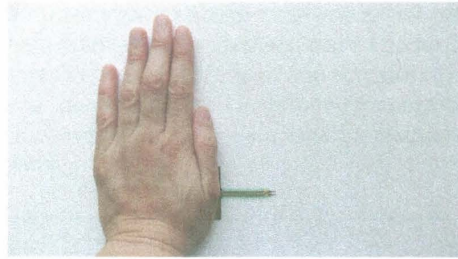


Figure 7-3. This larger force-sensing resistor is sensitive to a higher range of pressure, making it appropriate to be pressed firmly, leaned on, or stepped on

Accuracy

How accurate is your sensor? FSRs are very sensitive, but not always accurate. Depending on their range of sensitivity, they provide good answers to questions like, "Is this being pressed or not?" or, "Is this being stood on or not?" but they do not provide the level of accuracy needed if you were building a precise food scale or a body scale.

Shape, size, and weight

What are the physical characteristics of your sensor (see [Figure 7-4](#))? These greatly affect how a component can be worn on the body. Be sure to look at dimensions and technical drawings to determine if a sensor will fit in with your design.

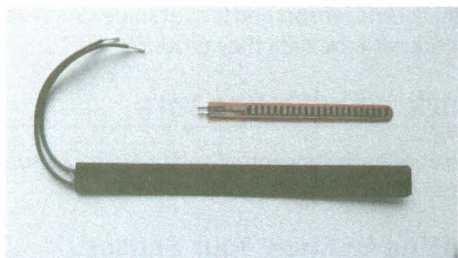


Figure 7-4. Flex sensors come in different lengths; a shorter flex sensor might be more appropriate for a project with limited surface area

Sensor output

What kind of information does your sensor provide and how? The sensor's output is what gets read by the microcontroller (Figure 7-5). You've learned so far that a varying voltage output can be read by the analog input pins on the Arduino. Later in this chapter, you will also learn how to read varying resistance and how a sensor transmits data via serial communication.

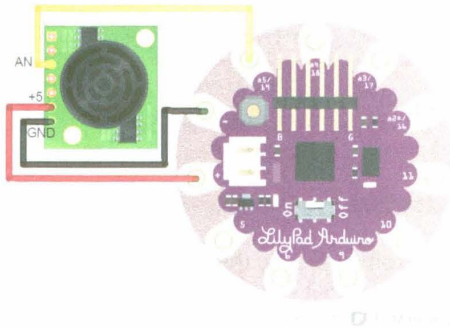


Figure 7-5. Maxbotix Ultrasonic sensors feature multiple outputs, including analog voltage, serial communication, and pulse width modulation (digital pulses)

Voltage Divider Circuit

Some of the sensors you work with are *variable resistors*. A variable resistor is a component that changes resistance in response to a changing condition. Variable resistors that you will encounter later in this chapter include flex sensors, stretch sensors, and light sensors.

The problem with trying to read a variable resistor with a microcontroller is that a microcontroller's analog input reads varying *voltage*, not varying resistance. Luckily, there is a simple circuit that enables a variable resistor to produce varying voltage: a *voltage divider* circuit (shown in Figure 7-6).

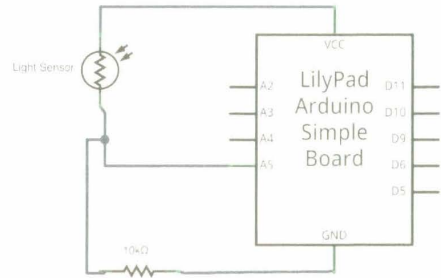


Figure 7-6. Voltage divider circuit diagram with variable resistor and fixed 10KΩ resistor

This circuit pairs a variable resistor connected from power to the analog input with a fixed resistor connected from ground to the same analog input. The fluctuating ratio of their resistances creates a varying voltage between them. For your purposes, the fixed resistor just needs to be in the same order of magnitude as the variable resistor. Many of the variable resistors you encounter in this chapter will work just fine with a fixed 10KΩ resistor.

You've actually already worked with a variable resistor in Chapter 6: the LilyPad Light Sensor. If you look closely in Figure 7-7, you can see the surface mount fixed resistor and the traces going to the pads.



Figure 7-7. Closeup of LilyPad light sensor: note the resistor on the right as well as how the circuit board traces create a voltage divider circuit

Communicating with I2C

All of the LilyPad sensors provide a varying voltage which can be read by an analog input pin on the Arduino. But Flora sensors communicate sensor values to the microcontroller through different means: I2C. While this book won't provide a comprehensive orientation to I2C, it's helpful for you to know at least some of the basic details.

Short for Inter-Integrated Circuit and referred to as “I-squared-C” or “I-two-C”, I2C is a two-wire serial communication protocol.

The following connections are made between any I2C sensor and the microcontroller (shown in Figure 7-8):

SCL	Serial clock pin—pulses on this pin provide the timing for the communication
SDA	Serial data pin—the wire on which the actual data is sent and received
gnd	Share a common ground with the microcontroller
3V	Flora sensors require 3V power, but check the datasheet for other I2C sensors

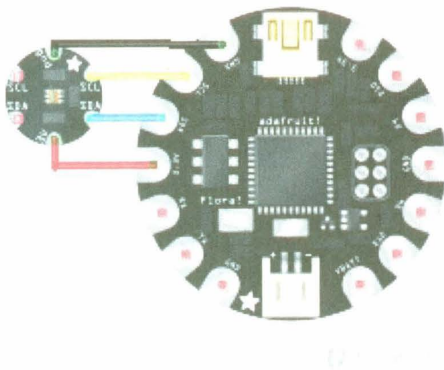


Figure 7-8. Flora sensor connections

On the Flora, SCL and SDA are clearly marked. On the Arduino Uno and the LilyPad Arduino, SDA is A4 and SCL is A5. For other boards, check their datasheets.

A neat aspect of working with I2C devices is that they can be chained (Figure 7-9) so that they don't take up lots of pins on the Flora. This also greatly reduces the amount of wiring. I2C sensors (and other devices) usually have a predetermined address. The address is used by the microcontroller to speak to a particular device in the chain of I2C devices. This information can be found in the device's datasheet.

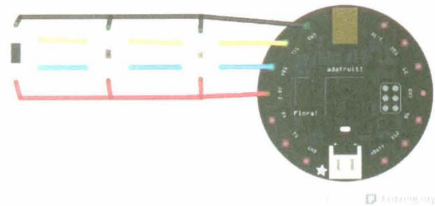


Figure 7-9. Circuit layout of Flora sensor chain

On the software side, there is an Arduino library called “Wire” that enables Arduinos to communicate with I2C devices and handles the nitty-gritty details of this protocol so you don't have to. *Libraries* are additional packages of code that can be added in to support different functionalities or tasks. Some are included with your Arduino download (“standard libraries”) and some you need to download and install yourself (“contributed libraries”). Wire is a standard library. In addition, Adafruit provides very useful and sophisticated libraries for many of its sensors, which are contributed and typically user-installed.

While I2C is a significantly more advanced serial communication protocol, Adafruit provides excellent documentation for working with the Flora sensors, so it is fairly straightforward to get up and running quickly without having a comprehensive understanding of how the details of the I2C protocol work.

Working with Sensor Data

Once you're able to get access to the sensor data, you then need to figure out what to do with it. In this section, I explore some concepts for making sense of sensor data, including thresholds, mapping, calibration, constraining, and smoothing.

Let's use the light sensor circuit from the Sensitive System experiment in the previous chapter (Figure 6-32) for the following examples. Here are the parts you will need:

- LilyPad Arduino Simple (SF DEV-10274)
- LilyPad LED (SF DEV-10081)
- LilyPad Light Sensor (SF DEV-08464)
- FTDI Board (AF 284, SF DEV-10275)
- USB mini-B cable (AF 899, DK WM5163-ND, RS 55010682, SF CAB-11301)
- Alligator clip test leads (AF 1008, RS 278-1156, SF PRT-11037)

Thresholds

A *threshold* can be used to set a boundary between one condition and another. You can think of it like a border between two countries or a fence between two yards. Setting the boundary makes it easier to distinguish one thing from the other.

When working with a range of sensor values, sometimes it's helpful to indicate what different ranges within those values mean. Thresholds are a good way to get started.

Say you're working with a LilyPad light sensor. It's great to have a bunch of numbers flying by on the Serial Monitor, but how can you use them to make something happen? For instance, if you want an LED to turn on when it's dark and off when it's light, you need to define what "dark" is.

By looking at the values in the serial monitor and exposing the sensor to varying conditions (in this

instance, turning the lights on and off), you can select a *threshold* value. This is a number above which you would consider the condition to be "light" and below which to be "dark" (see Figure 7-10).

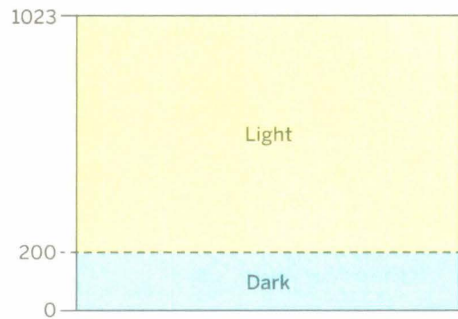


Figure 7-10. With a threshold set at 200, all values above are considered "light" and all below "dark"

You can implement a threshold in your code through the use of an if statement:

```
// if it is "dark"
if(lightSensorValue < 200){
  //Turn LED on
  digitalWrite(LEDpin, HIGH);
}
// if it is "light"
else{
  //Turn LED off
  digitalWrite(LEDpin, LOW);
}
```

This example will turn an LED connected to pin 11 on when it is dark and off when it is light. Here is the complete sketch:

```
/*
 * Make: Wearable Electronics
 * Single Threshold example
 */

//Initialize variables
int lightSensorValue = 0;
int lightSensorPin = A2;
int LEDpin = 11;
```

```

void setup() {
  //initialize serial communication:
  Serial.begin(9600);
}
void loop() {
  // read the light sensor pin and
  // store value in a variable:
  lightSensorValue = analogRead(A2);

  // if it is "dark"
  if(lightSensorValue < 200){
    //Turn LED on
    digitalWrite(LEDpin, HIGH);
  }
  // if it is "light"
  else{
    //Turn LED off
    digitalWrite(LEDpin, LOW);
  }

  // delay between readings:
  delay(100);
}

```

You can also have multiple thresholds (Figure 7-11) and use the `else if` conditional statement.

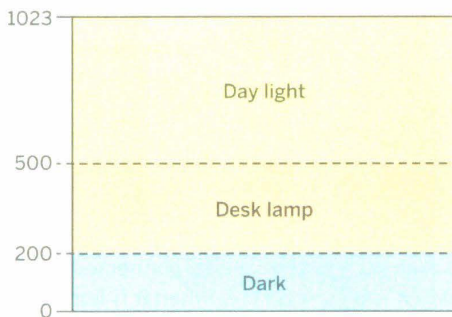


Figure 7-11 Multiple thresholds set at 200 and 500

The following sketch uses two thresholds and prints out a description of the light level as well as the raw sensor value in the Serial Monitor:

```

/*
  Make: Wearable Electronics

```

```

  Multiple Threshold example
  */

  //initialize variables
  int lightSensorValue = 0;
  int lightSensorPin = A2;
  int LEDpin = 11;
  int threshold1 = 500;
  int threshold2 = 200;

  void setup() {
    //initialize serial communication:
    Serial.begin(9600);
  }
  void loop() {
    // read the light sensor pin and
    // store value in a variable:
    lightSensorValue =
      analogRead(lightSensorPin);

    // print the light sensor value
    Serial.print("Light Sensor Value: ");
    Serial.print(lightSensorValue);

    // get ready to print light level
    Serial.print(", Light Level: ");

    //if the value is greater than
    // threshold #1
    if(lightSensorValue>threshold1){
      Serial.println("daylight");
    }
    //if the value is less or equal to
    // threshold #1 and greater than
    // threshold #2
    else if(lightSensorValue>threshold2){
      Serial.println("desk lamp");
    }
    //if the value is equal to or less than
    // threshold #2
    else{
      Serial.println("dark");
    }

    // delay between readings:
    delay(100);
  }
}

```

Give this code a try and modify the values so that they better match your environment.



Figure 7-12. “Capacity Indicator Bag” by Sally Chan uses an FSR and a 3-LED display to provide a visual indication of the weight of the bag’s contents

Mapping

Mapping is a way to translate a value from one range of numbers to another. It can be used to create a direct relationship between an input and an output. For instance, the value provided by a light sensor could control the *brightness* of an LED (as opposed to turning the LED on and off as you did earlier).

To accomplish this, there is a very useful Arduino function called `map()`. It looks like this:

```
map(value, fromLow, fromHigh,
    toLow, toHigh)
```

`value` is the value that you would like to map. `fromLow` and `fromHigh` is the low and high end of the original data set. `toLow` and `toHigh` are the low and high values of the mapped data set. If you were to map the full range of analog input (0 to 1023) to the full range of analog output (0 to 255), you’d use the following line of code (illustrated in [Figure 7-13](#)):

```
map(lightSensorValue, 0, 1023, 0, 255)
```

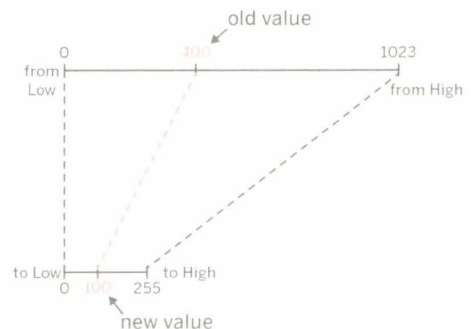


Figure 7-13 Mapping a value from 0-1023 to 0-255

A complete sketch looks something like this:

```
/*
 * Make: Wearable Electronics
 * Mapping example
 */

//initialize variables
int lightSensorValue = 0;
int lightSensorPin = A2;
int LEDpin = 1;
int mappedLightSensorValue = 0;

void setup() {
  //initialize serial communication:
  Serial.begin(9600);
}

void loop() {
  // read light sensor pin and
  // store value in a variable:
  lightSensorValue
```

```

analogRead(lightSensorPin);
//map sensor value
mappedLightSensorValue
  map(lightSensorValue, 0, 1023, 0, 255);
//set analog output accordingly
analogWrite(LEDpin, mappedLightSensorValue);

// print the sensor and mapped sensor values:
Serial.print( "Light Sensor Value: ");
Serial.print(lightSensorValue);
Serial.print( ", Mapped Light Sensor Value: ");
Serial.println(mappedLightSensorValue);

// delay between readings:
delay(100);
}

```

Many sensors don't have values that fully occupy the 0 to 1023 range. If you have a light sensor whose lowest value is 25 and highest is 940, you can change the *from* values accordingly (see Figure 7-14):

```
map(lightSensorValue, 25, 940, 0, 255)
```

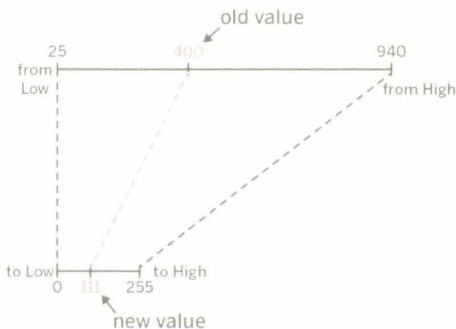


Figure 7-14. Mapping a value from 25–940 to 0–255

If you are working with a LilyPad light sensor as your input and using the mapped value to control an LED on the analog output pin, you would have an LED that brightens and dims in a way that mimics the conditions of the room. If you wanted to inverse the relationship so that the LED gets brighter as the

room gets darker, you can simply flip the *to* values, as illustrated in Figure 7-15:

```
map(lightSensorValue, 25, 940, 255, 0)
```

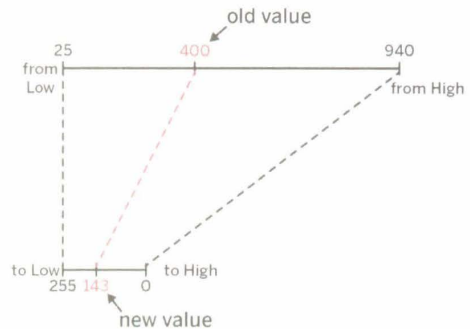


Figure 7-15. Mapping a value from 25–940 to 255–0; this will invert the relationship

See also:

- [Arduino map\(\) reference page \(http://arduino.cc/en/Reference/Map\)](http://arduino.cc/en/Reference/Map)
- [Arduino In, Out Serial example \(http://arduino.cc/en/Tutorial/AnalogInOutSerial\)](http://arduino.cc/en/Tutorial/AnalogInOutSerial)

Calibration

Calibration is a way to fine-tune your code so that it is responsive to a specific set of conditions. The range of what a sensor senses will differ based on its environment and context (Figure 7-16). The amount of light available in your bedroom or studio will differ greatly from that on the street or in a park. A force-sensing resistor will read different values when stepped on by a 5-year-old than a 50-year-old. If you know you'll be using your project in varying contexts, it's worth including a calibration routine in your code. You can determine the highest and lowest possible values and configure the rest of your program accordingly.

This is great to include in your setup, but you can also create a calibration routine that is triggered by a button, should you need to recalibrate without restarting the Arduino entirely.

To see an example of this, in Arduino go to File → Examples → 03.Analog → Calibration. In the code, change the analog input pin to A2 and the digital output pin to pin 11. Then upload the code to your Arduino board.

This example looks for the highest and lowest values that occur during the first five seconds that the program is running. Once the Arduino is programmed, in order to recalibrate the sensor values, press the reset button and expose the sensor to the highest and lowest light conditions during the following five seconds.

		
High: 1019 Low: 729	High: 951 Low: 556	High: 342 Low: 0

Figure 7-16. The highs and lows of a light sensor value can differ according to the current conditions; calibration can help with this

See also:

- [Arduino calibration example](http://arduino.cc/en/Tutorial/Calibration) (<http://arduino.cc/en/Tutorial/Calibration>)

Constraining

Sometimes your sensor will provide readings that fall outside of your desired range (Figure 7-17). For these cases, Arduino provides a function called `constrain`. The three parameters needed are the data that is being constrained, the lowest value you would like to keep, and the highest value you would like to keep. If there are any values that are

below or above the specified range, the `constrain` function will convert them to the lowest or highest desired values, respectively.

In practice, it might look something like this:

```
/*
 * Make: Wearable Electronics
 * Constrain example
 */
//initialize variables
int lightSensorPin = A2;
int lightSensorValue = 0;
int constrainedLightSensorValue = 0;

void setup() {
  //initialize serial communication:
  Serial.begin(9600);
}

void loop() {
  //read light sensor pin and store
  // value in a variable:
  lightSensorValue = analogRead(lightSensorPin);
  //constrain the light sensor values
  // to 300 to 650
  constrainedLightSensorValue =
    constrain(lightSensorValue, 300, 650);

  //print the results:
  Serial.print("Light Sensor Value: ");
  Serial.print(lightSensorValue);
  Serial.print
    (" , constrained Light Sensor Value: ");
  Serial.println(constrainedLightSensorValue);

  // delay between readings:
  delay(100);
}
```

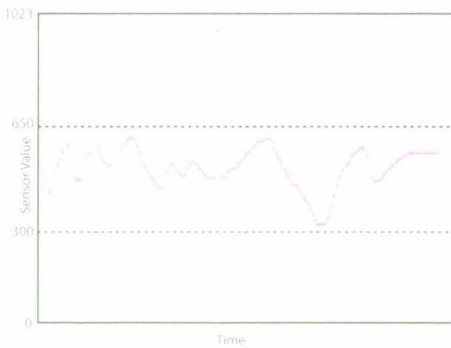


Figure 7-17 The `constrain()` function allows sensor readings to be constrained within a set range

See also:

- [Arduino `constrain\(\)` reference page](http://arduino.cc/en/Reference/Constrain) (<http://arduino.cc/en/Reference/Constrain>)
- [Arduino calibration example](http://arduino.cc/en/Tutorial/Calibration) (<http://arduino.cc/en/Tutorial/Calibration>)

Smoothing

While some sensors produce data that is smooth and predictable, others offer a dataset that's rougher around the edges (**Figure 7-18**). Smoothing can help turn an erratic datastream into something cleaner and easier to work with.

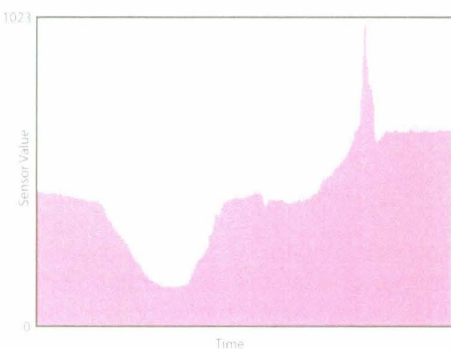


Figure 7-18 Some data requires smoothing

To give this a try in Arduino, go to File → Examples → 03.Analog → Smoothing. Once the sketch is open, change the analog input pin to A2 and upload the sketch to your Arduino. Open the serial monitor and see what the new, smoothed sensor data looks like!

See also:

- [Arduino smoothing example](http://arduino.cc/en/Tutorial/Smoothing) (<http://arduino.cc/en/Tutorial/Smoothing>)
- [Arduino's `runningAverage` class](http://playground.arduino.cc/Main/RunningAverage) (<http://playground.arduino.cc/Main/RunningAverage>)



Graphing

Looking at data in the serial monitor can be a good place to start but sometimes it's helpful to see a visual representation of how the data changes over time. Check out the [Arduino Graph example](http://arduino.cc/en/Tutorial/Graph) (<http://arduino.cc/en/Tutorial/Graph>) for more information.

Experiment: Woo! Shirt

Using the light sensor circuit that you've been working with in this section, incorporate the circuit into a shirt with the light sensor positioned in the armpit of the shirt.

Here's some starter code for you to modify. Upload it to your Arduino, and then put the shirt on:

```
/*
 * Make: Wearable Electronics
 * Woo! Shirt Experiment
 */

//Initialize variables
int lightSensorValue = 0;
int lightSensorPin = A2;
int LEDpin = 11;
int wooThreshold = 120;
```

```

void setup()
{
  //initialize serial communication:
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop(){
  // read the value from the sensor
  lightSensorValue = analogRead(lightSensorPin);

  //if the arm is up
  if(lightSensorValue > wooThreshold){
    //print Wooo!
    Serial.print("Wooo!");
    //Turn LED on
    digitalWrite(LEDpin, HIGH);
  }
  // if the arm is down
  else{
    // print boo
    Serial.println( "boo" );
    //turn LED off
    digitalWrite(LEDpin, LOW);
  }

  Serial.print(" Sensor Value: ");
  Serial.println(lightSensorValue);

  delay(100); // delay for 1/10 of a second
}

```

This code prints a "Wooo!" when it detects light and a "boo" when it does not. Based on what you've learned about the concepts of thresholds, mapping, calibration, smoothing, and constraining, create a program that reliably prints "Wooo!" when your arm is raised.



Figure 7-19. Wooo! shirt (illustration by Jen Liu)

Keep in mind that using the Serial Monitor as a feedback mechanism is great for prototyping but it will keep you tethered to the computer. Later on, you can build a more creative response into your design using LEDs or other possible outputs that you'll learn about later in [Chapter 8](#) so that you can "Woool!" more effectively in the wild.

What to Sense

It's easy to hear about a cool sensor and decide to do a project with it.

"Oh, there's a really neat X sensor that just came out. I should obviously do an X project!"

But this leads to an interaction that's designed around the technology rather than *technology that's designed around a particular interaction*.

When working with sensors, a good place to start is to think about *what you're trying to sense*. What is the motion, action, or condition? What is the context and environment? What are the important aspects to consider? Then you can ask questions like these:

- "What different sensor (or sensors) could I use?"
- "What do I want to measure?" (sound, light, pressure, presence, etc.)
- "Where should the sensors live?"
- "What should I be looking for in the data I am gathering from them?"

In the following section, you'll look at some possibilities for what to sense and a starting selection of sensors that will fit the bill. But keep in mind that this is just the tip of the iceberg. Once you have a project idea in mind, you should go out and research what's available to best help your idea come to life.

Flex

Bodies are bendy and it just so happens that *flex sensors* sense a flex or a bend ([Figure 7-20](#)). They're very good for areas of the body that bend in a broad, round arc. They work well on elbows, knees, fingers, and wrists. They are variable resistors and need to be used in combination with a voltage divider circuit (see [Figure 7-21](#)) in order to be read by a microcontroller.



Figure 7-20. Flex sensor

Here are some factors to consider when choosing a flex sensor:

Length

Flex sensors come in different lengths, usually 2.2 inches (AF 1070, SF SEN-10264) or 4.5 inches (AF 182, SF SEN-08606). Use whichever best fits your application. For instance, the longer ones work well with fingers, but the shorter ones might be more appropriate for toes.

Directionality

Flex sensors can be single or bi-directional. Bi-directional flex sensors (RO¹ RB-Ima-11) sense flex in both directions, whereas the single-direction can only sense flex in one direction. Single direction is fine for many human joints like fingers, elbows, and knees. But the bi-directional are useful for joints like the wrist,

1. RobotShop (for a complete list of supplier abbreviations, see [About Part Numbers](#) on page xiv).

where the bend can take place in both directions.

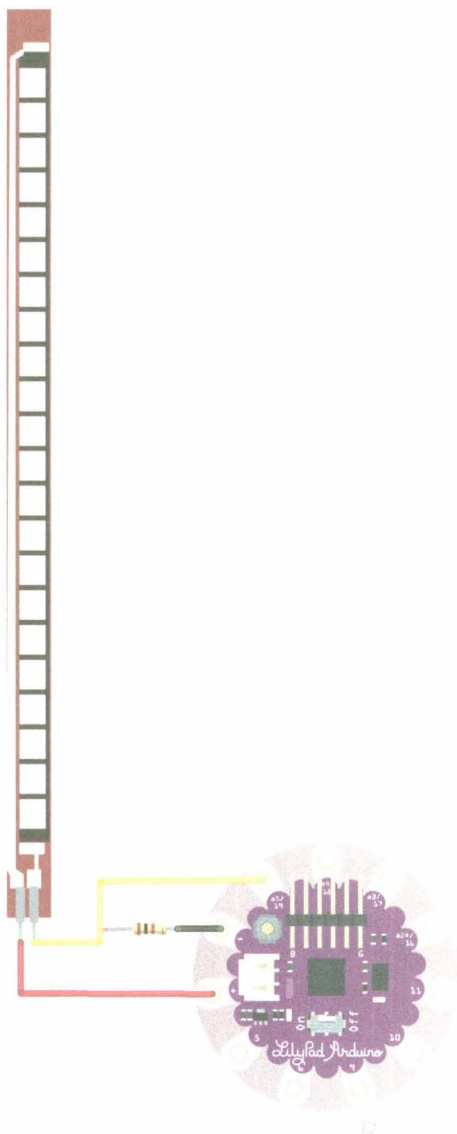


Figure 7-21. Flex sensor circuit diagram

Resistance range

Some flex sensors are also available in different resistance ranges (RO RB-1ma-24, RO RB-1ma-25). For your purposes, there are no real advantages or disadvantages that come with the options in this category. Just be sure that you're using a resistor of the appropriate size in your voltage divider circuit.

The biggest challenges in working with flex sensors are positioning and protection. In order to get an accurate reading of the flex of your elbow, the sensor needs to be positioned in the same place on your elbow every time. Creating a secure pocket for the sensor can help with this, as shown in Figures 7-22 and 7-23.



Figure 7-22. Sleeve to hold flex sensor in place



Figure 7-23. Flex sensor on a bent elbow

The other thing to consider is that while flexing is a pretty rigorous and strenuous activity, many flex systems are fairly delicate, particularly at their connection terminals. Be sure to protect your

connections. Reinforce with heat shrink, and protect them with some sort of material.



Figure 7-24 “The Gloves Project” uses flex sensors to create experimental gestural music; the project is developed by Rachel Freire, Imogen Heap, Seb Madgwick, Tom Mitchell, Hannah Perner Wilson, Kelly Snook, and Adam Stark (photograph by Hannah Perner Wilson)

Force

Bodies often touch and get touched. One way to sense touch is through the use of *force-sensing resistors*, or *FSRs* (Figure 7-25). FSRs have a makeup that’s similar to flex sensors but are configured to be sensitive to pressure rather than bending. They are also variable resistors and have delicate connections similar to flex sensors (see Figure 7-26).

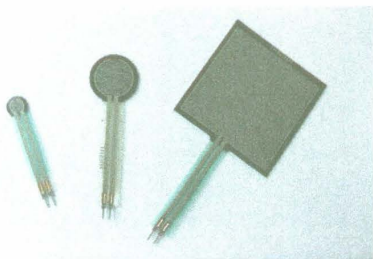


Figure 7-25 Force-sensing resistors

They come in different shapes and sizes. Different types are suited for different applications. See Table 7-1 for details.



Figure 7-26 Heatshrink tubing is used to protect the delicate solder connections between the sensor and wires

Table 7-1. FSR comparison

Type	Sensing area	Part number	Notes
Small (round)	0.16" diameter	AF SEN-09673	Very versatile; best for sensing touch at highly specific locations
Medium (round)	0.5" diameter	AF 166, SF SEN-09375	Excellent for sensing the pressure of a fingertip
Large (square)	1.75x1.5"	AF 1075, SF SEN-09376	Sits well on the top of a hand, shoulder, ball of the foot
Long	0.25x24"	AF 1071, SF SEN-09674	Great for sensing pressure along the length of an arm or leg

Figure 7-27 shows a circuit diagram, and Figure 7-28 shows how you can keep an FSR secured inside a pocket.

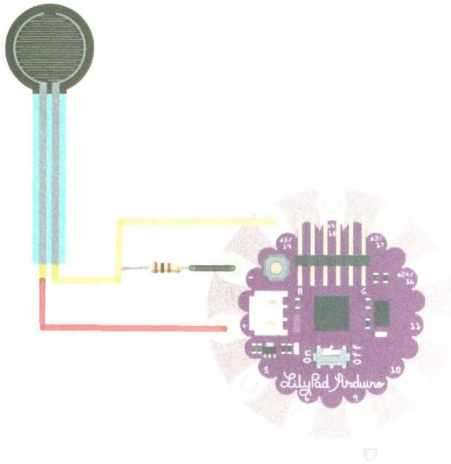


Figure 7-27. FSR circuit diagram



Figure 7-28. A pocket sewn onto the sock helps keep the FSR securely in place



Figure 7-29. When pressure is applied to the ball of the foot the change in sensor data can be read by the microcontroller



Figure 7-30. Rachael Kess's "Snowman" mask blushes when the wearer touches her cheek

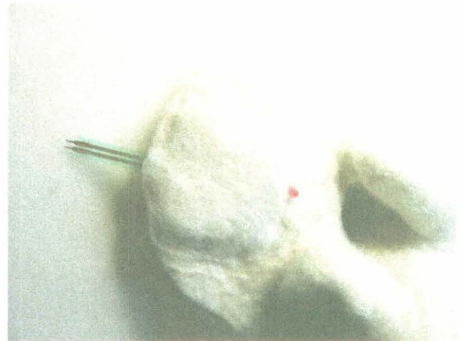


Figure 7-31. Work in progress image from Rachael Kess's "Snowman"; FSR incorporated into a hand-felted mask

Stretch

From the bend of a knee to the expansion and contraction of a rib cage with each breath, properly positioned stretch sensors (as shown in figure 7-32) can capture the fluctuating nuances and curves of the human form. A stretch sensor is simply a conductive rubber cord whose resistance decreases the more it gets stretched. This is yet another example of a variable resistor.



Figure 7-32. Stretch sensors with hooks attached

Stretch sensors come precut at different lengths with hooks crimped to either end for easy connection (RO RB-lma-12, RB-lma-13, RB-lma-14, RB-lma-15, RB-lma-16, RB-lma-17, RB-lma-18), or you can buy it by the meter and cut it to whatever length you need (AF 519). (See [Figure 7-33](#).)

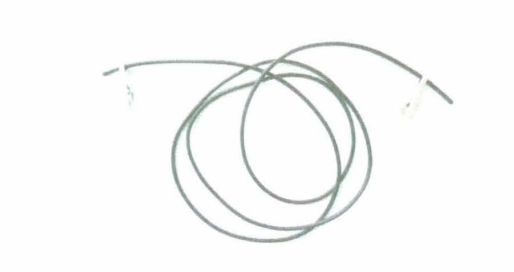


Figure 7-33. Stretch sensor with hardware for customization

Stretch sensors are a fun material to work with. They can also be elegantly incorporated into textiles through knitting or weaving.



Figure 7-34. “Aeolia” by Sarah Kettley, with Tina Downes, Martha Glazzard, Nigel Marshall, and Karen Harrigan, explores the process of incorporating stretch sensors into garments through weaving, knitting, and embroidery techniques (photograph by Tina Downes and Catherine Northall)

Movement, Orientation, and Location

People are active and mobile creatures. They reach for things they want, turn toward loud noises, and crouch down to coax the cat from under the bed. When creating wearables that react to events such as these, it is helpful to be able to sense movement.

A cheap and easy way to sense movement is through the use of tilt switches (shown in [Figure 7-35](#); see “Tilt Switches” on [page 55](#)).

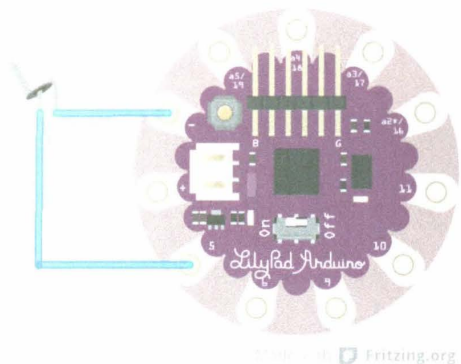


Figure 7-35. A basic tilt switch can be read by a digital input pin

But there are also far more sophisticated sensors that you can use. *Accelerometers* measure acceleration or changes in speed of movement. They can also provide a good measurement of tilt due to the changing relationship to gravity.

Accelerometers have a set number of axes—directions in which they can measure. The ones shown in Figure 7-36 are *three-axis accelerometers*, meaning they can measure acceleration on the x, y, and z plane.

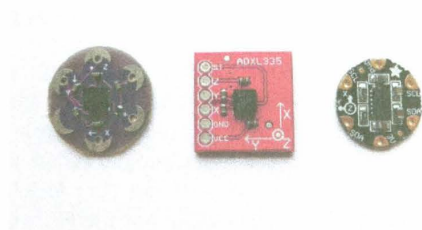


Figure 7-36. Accelerometers: LilyPad Accelerometer, ADXL335, Flora Accelerometer

The ADXL335 triple-axis accelerometer is available in a variety of form factors—both on a standard breakout board (AF 163, SF SEN-09269) as well as on a LilyPad board (SF DEV-09267). These boards contain the same chip but are intended for different uses (conductive thread circuit versus breadboard circuit). This is an analog accelerometer, meaning that they output varying voltage for each axis reading. These three outputs can be connected to three different analog inputs on the Arduino. Connections are shown in Figures 7-37 and 7-38.

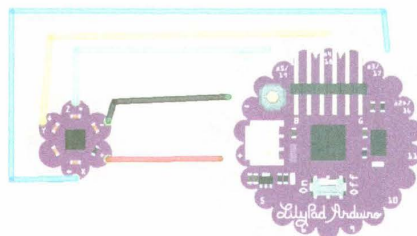


Figure 7-37. LilyPad Accelerometer circuit layout

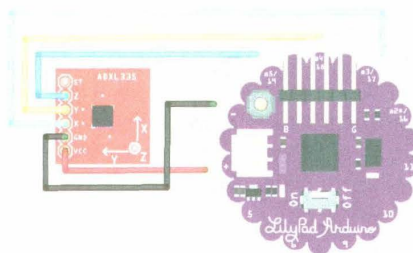


Figure 7-38. ADXL335 breakout circuit layout

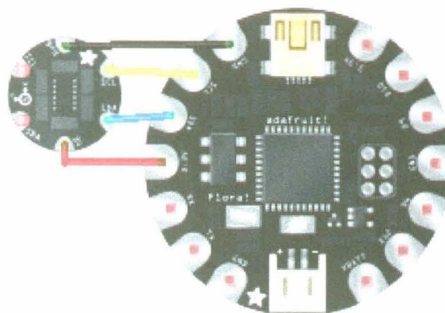
When working with analog accelerometers, in order to get actual acceleration readings, the raw data from the analog input needs to be interpreted. If need be, see the sensor datasheet for further information on how to do this. If you're just working with relative tilt, observing the changes in sensor data in the serial monitor is often good enough to get you started.



Figure 7-39. The accelerometer shirt by Leah Buechley uses accelerometer data to control the color of an RGB LED

There are also digital accelerometers that communicate their data over a serial interface. These require one fewer connection between the sensor and the circuit board; and although they are a little more complex on the code front, they often provide more functionality and slightly less noisy data.

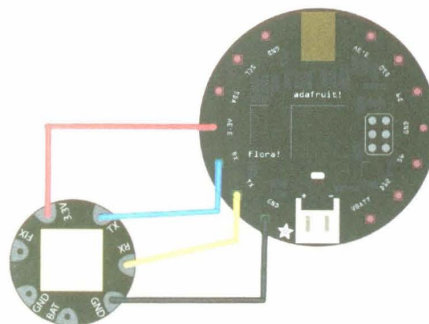
An example of a digital accelerometer is the Flora accelerometer (AF 1247). This module actually includes both an accelerometer and compass. Because it is digital, it is able to provide a lot of data with only four connections. This, like all Flora sensors, uses I2C as the serial communication method. The magnetometer on board senses magnetic north or the direction of whatever the strongest magnetic field is. This can be extremely useful when you want to determine which way a person is facing. [Figure 7-40](#) shows a circuit diagram for this component.



CC BY-SA 4.0

Figure 7-40. Flora accelerometer/compass circuit diagram

If tilt, motion, and orientation aren't enough, and you want your wearable to know where you are on the planet, GPS is the way to go. Just like your car, bike, or phone, your jacket or disco pants can have GPS, too. There are a number of Arduino-compatible GPS units available, but the Flora GPS ([Figure 7-41](#)) is a compact and sewable option.



CC BY-SA 4.0

Figure 7-41. Circuit layout for Flora GPS (AF 1059)



Figure 7-42. Flora GPS Jacket by Adafruit, Becky Stern, and Tyler Cooper (photographed by Collin Cunningham for Adafruit)

See also:

- [SparkFun accelerometer buying guide](https://www.sparkfun.com/pages/accel_gyro_guide) (https://www.sparkfun.com/pages/accel_gyro_guide)
- [LilyPad Accelerometer example](http://lilypardarduino.org/?p=384) (<http://lilypardarduino.org/?p=384>)
- [LilyPad Accelerometer Shirt](http://bit.ly/UcA1pS) (<http://bit.ly/UcA1pS>)
- [Flora Accelerometer tutorial](http://bit.ly/1zMP3Up) (<http://bit.ly/1zMP3Up>)
- [Flora GPS tutorial](http://learn.adafruit.com/flora-wearable-gps) (<http://learn.adafruit.com/flora-wearable-gps>)
- [Flora GPS Jacket](http://learn.adafruit.com/flora-gps-jacket) (<http://learn.adafruit.com/flora-gps-jacket>)

Heart Rate and Beyond

Your heart beats faster when you're excited, and your skin gets clammy when you're nervous. Besides sensing your environment and your movements, you can also use sensors to learn more about what is happening within someone's body. A great place to start sensing these biometrics is pulse or heart rate.

Optical heart rate sensors (Figure 7-43), such as the Pulse Sensor Amped (AF 1093, SF SEN-11574), are a small, lower-cost solution for measuring pulse. This type of sensor measures the mechanical flow of blood, usually in a finger or earlobe. It contains

an LED that shines light into the capillary tissue and a light sensor that reads what is reflected back. It produces varying analog voltage that can be ready by the analog input on any Arduino (Figure 7-44).

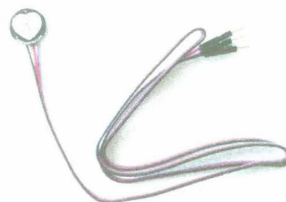


Figure 7-43. Pulse sensor

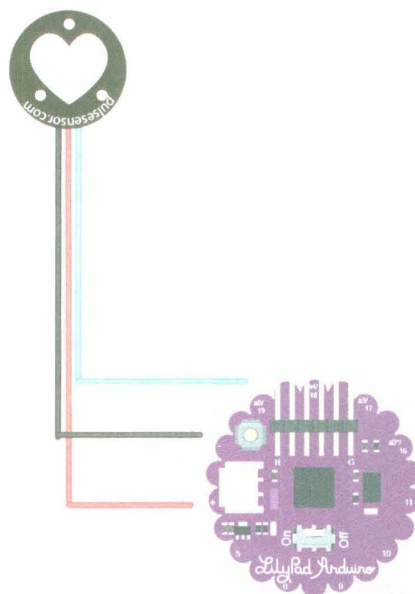


Figure 7-44. Pulse sensor circuit diagram



Figure 7-45: “Heart Strings” by Jackson McConnell and Imman Pirani uses the Pulse Sensor Amped to add another layer of exchange to Skype: each wearer feels the heartbeat of the person on the other end of the video call

Chest strap heart monitors are a more expensive but more accurate solution for measuring heart rate. They measure the actual electrical frequency of the heart through two conductive electrodes (oftentimes made of conductive fabric) that must be pressed firmly against the skin. Polar produces heart rate monitors that wirelessly transmit a signal with every heartbeat. There are multiple options for receiving this wireless signal. The Polar Heart Rate Monitor Interface (SF SEN-08661) receives the wireless signal from the heart rate monitor band and shares it with the Arduino via I2C (see Figures 7-46 and 7-47). The Heart Rate Educational Starter Pack (AF 1077) includes a simpler setup with a receiver whose output pin pulses high when a heartbeat is detected.



Figure 7-46: Polar heart rate monitor band



Figure 7-47: Polar heart rate monitor band (inside)



Figure 7-48: “Heart Spark” by Eric Boyd is a custom-designed printed circuit board necklace that receives a signal from a Polar heart rate monitor band and blinks in unison with the wearer’s heartbeat

Beyond heart rate, there are many other biological signals you might want to measure. Here are a few:

Galvanic skin response (GSR)

A method of measuring the conductivity of the skin. Changes in this conductivity can indicate a response to physical or psychological stimulus. GSR sensors are used in classic lie detectors. A GSR sensor can be built with some basic inexpensive electronic components.

Electromyography (EMG)

A method of measuring of muscle activity by detecting its electrical potential. The MuscleV3 Sensor Kit (SF SEN-11776) provides varying analog voltage so you can easily read muscle activity with an Arduino analog input pin.

Electroencephalography (EEG)

A method of measuring electrical activity in the scalp. EEG headsets are often used in thought-controlled computing applications.

To learn more about how to work with these types of sensors, check out Make Volume 26 for the “Bi-sensing” article by Sean Montgomery (Figure 7-49) and Ira Laefsky.

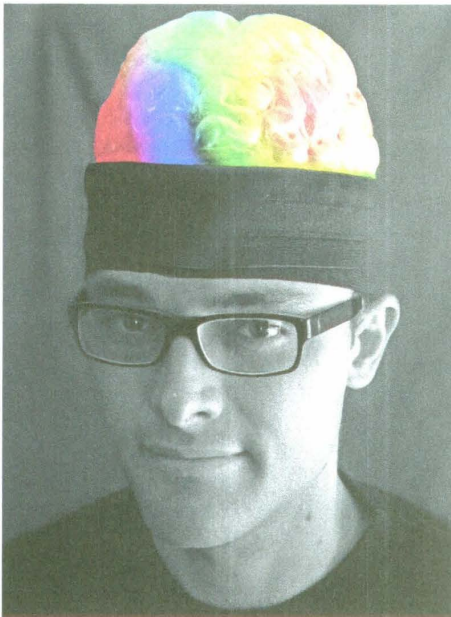


Figure 7-49. Sean Montgomery creates a variety of biometric-data-driven wearables; he is pictured here wearing his “Thinking Cap,” which responds to fluctuations in EEG signals

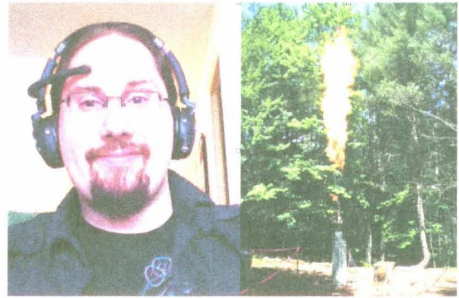


Figure 7-50. “PyroKinesis” by Seth Hardy uses EEG readings to enable wearers to control a flame effect with their brainwaves

See also:

- “Polar Heart Rate Monitor Interface + Arduino” (<http://bit.ly/UcAfxh>)
- “Pulse Sensor Getting Started Guide” (<http://bit.ly/1oMP41P>)

Proximity

Sometimes you will want to know how close or far away something is from the body. Proximity sensors are useful for detecting nearby objects, walls, or even other people (Figure 7-51). When selecting a proximity sensor, it is worth considering what your desired range or detecting distance is, as well as what sort of beam width you need to monitor.

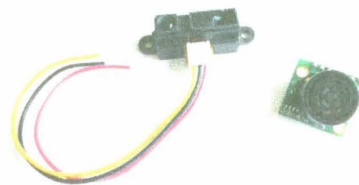


Figure 7-51. Proximity sensors

What Is a Sensor?

There are two types of proximity sensors that are fairly easy to get up and running: infrared or ultrasonic.

Infrared or IR sensors (Figure 7-52) use light to measure proximity. The sensor sends out a beam of infrared light (invisible to the human eye) that bounces off the object in front of it and is read by the sensor. IR sensors are the less expensive option for proximity sensors but are more easily tricked by heat and sunlight. They tend to have shorter, more focused distance ranges, such as 3 to 30cm (SF SEN-08959), 10 to 80cm (AF 164, SF SEN-00242), or 20 to 150cm (AF 1031, SF SEN-08958).

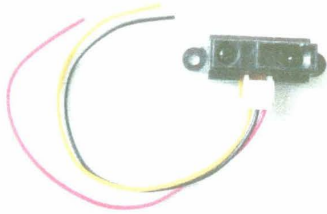


Figure 7-52 Infrared proximity sensor

These sensors can be directly connected to the analog input on any Arduino board, as shown in Figure 7-53.

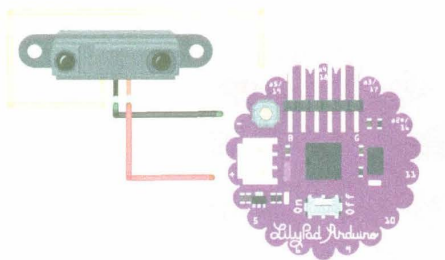


Figure 7-53 IR circuit diagram

Ultrasonic sensors (Figure 7-54) work similarly except that instead of sending out light, they send out ultrasonic sound (which can't be heard by humans). The sound bounces off whatever is proximate and returns to the sensor. The proximity is determined by the length of time it takes for the sound to return. Maxbotix manufactures a sophisticated line of ultrasonic sensors that meet a range of needs from the most basic to highly sensitive and rugged. They come in a variety of beam widths, have long sensing ranges (0 to 150 inches on their most basic model: AF 979, SF SEN-08502), and even are available with waterproof outdoor housings. Ultrasonic sensors are more expensive and bulkier than IR sensors, but they are more precise and harder to trick. A sample circuit is shown in Figure 7-55.

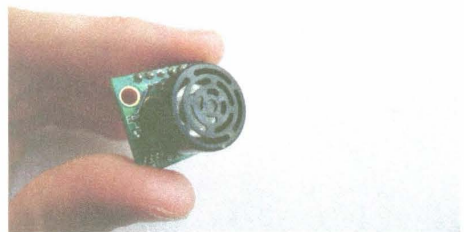


Figure 7-54 Ultrasonic proximity sensor

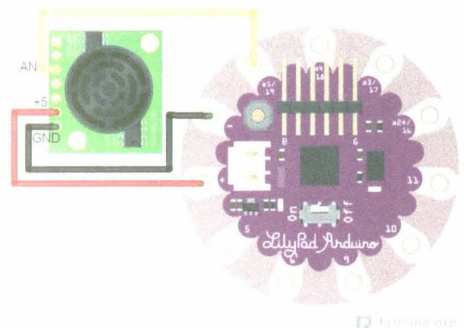


Figure 7-55 Ultrasonic circuit diagram

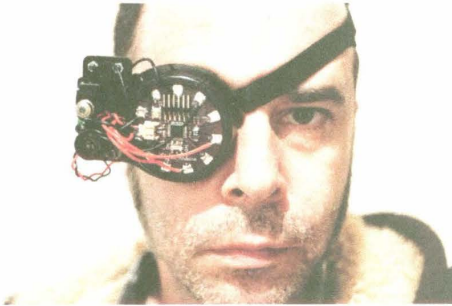


Figure 7-56. “Augmented Vision” by Greg McRoberts is a wearable seeing aid device that uses flashing RGB LED to represent fluctuating data gathered by an infrared heat sensor and ultrasonic distance sensor

Light

Remember your old friend the light sensor? You used a LilyPad Light Sensor (SF DEV-08464) in your first analog input example, but light sensors come in many other forms (Figure 7-57).

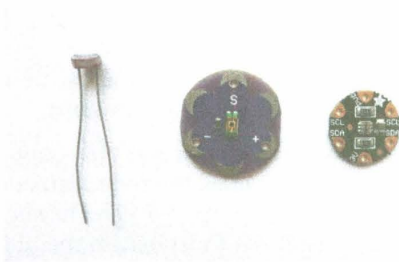


Figure 7-57. Photocell, LilyPad Light Sensor, Flora Light Sensor

The most basic type of light sensor is the photocell (AF 161, SF SEN-09088). Its resistance varies based on the level of light it senses. Some have resistance that increases as the light level increases, but some have the reverse relationship. This can be quickly determined by viewing the sensor values in the serial monitor (see the example in “Analog Input” on page 108). Figure 7-58 shows the circuit design for using this sensor.

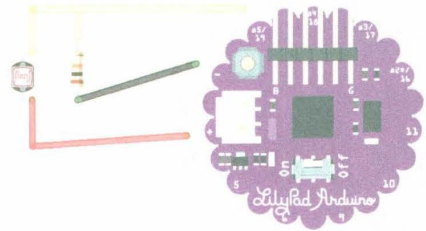


Figure 7-58. Photocell with voltage divider circuit connected to LilyPad Arduino Simple

The photocell is a great sensor to work with because it is small, easy to manipulate, and incredibly inexpensive. It can be used to sense ambient light levels, but it can also be used for less intuitive purposes like determining whether a jacket is open or closed or if the heel of a shoe is on the ground or in the air.

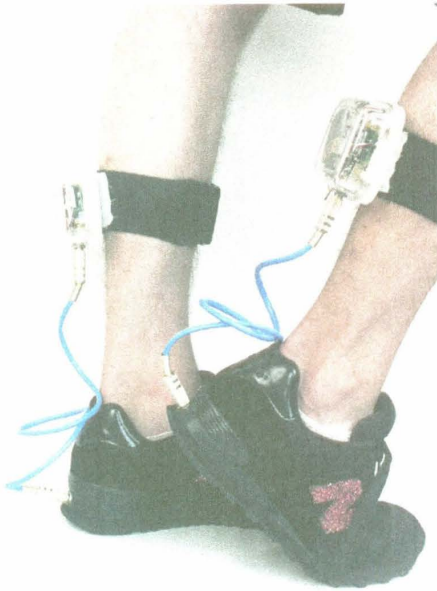
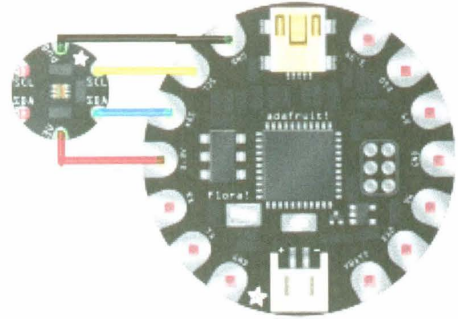


Figure 7-59. “Perform-o-shoes” by Andrew Schneider are music-controlling footwear that have a photocell embedded in the bottom of the heel; the higher the shoe is off the ground, the faster the music track will play

The Flora Lux Sensor (AF 1246) is a more sophisticated light sensor (see [Figure 7-60](#)). It measures infrared, full-spectrum, and human-visible light, which means which means that your wearable can know the difference between daylight, artificial light, or even light that’s invisible to humans. This sensor has an I2C interface.



© Inking.org

Figure 7-60. Flora Lux Sensor circuit diagram

See also:

- [Adafruit’s Flora Lux Sensor tutorial](https://learn.adafruit.com/flora-lux-sensor) (<https://learn.adafruit.com/flora-lux-sensor>)

Color

Color is hugely important in the worlds of design and fashion. Whether it be a chameleon effect or a dynamic effort to stand out from the crowd, the ability to sense color enables a garment to be able to perceive and react to its stylistic context.

There are many color sensors out there, but the Flora Color Sensor (AF 1356) has the added bonus of being sewable (see [Figure 7-61](#)) and having an onboard LED that helps to illuminate the object whose color you are trying to sense.

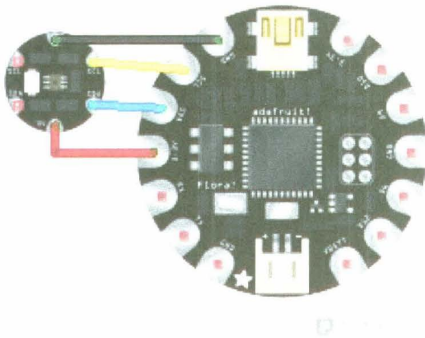


Figure 7-61. Flora Color Sensor circuit diagram

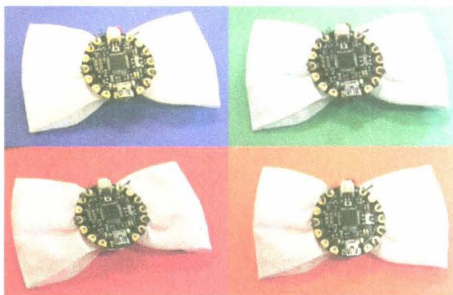


Figure 7-62. Intended to get fourth- and fifth-grade girls interested in wearable computing and programming, Glowbowz by Jaymes Dec are the world's first programmable hair bow; RGB LEDs sewn into Glowbowz can be programmed to change colors based on any sensor data; this version of Glowbowz uses a color sensor to match the bow's color to whatever outfit the wearer wants

See also:

- [Adafruit's Chameleon Scarf tutorial](https://learn.adafruit.com/chameleon-scarf) (<https://learn.adafruit.com/chameleon-scarf>)

Sound

Sounds can provide significant clues about what is going on around you. By detecting sound level, you can create wearables that are more sensitive to their environment, like a scarf that purrs when it is

whispered to or a collar that pops up in response to loud noises.

A simple microphone can act as a great sensor for audio-reactive projects. For getting started with reading an audio signal in Arduino, a small electret microphone will do the trick. These little guys can't be plugged directly into an analog input—their fluctuating signal is measured in microvolts, which is far too subtle for the ears of your microcontroller. But they are available on breakout boards (AF 1064, SF BOB-09964) that feature an amplifier chip and other components (see Figures 7-63 and 7-64) that allow it to be directly connected to an Arduino. The Adafruit model features a *trimpot* (a knob that can be adjusted with a screwdriver) on the back that allows you to make adjustments to the gain on the fly. Figure 7-65 shows a circuit you can use, and you can also use conductive thread, as shown in Figure 7-66.



Figure 7-63. Electret Microphone Amplifier—MAX4466 with Adjustable Gain (front)



Figure 7-64. Electret Microphone Amplifier—MAX4466 with Adjustable Gain (back)

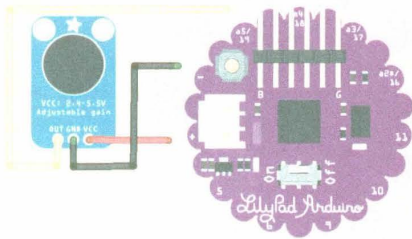


Figure 7-65. Electret Microphone Amp circuit layout



Figure 7-66. Holes meant for headers can also be used for conductive thread connections

The microphone signal can be read by an analog input pin, but you need to do further calculations to determine the amplitude (volume) that the mic is detecting. Here's some code to get you started:

```
/*
  Make: Wearable Electronics
  Mic Example
  Based on "Example Sound Level Sketch for the
  Adafruit Microphone Amplifier"
  http://bit.ly/1qLN7hk
  */

int micPin = A2;

// Sample window width in mS (50 mS = 20Hz)
int sampleWindow = 50;

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

void loop() {
  // Start of sample window
  unsigned long startMillis = millis();
  int amplitude;
  int micReading;
  int maxReading = 0;
  int minReading = 1024;

  // collect mic readings and find the
  // max and min
  while (millis() - startMillis < sampleWindow){
    micReading = analogRead(micPin);

    if (micReading > maxReading){
      maxReading = micReading;
      //save the maximum reading
    }
    else if (micReading < minReading){
      minReading = micReading;
      // save the minimum reading
    }
  }

  //find the amplitude
  amplitude = (maxReading - minReading);
  Serial.println(amplitude);
}
```

See these other examples for details:

- [Arduino Cookbook](#), Recipe 6.7
- [Adafruit Microphone Amplifier Breakout tutorial](#) (<http://bit.ly/UcBcpq>)

Temperature

Clothing provides warmth and protection. It makes sense that responsive clothing might want to react to temperature. Temperature sensors can be used to sense both environmental conditions as well as the warmth of the body. There are many temperature sensors available from analog to digital, to high temperature to waterproof, to noncontact. They are also sometimes combined with sensors for barometric pressure, humidity, and altitude.

Each of these units work in its own way and requires a bit of research and testing. An easy place to start is with a thermistor. A thermistor is a variable resistor and can be connected to the Arduino with a simple voltage divider circuit (see [Figure 7-67](#)).

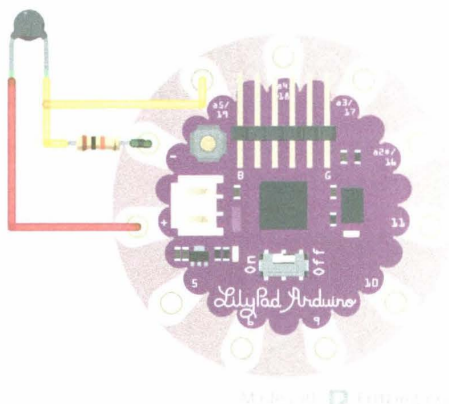


Figure 7-67. Thermistor circuit layout

Thermistors are not the most precise temperature sensors, but they are excellent for rough temperature comparison. For example, you can easily use a threshold to create a distinction between what is considered hot and what is considered cold.

For more precise reading, try working with a LilyPad Temperature Sensor (SF DEV-08777, shown in [Figure 7-68](#)), or a TMP36, a simple analog temperature sensor ([Figure 7-69](#)).

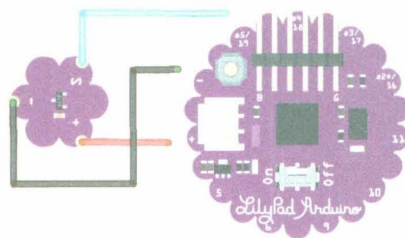


Figure 7-68. LilyPad Temperature Sensor circuit layout

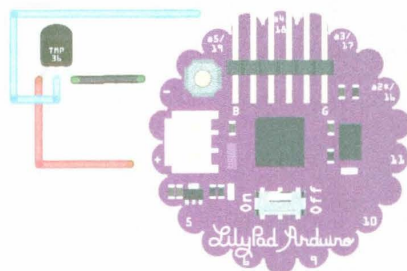


Figure 7-69. TMP36 circuit layout

The following code will work with either the LilyPad Temperature Sensor or the TMP 36. Be sure to change the `supplyVoltage` variable to whatever voltage you are working with in your circuit (see [Figures 7-70](#), [7-71](#), [7-72](#), and [7-73](#)).

```
/*
 * Make: Wearable Electronics
 * Temperature sensor example
 */

// This is a reference voltage for your power
// supply. Measure it with a multimeter when
// running and change to the correct voltage.
float supplyVoltage = 5.0;
```

```

int tempSensorPin = A2;
int tempSensorValue;
float tempSensorVoltage;

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600
  // bits per second:
  Serial.begin(9600);
}

void loop() {
  // read the temperature sensor value
  tempSensorValue = analogRead(tempSensorPin);
  ;

  // convert the reading to voltage based off
  // the reference voltage
  float tempSensorVoltage =
    (tempSensorValue * supplyVoltage) / 1024.0;

  // convert the reading to Celsius
  // converting from 10 mv per degree with
  // 500 mv offset
  float temperatureC =
    (tempSensorVoltage - 0.5) * 100;
  // to degrees ((tempSensorVoltage * 300mV)
  // times 100)

  // print in Celsius
  Serial.print("Degrees C: ");
  Serial.print(temperatureC);

  // convert to Fahrenheit
  float temperatureF =
    (temperatureC * 9.0 / 5.0) + 32.0;

  // print in Fahrenheit
  Serial.print(" , Degrees F: ");
  Serial.println(temperatureF);

  delay(100);
}

```

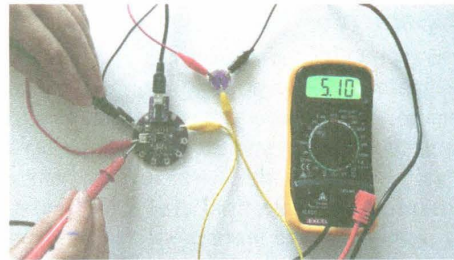


Figure 7-70. Measuring the voltage from a 5V FTDI board

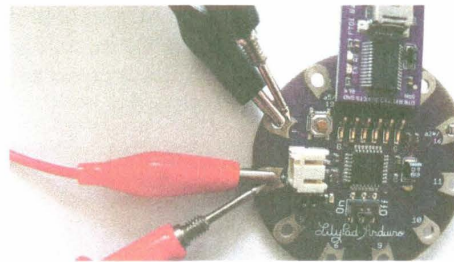


Figure 7-71. Measuring the voltage from a 5V FTDI board (detail)

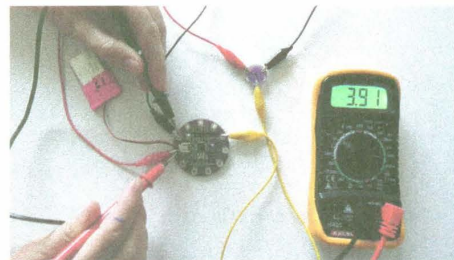


Figure 7-72. Measuring the voltage from a 3.7V lithium polymer battery

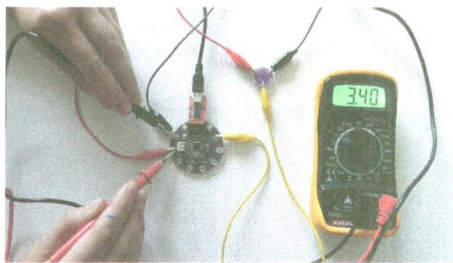


Figure 7-73. Measuring the voltage from a 3.3V FTDI board

See also:

- Adafruit's [TMP 36 Temperature Sensor Overview](http://bit.ly/UcBvAp) (<http://bit.ly/UcBvAp>)
- SparkFun Inventor's Kit, example 7
- *Arduino Cookbook*, Recipes 6.8 and 13.5

DIY Sensors

In addition to manufactured sensors, you can also create your own. As you saw earlier, a variable resistor is simply something that changes resistance in response to a changing condition. Think about this from a material perspective, and you can end up with some pretty interesting results.



Figure 7-74. "Felt Stretch Sensor" by Lara Grant

There are many DIY techniques for sandwiching a semi-resistive material between two pieces of conductive material ([Figure 7-75](#)). The semi-resistive material can be a plastic (like Velostat) or a fabric

(like some made by Eeonyx), and the conductive material can be conductive fabric, thread, yarn, wire, mesh, or anything else you can dream up. Use a connection to each conductor as the two sides of a variable resistor, and you can monitor the change in values as you apply pressure to—or flex—the sensing sandwich that you've created.

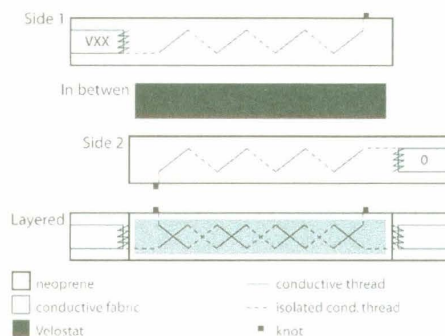


Figure 7-75. Flex sensor assembly diagram by Hannah Perner-Wilson

You can also develop a material that is itself a variable resistor. Many artists and makers have experimented with felting together sheep's wool and conductive fibers (such as steel or copper wool). The addition of the nonconductive sheep's wool to the mix creates electrical resistance. The more you compress the mixed wool, the closer the conductive fibers become to each other—thus lowering the resistance and increasing conductivity.

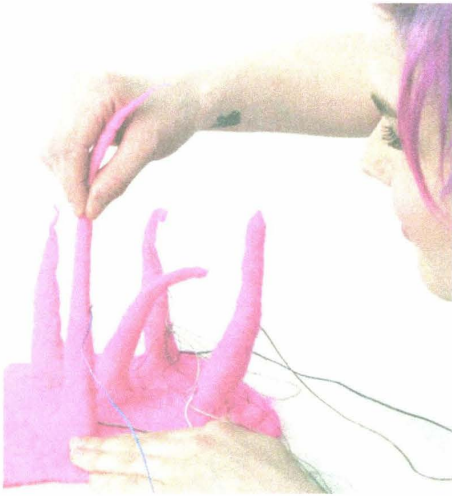


Figure 7-76. "Felt Stroke Sensor" by Lara Grant



Figure 7-77. "Felt Pressure-Sensitive Button" by Lara Grant

A similar effect can be achieved with knitting or crocheting somewhat conductive yarns

(Figure 7-78). The more the knit is stretched or pressed, the more highly conductive it becomes.



Figure 7-78. Knitted Pressure Sensors from *How To Get What You Want*

Hannah Perner-Wilson and Mika Satomi maintain a website called [How To Get What You Want](http://www.kobakant.at/DIY) (<http://www.kobakant.at/DIY>), which is home to a vast repository of DIY Wearable Technology documentation. Check out their "Sensor" section for a helpful collection of tutorials on how to make your own sensors.

Experiment: Body Listening

The interfaces you use tend to target specific areas of the body, such as hands, fingers, and feet. But what are other parts or areas of the body that aren't properly considered? For this experiment, create an interface for a part of the body that you think is not listened to enough.

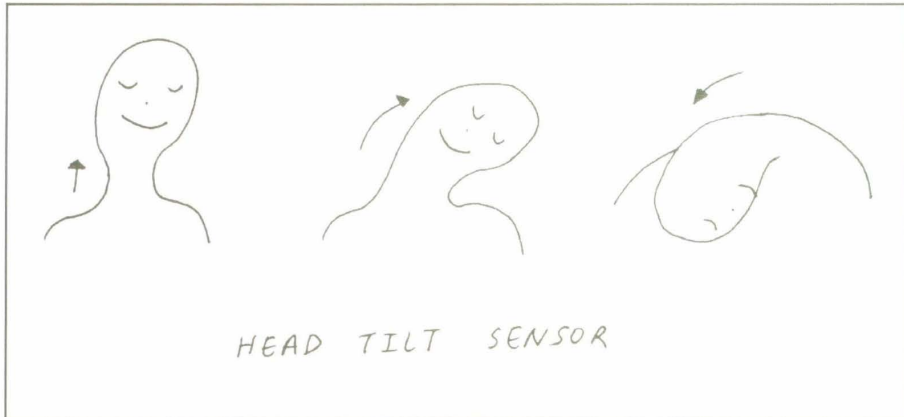


Figure 7-79. Head Tilt Sensor (illustration by Jen Liu)

Here's a process to follow:

1. Decide on a body part or area of focus.
2. Make a list of five ways you can sense or listen to that area.
3. Pick one approach that you can easily prototype.
4. Prototype it.
5. Try out your invention.
6. Make adjustments to code and hardware as needed.
7. Repeat until you think it listens well.



Figure 7-80. "Kegel Organ" by Erin Lewis allows the user to play a musical instrument through contractions of the pelvic floor muscles

Other Sensors

This introduction to sensors is really meant as a springboard to launch you into the deep and beautiful pool of sensor possibilities. Remember to start from your concept and work out from there. "Is there a sensor that senses X?" is a great question to bring to a search engine, an online forum, or your neighborhood nerd friend. From there, let the datasheet be your guide and you'll be on your way to producing smartly sensitive wearable systems.



Figure 7-81. "Concussion Helmet" by Michael Vaughan provides a visual indication when hockey players have been hit too hard in the head to return to the game