Embedded Systems

Getting Input from Sensors

Introduction:

- Sensors respond to input from the physical world and convert this into an electrical signal that Arduino can read on an input pin.
- The nature of the electrical signal provided by a sensor depends on the kind of sensor and how much information it needs to transmit.
- Some sensors (such as photoresistors and Piezo knock sensors) are constructed from a substance that alters their electrical properties in response to physical change.
- Others are sophisticated electronic modules that use their own microcontroller to process information before passing a signal on for the Arduino.

Introduction:

- **Sensors use the following methods to provide information:**
  - ***Digital on/off:*** Some devices, such as a tilt sensor and a motion sensor, simply switch a voltage on and off. These can be treated like a switch.
  - ***Analog:*** Other sensors provide an analog signal (a voltage that is proportional to what is being sensed, such as temperature or light level). Light detecting, motion, vibration, sound, and acceleration demonstrate how analog sensors can be used. All of them use the **analogRead** command.
  - ***Pulse width:*** Distance sensors, such as the PING))), provide data using pulse duration proportional to the distance value. Applications using these sensors measure the duration of a pulse using the **pulseIn** command.

Introduction:

- **Sensors use the following methods to provide information:**
  - ***Serial:*** Some sensors provide values using a serial protocol. For example, the RFID (radio frequency identification) reader and the GPS communicate through the Arduino serial port.
  - ***Synchronous protocols:*** I2C and SPI. The I2C and SPI digital standards were created for microcontrollers like Arduino to talk to external sensors and modules. These protocols are used extensively for sensors, actuators, and peripherals.

Detecting Movement:

- **Problem:**
  - You want to detect when something is moved, tilted, or shaken.

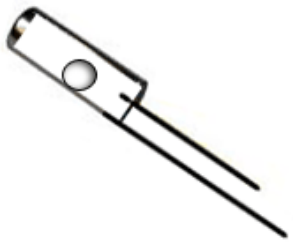
- **Solution:**
  - **This sketch uses a switch that closes a circuit when tilted, called a *tilt sensor*. The switch recipes in Chapter 5 (Recipes 5.1 and 5.2) will work with a tilt sensor substituted for the switch. The sketch below (circuit shown in Figure 6-1) will switch on the LED attached to pin 11 when the tilt sensor is tilted one way, and the LED connected to pin 12 when it is tilted the other way:**

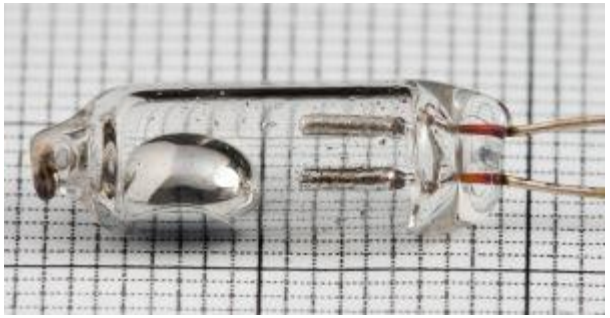
Detecting Movement:

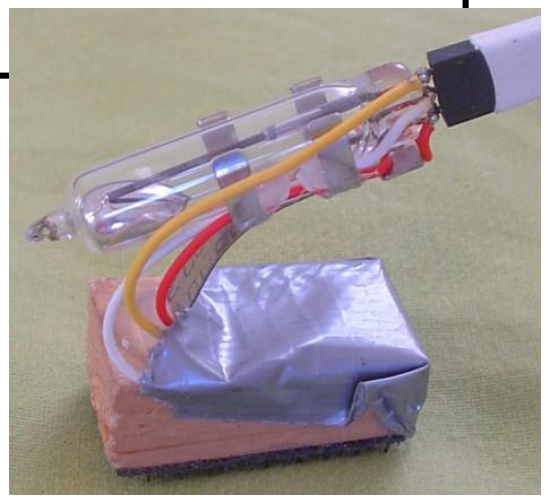
```
/*
tilt sketch

  a tilt sensor attached to pin 2 lights one of
  the LEDs connected to pins 11 and 12 depending
  on which way the sensor is tilted
*/

const int tiltSensorPin = 2;              //pin the tilt sensor is connected to
const int firstLEDPin = 11;               //pin for one LED
const int secondLEDPin = 12;              //pin for the other

void setup()
{
  pinMode (tiltSensorPin, INPUT);         //the code will read this pin
  digitalWrite (tiltSensorPin, HIGH);     //and use a pull-up resistor

  pinMode (firstLEDPin, OUTPUT);          //the code will control this pin
  pinMode (secondLEDPin, OUTPUT);         //and this one
}
```

Detecting Movement:

```
void loop()
{
  if (digitalRead(tiltSensorPin)){        //check if the pin is high
    digitalWrite(firstLEDPin, HIGH);      //if it is high turn on firstLED
    digitalWrite(secondLEDPin, LOW);      //and turn off secondLED
  }
  else{                                   //if it isn't
    digitalWrite(firstLEDPin, LOW);       //do the opposite
    digitalWrite(secondLEDPin, HIGH);
  }
}
```







Detecting Movement:

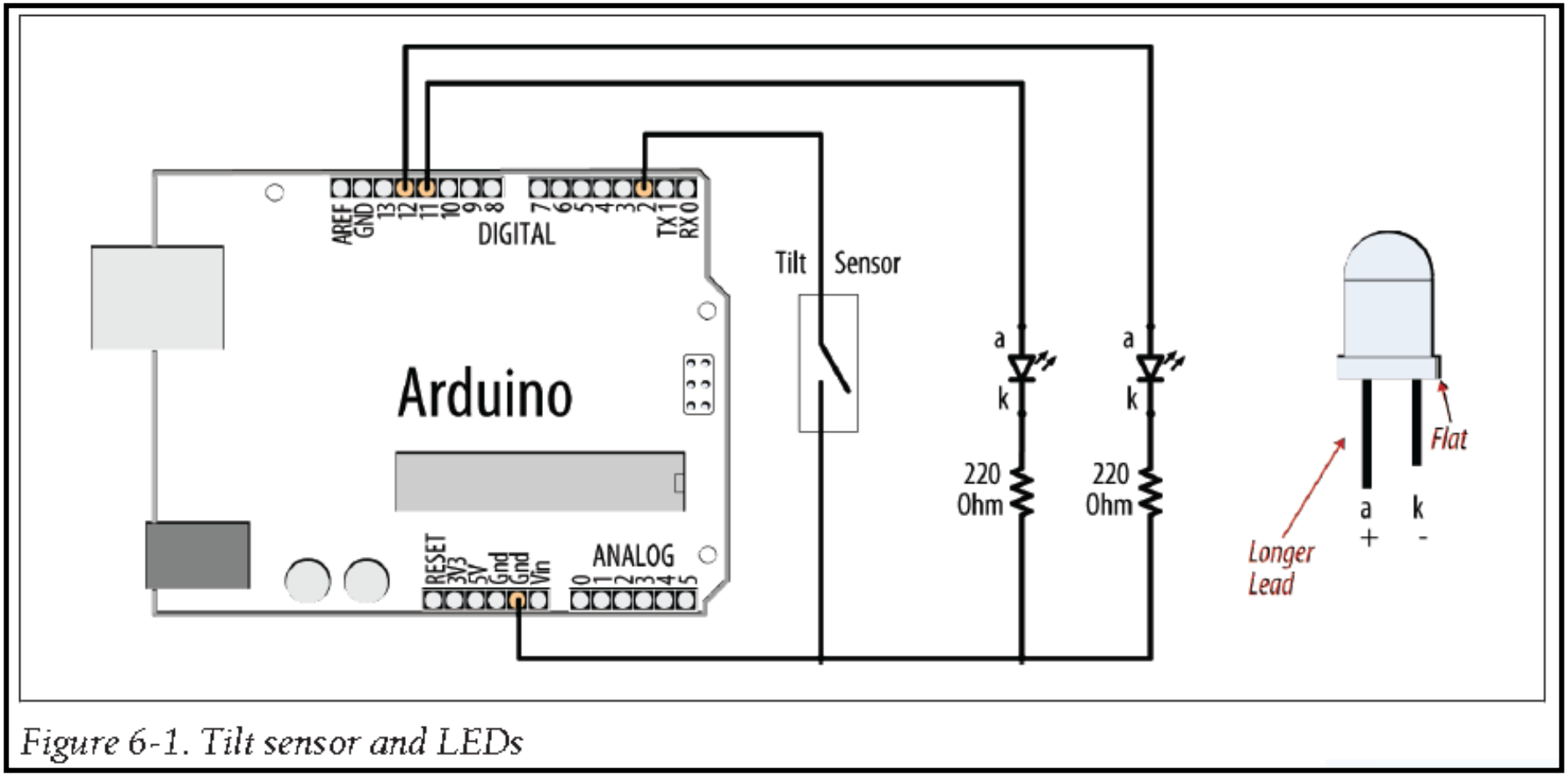


Figure 6-1. Tilt sensor and LEDs

Detecting Light:

- **Problem:**
  - You want to detect changes in light levels. You may want to detect a change when something passes in front of a light detector or to measure the light level—for example, detecting when a room is getting too dark.
- **Solution:**
  - The easiest way to detect light levels is to use a light dependent resistor (LDR). This changes resistance with changing light levels, and when connected in the circuit shown in Figure 6-2 it produces a change in voltage that the Arduino analog input pins can sense.

Detecting Light:

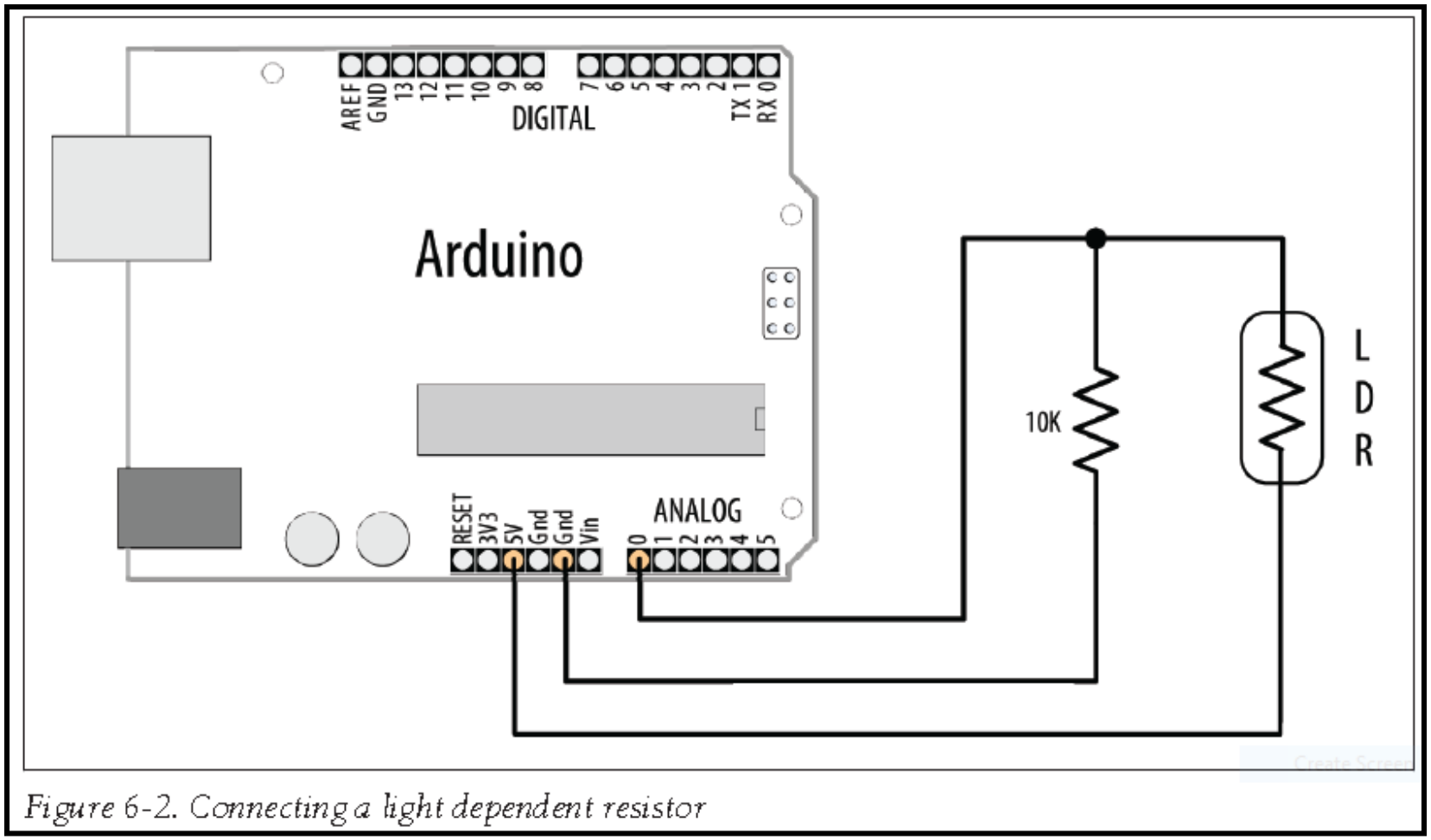


Figure 6-2. Connecting a light dependent resistor

Detecting Light:

The sketch for this recipe is simple:

```
const int ledPin =  13;      // LED connected to digital pin 13
const int sensorPin = 0;     // connect sensor to analog input 0

void setup()
{
  pinMode(ledPin, OUTPUT);  // enable output on the led pin
}

void loop()
{
  int rate = analogRead(sensorPin);    // read the analog input
  digitalWrite(ledPin, HIGH);   // set the LED on
```

Detecting Light:

```
  delay(rate);                    // wait duration dependent on light level
  digitalWrite(ledPin, LOW);      // set the LED off
  delay(rate);
}
```

Detecting Motion (Integrating Passive Infrared Detectors):

- **Problem:**
  - You want to detect when people are moving near a sensor.

- **Solution:**
  - Use a motion sensor such as a Passive Infrared (PIR) sensor to change values on a digital pin when someone moves nearby. Sensors such as the SparkFun PIR Motion Sensor (SEN- 8630) and the Parallax PIR Sensor (555-28027) can be easily connected to Arduino pins, as shown in Figure 6-3.

Detecting Motion (Integrating Passive Infrared Detectors):

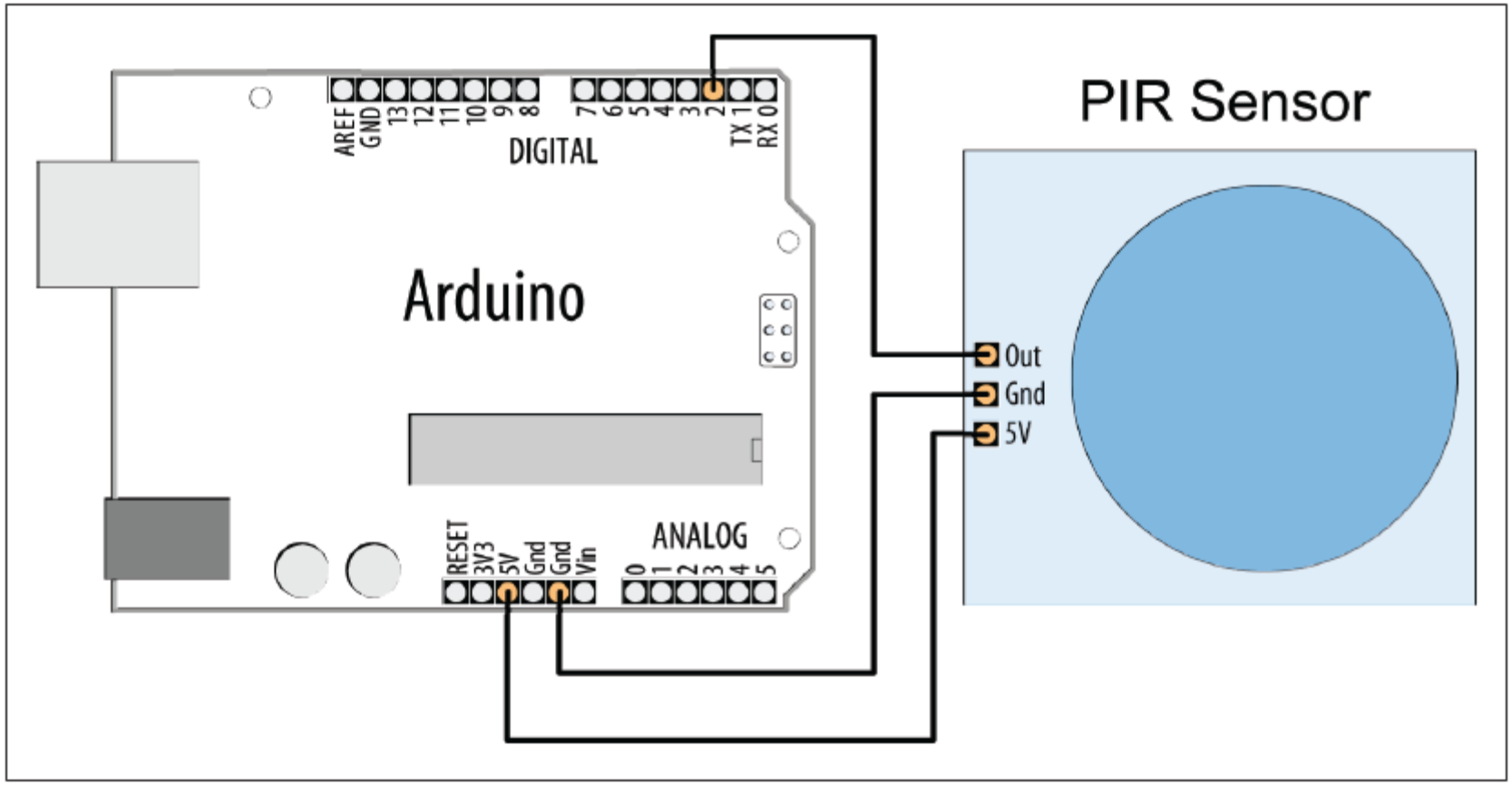
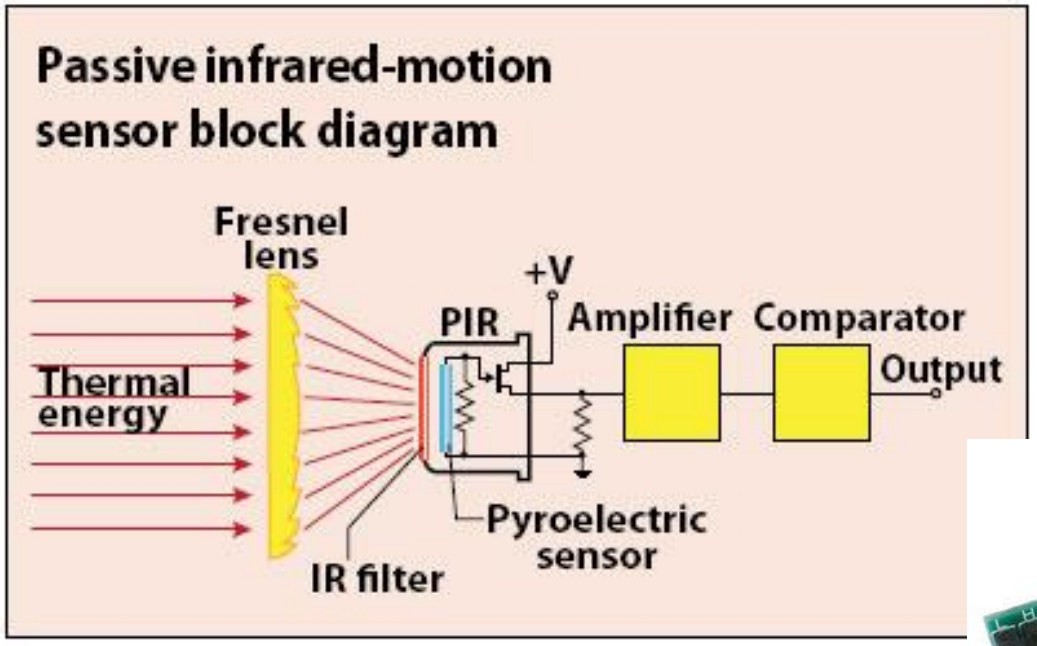
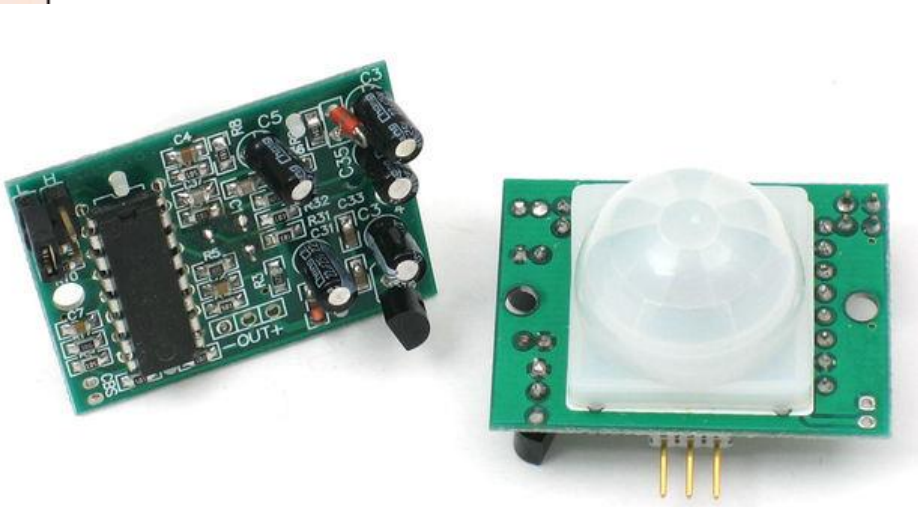


Figure 6-3. Connecting a PIR motion sensor

Detecting Motion (Integrating Passive Infrared Detectors):





Detecting Motion (Integrating Passive Infrared Detectors):

The following sketch will light the LED on Arduino pin 13 when the sensor detects motion:

```
/*
   PIR sketch
   a Passive Infrared motion sensor connected to pin 2
   lights the LED on pin 13
*/

const int ledPin = 13;            // choose the pin for the LED
const int inputPin = 2;           // choose the input pin (for the PIR sensor)

void setup() {
  pinMode(ledPin, OUTPUT);         // declare LED as output
  pinMode(inputPin, INPUT);        // declare pushbutton as input
}

void loop(){
  int val = digitalRead(inputPin);  // read input value
  if (val == HIGH)                   // check if the input is HIGH
```

Detecting Motion (Integrating Passive Infrared Detectors):

```
  {
    digitalWrite(ledPin, HIGH);     // turn LED on if motion detected
    delay(500);
    digitalWrite(ledPin, LOW);      // turn LED off
  }
}
```

Measuring Distance:

- **Problem:**
  - You want to measure the distance to something, such as a wall or someone walking toward the Arduino.

- **Solution:**
  - **This recipe uses the popular Parallax PING))) ultrasonic distance sensor to measure the distance of an object ranging from 2 centimeters to around 3 meters. It displays the distance on the Serial Monitor and flashes an LED faster as objects get closer (Figure 6-4 shows the connections):**

Measuring Distance:

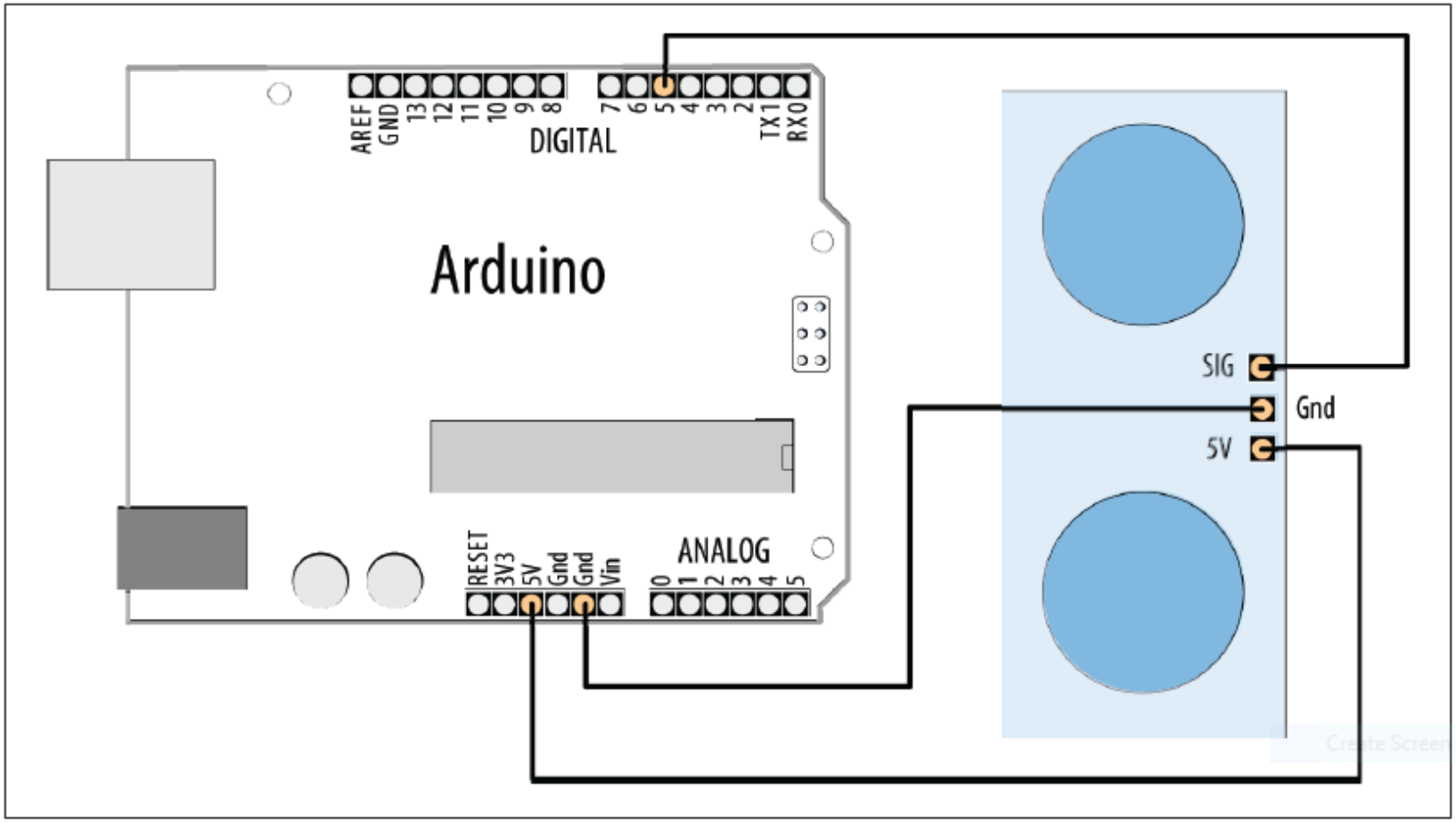
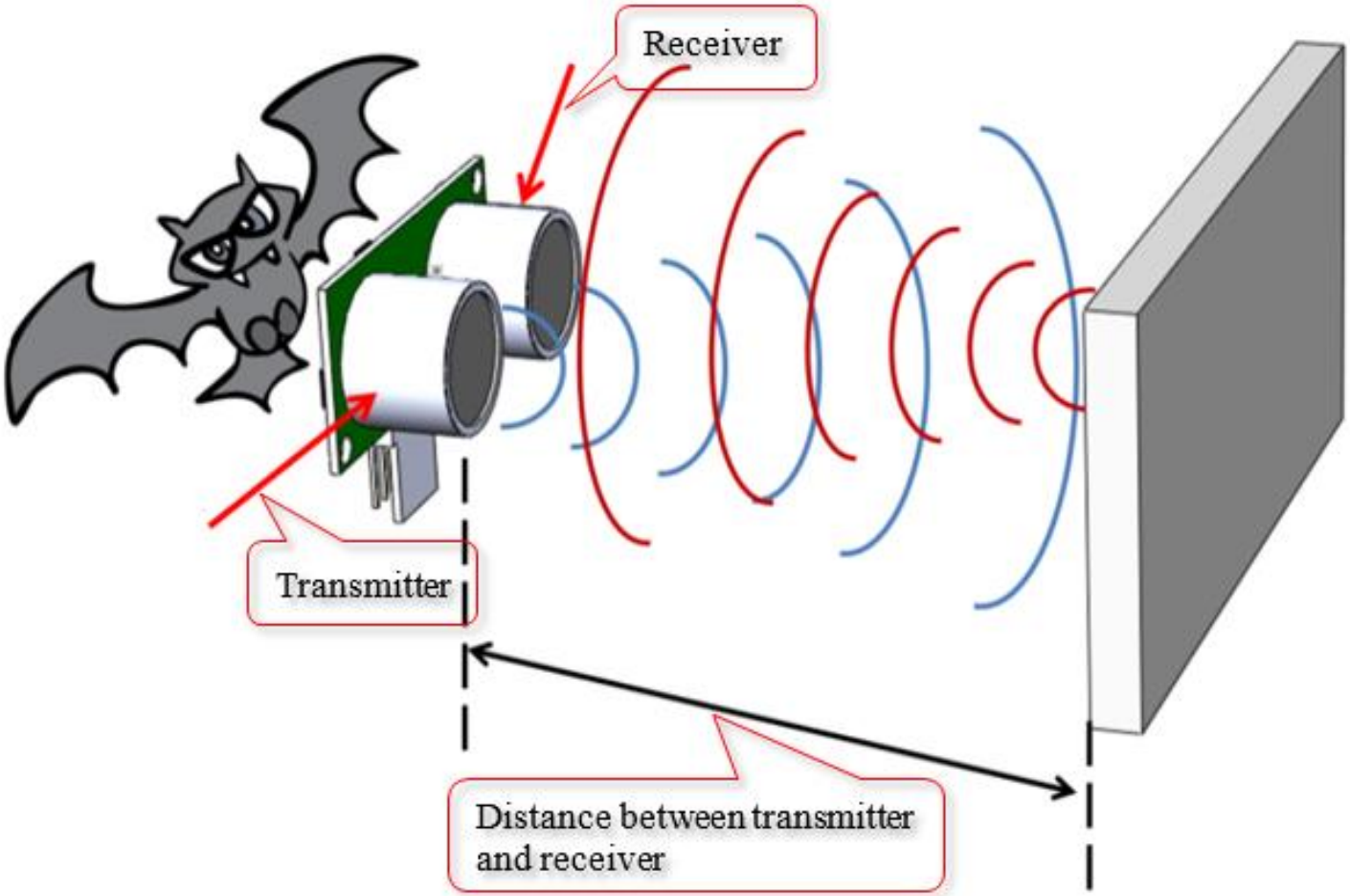


Figure 6-4. Ping))) sensor connections

Measuring Distance:



Measuring Distance:

```
/* Ping))) Sensor
 * prints distance and changes LED flash rate
 * depending on distance from the Ping))) sensor
 */


const int pingPin = 5;
const int ledPin  = 13; // pin connected to LED

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  int cm = ping(pingPin) ;
  Serial.println(cm);
  digitalWrite(ledPin, HIGH);
  delay(cm * 10 ); // each centimeter adds 10 milliseconds delay
  digitalWrite(ledPin, LOW);
  delay( cm * 10);
}
```

Measuring Distance:

```
// following code based on  http://www.arduino.cc/en/Tutorial/Ping
// returns the distance in cm
int ping(int pingPin)
{
  // establish variables for duration of the ping,
  // and the distance result in inches and centimeters:
  long duration, cm;

  // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);

  pinMode(pingPin, INPUT);
  duration = pulseIn(pingPin, HIGH);

  // convert the time into a distance
  cm = microsecondsToCentimeters(duration);
  return cm ;
}
```

Measuring Distance:

pulseIn()

Description

Reads a pulse (either HIGH or LOW) on a pin. For example, if **value** is **HIGH**, **pulseIn()** waits for the pin to go **HIGH**, starts timing, then waits for the pin to go **LOW** and stops timing. Returns the length of the pulse in microseconds or 0 if no complete pulse was received within the timeout.

The timing of this function has been determined empirically and will probably show errors in shorter pulses. Works on pulses from 10 microseconds to 3 minutes in length. Please also note that if the pin is already high when the function is called, it will wait for the pin to go **LOW** and then **HIGH** before it starts counting. This routine can be used only if interrupts are activated. Furthermore the highest resolution is obtained with short intervals.

Syntax

pulseIn(pin, value)
pulseIn(pin, value, timeout)

Measuring Distance:

```
long microsecondsToCentimeters(long microseconds)
{
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the
  // object we take half of the distance travelled.
  return microseconds / 29 / 2;
}
```

Measuring Distance:

Discussion

Ultrasonic sensors provide a measurement of the time it takes for sound to bounce off an object and return to the sensor.

The "ping" sound pulse is generated when the `pingPin` level goes `HIGH` for two microseconds. The sensor will then generate a pulse that terminates when the sound returns. The width of the pulse is proportional to the distance the sound traveled and the sketch then uses the `pulseIn` function to measure that duration. The speed of sound is 340 meters per second, which is 29 microseconds per centimeter. The formula for the distance of the round trip is: RoundTrip = microseconds / 29

So, the formula for the one-way distance in centimeters is: microseconds / 29 / 2

The MaxBotix EZ1 is another ultrasonic sensor that can be used to measure distance. It is easier to integrate than the Ping))) because it does not need to be "pinged." It can provide continuous distance information, either as an analog voltage or proportional to pulse width. Figure 6-5 shows the connections.

Measuring Distance:

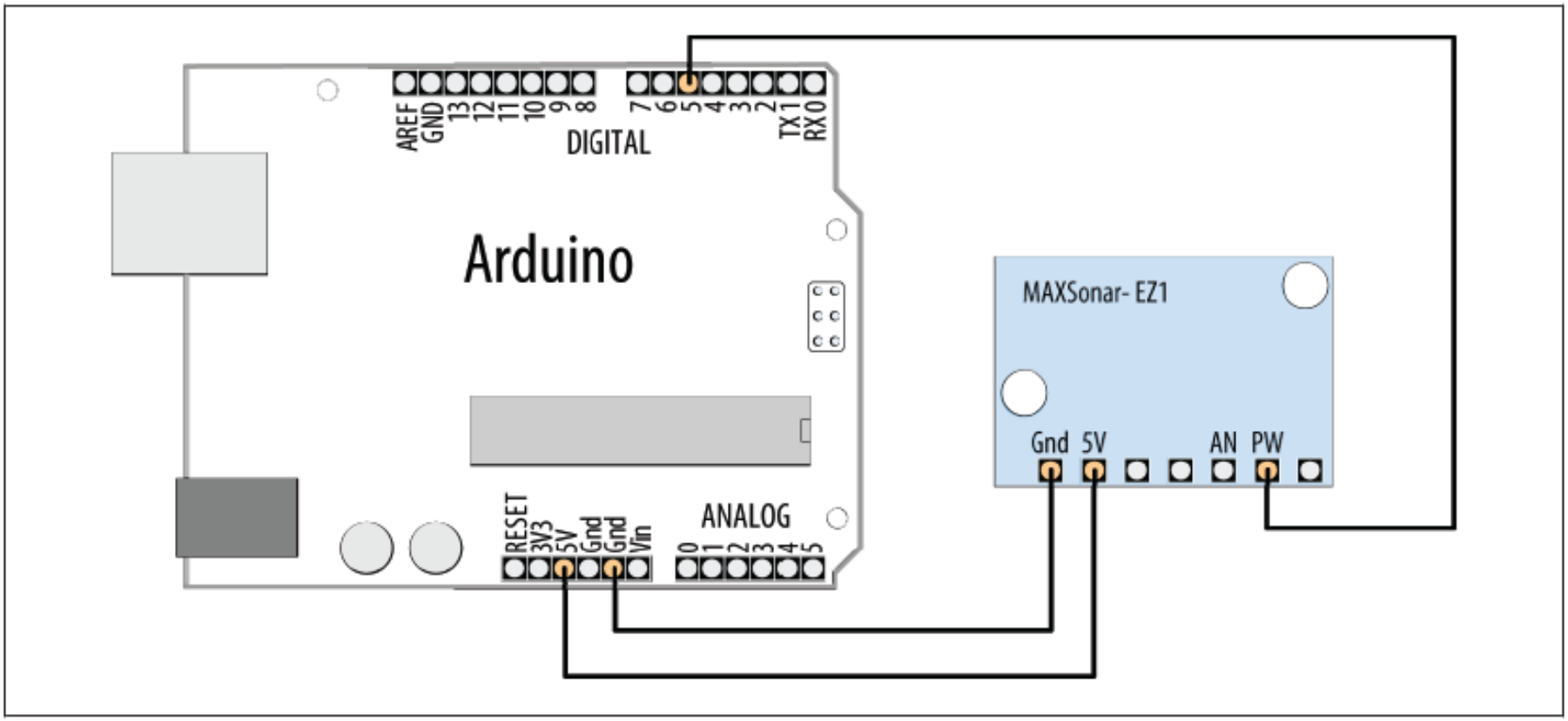


Figure 6-5. Connecting EZ1 PW output to a digital input pin

Measuring Distance:

```
const int sensorPin = 5;
const int ledPin  = 13; // pin connected to LED

long value = 0;
int cm = 0;
int inches = 0;

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  value = pulseIn(sensorPin, HIGH) ;
  cm = value / 58;         // pulse width is 58 microseconds per cm
  inches = value / 147;    // which is 147 microseconds per inch
  Serial.print(cm);
  Serial.print(',');
  Serial.println(inches);

  digitalWrite(ledPin, HIGH);
  delay(cm * 10 ); // each centimeter adds 10 milliseconds delay
```

Measuring Distance:

```
  digitalWrite(ledPin, LOW);
  delay( cm * 10);

  delay(20);
}
```

Measuring Distance:

You can also obtain a distance reading from the EZ1 through its analog output—connect the AN pin to an analog input and read the value with analogRead. The following code prints the analog input converted to inches:

```
value = analogRead(0);
inches = value / 2;   // each digit of analog read is around 5mv
Serial.println(inches);
```

The analog output is around 9.8mV per inch. The value from analogRead is around 4.8mV per unit (see Recipe 5.6 for more on analogRead) and the preceding code rounds these so that each group of two units is one inch. The rounding error is small compared to the accuracy of the device, but if you want a more precise calculation you can use floating point as follows:

```
value = analogRead(0);
float mv = (value /1024.0) * 5000 ;
float inches  =  mv / 9.8; // 9.8mv per inch
Serial.println(inches) ;
```

Measuring Distance Accurately:

- **Problem:**
  - You want to measure how far objects are from the Arduino with more accuracy than in Recipe 6.4

- **Solution:**
  - Infrared (IR) sensors generally provide an analog output that can be measured using analogRead. They can have greater accuracy than ultrasonic sensors, albeit with a smaller range (a range of 10 centimeters to 1 or 2 meters is typical for IR sensors). This sketch provides similar functionality to Recipe 6.4, but it uses an infrared sensor—the Sharp GP2Y0A02YK0F (Figure 6-6 shows the connections):.

Measuring Distance Accurately:

```
/* ir-distance sketch
 * prints distance and changes LED flash rate based on distance from IR sensor
 */

const int ledPin    = 13; // the pin connected to the LED to flash
const int sensorPin = 0;  // the analog pin connected to the sensor

const long referenceMv = 5000; // long int to prevent overflow when multiplied

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}
```

Measuring Distance Accurately:

```
void loop()
{
  int val = analogRead(sensorPin);
  int mV = (val * referenceMv) / 1023;

  Serial.print(mV);
  Serial.print(",");
  int cm = getDistance(mV);
  Serial.println(cm);

  digitalWrite(ledPin, HIGH);
  delay(cm * 10 ); // each centimeter adds 10 milliseconds delay
  digitalWrite(ledPin, LOW);
  delay( cm * 10);

  delay(100);
}
```

Measuring Distance Accurately:

```
// the following is used to interpolate the distance from a table
// table entries are distances in steps of 250 millivolts
const int TABLE_ENTRIES = 12;
const int firstElement = 250; // first entry is 250 mV
const int INTERVAL  = 250; // millivolts between each element
static int distance[TABLE_ENTRIES] = {150,140,130,100,60,50,40,35,30,25,20,15};

int getDistance(int mV)
{
   if( mV >  INTERVAL * TABLE_ENTRIES-1 )
      return distance[TABLE_ENTRIES-1];
   else
   {

      int index = mV / INTERVAL;
      float frac = (mV % 250) / (float)INTERVAL;
      return distance[index] - ((distance[index] - distance[index+1]) * frac);
    }
}
```

Measuring Distance Accurately:

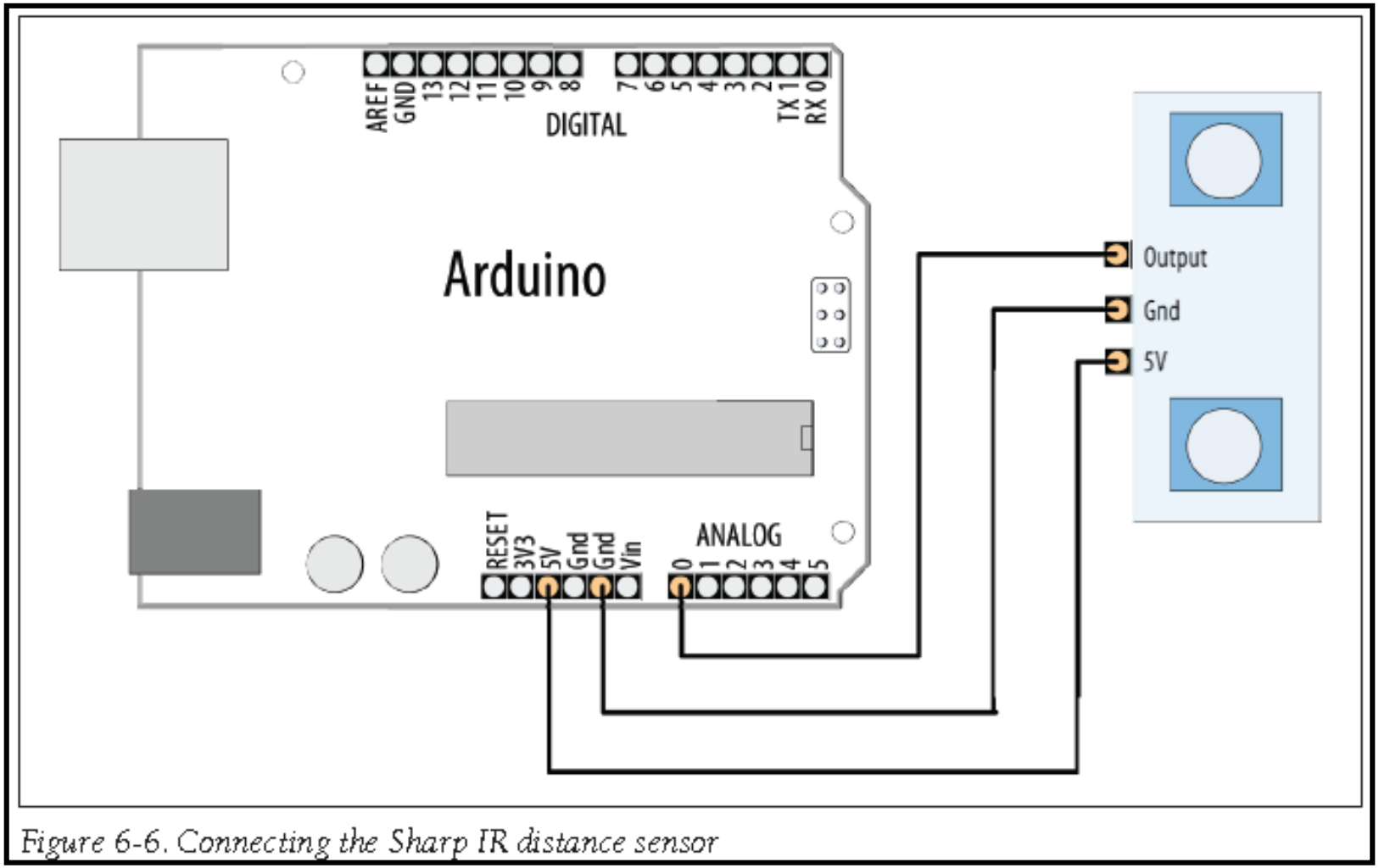


Figure 6-6. Connecting the Sharp IR distance sensor

Measuring Distance Accurately:

Discussion

The output from the IR sensor is not linear—in other words, the value read from analogRead is not proportional to distance. So, the calculation is more complicated than the one used in Recipe 6.4. The sketch in this recipe's Solution uses a table to interpolate the actual distance by finding the nearest entry in the table and adjusting it based on the ratio of the measured value to the next table entry (this technique is called *interpolating*). You may need to adjust the table values for your sensor—you can do this with information from your data sheet or through trial and error.

The conversion from voltage to distance is done in this function:

```
int getDistance(int mV)
```

Measuring Distance Accurately:

The function first checks if the value is within the range given in the table. The shortest valid distance is returned if the value is not within range:

```
if( mV >  INTERVAL * TABLE_ENTRIES )
   return distance[TABLE_ENTRIES-1]; //TABLE_ENTRIES-1 is last valid entry
```

If the value is within the table range, integer division calculates which entry is closest but is lower than the reading:

```
int index = mV / INTERVAL ;
```

The modulo operator (see Chapter 3) is used to calculate a fractional value when a reading falls between two entries:

```
float frac = (mV % 250) / (float)INTERVAL;

return distance[index] + (distance[index]* (frac / interval));
```

Detecting Vibration:

- **Problem:**
  - You want to respond to vibration; for example, when a door is knocked on.

- **Solution:**
  - **A Piezo sensor responds to vibration. It works best when connected to a larger surface that vibrates. Figure 6-7 shows the connections:**

Detecting Vibration:

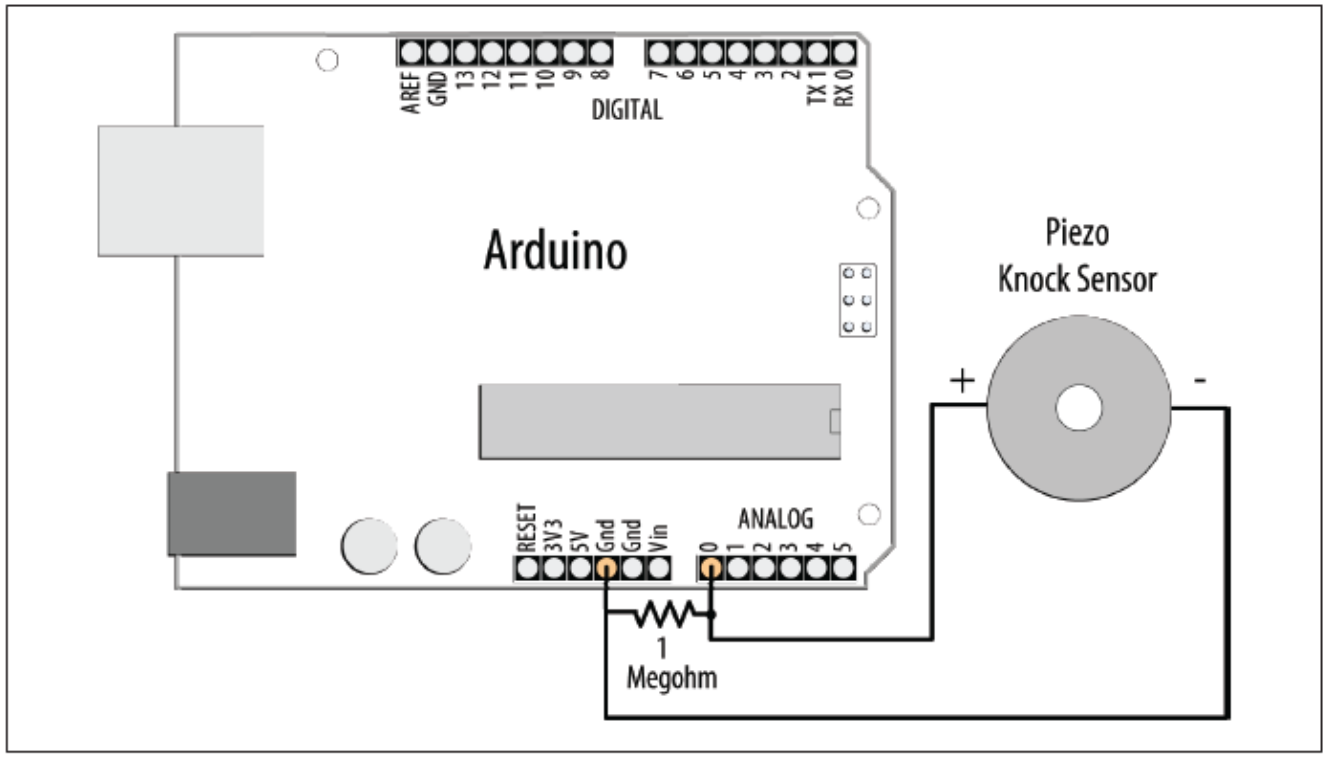
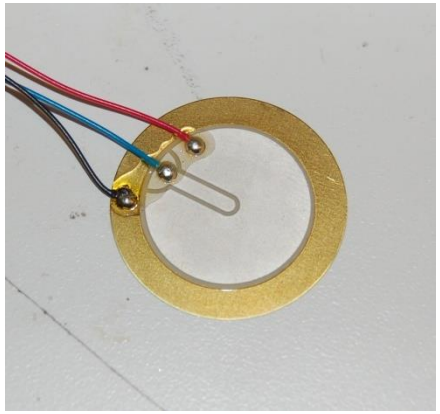
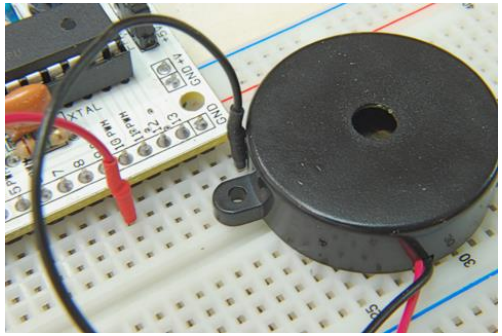


Figure 6-7. Knock sensor connections





Detecting Vibration:

```
/* piezo sketch
 * lights an LED when the Piezo is tapped
 */

const int sensorPin = 0;  // the analog pin connected to the sensor
const int ledPin  = 13;     // pin connected to LED
const int THRESHOLD = 100;


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

void loop()
{
  int val = analogRead(sensorPin);
  if (val >= THRESHOLD)
  {
    digitalWrite(ledPin, HIGH);
    delay(100);  // to make the LED visible
  }
  else
    digitalWrite(ledPin, LOW);
}
```

Detecting Vibration:

Discussion

A Piezo sensor, also known as a knock sensor, produces a voltage in response to physical stress. The more it is stressed, the higher the voltage. The Piezo is polarized and the positive side (usually a red wire or a wire marked with a "+") is connected to the analog input; the negative wire (usually black or marked with a "–") is connected to ground. A high-value resistor (1 megohm) is connected across the sensor.

Detecting Sound:

- **Problem:**
  - You want to detect sounds such as clapping, talking, or shouting.

- **Solution:**
  - This recipe uses the BOB-08669 breakout board for the Electret Microphone (Spark-Fun). Connect the board as shown in Figure 6-8 and load the code to the board.

Detecting Sound:

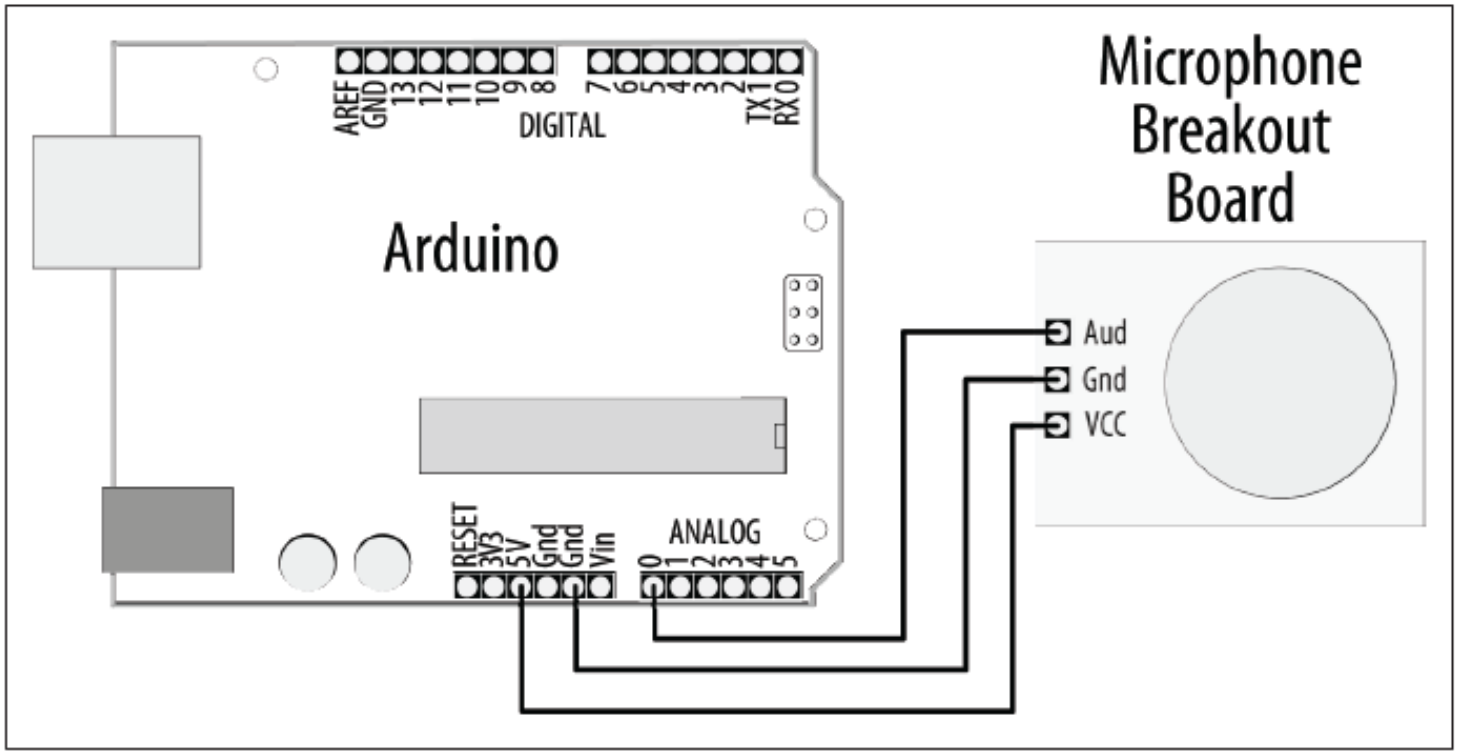


Figure 6-8. Microphone board connections

Detecting Sound:

The built-in LED on Arduino pin 13 will turn on when you clap, shout, or play loud music near the microphone. You may need to adjust the threshold—use the Serial Monitor to view the high and low values, and change the threshold value so that it is between the high values you get when noise is present and the low values when there is little or no noise. Upload the changed code to the board and try again:

```
/*
microphone sketch

SparkFun breakout board for Electret Microphone is connected to analog pin 0
*/

const int ledPin = 13;            //the code will flash the LED in pin 13
const int middleValue = 512;      //the middle of the range of analog values
const int numberOfSamples = 128;  //how many readings will be taken each time

int sample;                       //the value read from microphone each time
long signal;                      //the reading once you have removed DC offset
long averageReading;              //the average of that loop of readings

long runningAverage=0;          //the running average of calculated values
const int averagedOver= 16;     //how quickly new values affect running average
                                //bigger numbers mean slower

const int threshold=400;        //at what level the light turns on
```

Detecting Sound:

```
void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  long sumOfSquares = 0;
  for (int i=0; i<numberOfSamples; i++) { //take many readings and average them
    sample = analogRead(0);               //take a reading
    signal = (sample - middleValue);      //work out its offset from the center
    signal *= signal;                     //square it to make all values positive
    sumOfSquares += signal;               //add to the total
  }
  averageReading = sumOfSquares/numberOfSamples;     //calculate running average
  runningAverage=(((averagedOver-1)*runningAverage)+averageReading)/averagedOver;

  if (runningAverage>threshold){          //is average more than the threshold ?
    digitalWrite(ledPin, HIGH);           //if it is turn on the LED
  }else{
    digitalWrite(ledPin, LOW);            //if it isn't turn the LED off
  }
  Serial.println(runningAverage);         //print the value so you can check it
}
```

Detecting Sound:

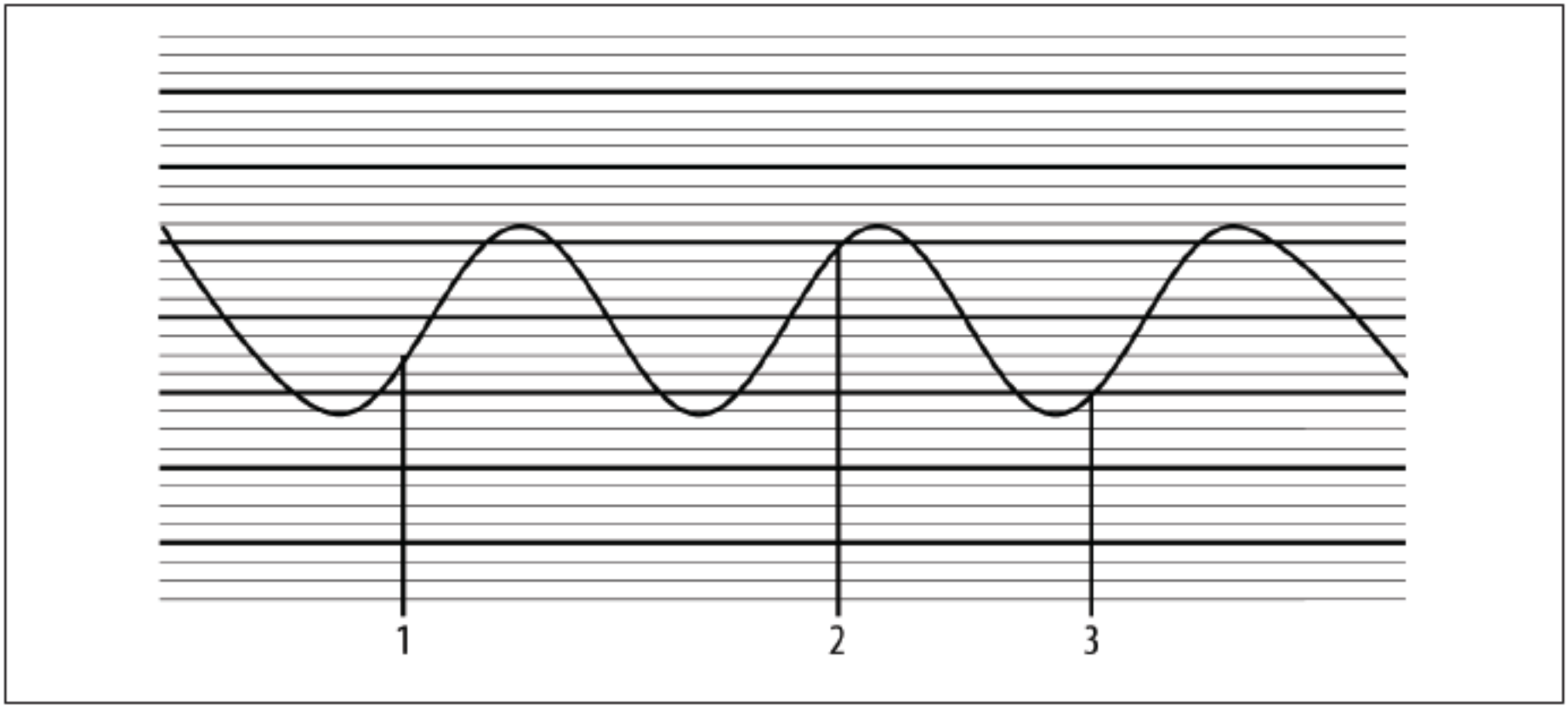


Figure 6-9. Audio signal measured in three places

Detecting Sound:

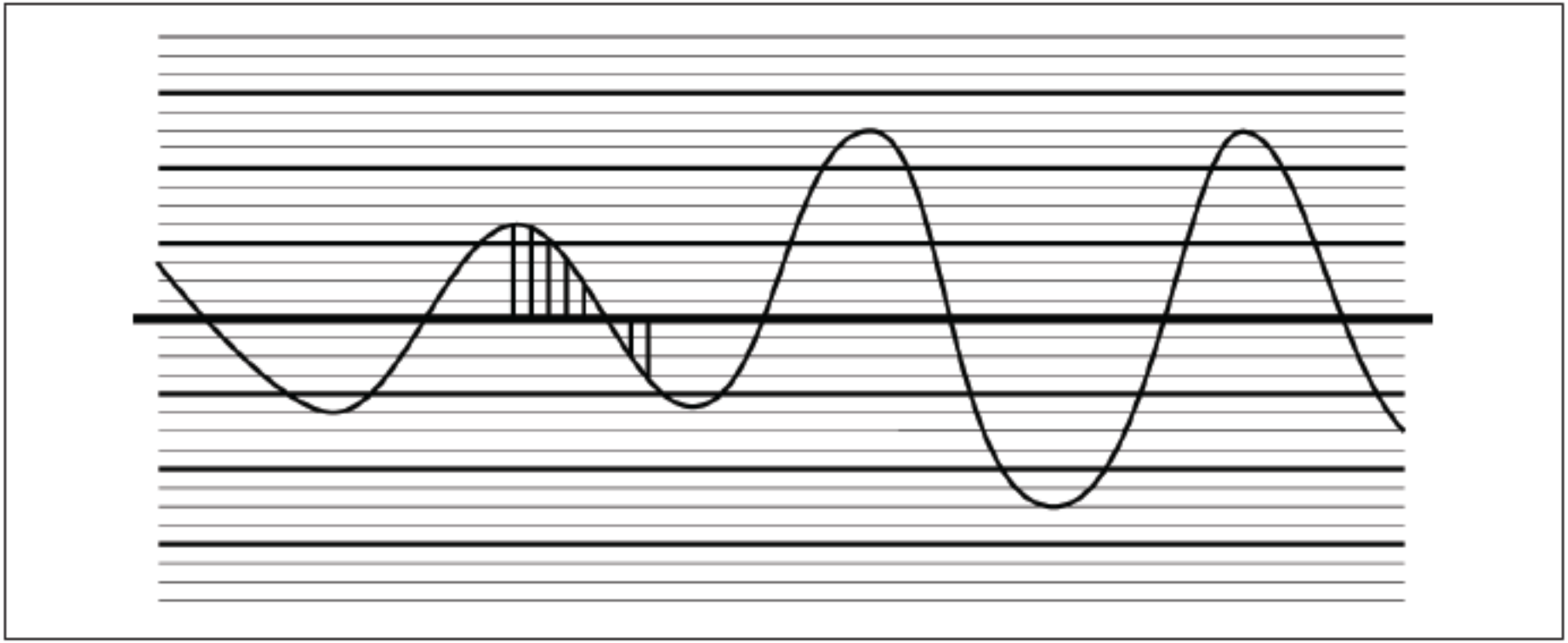


Figure 6-10. Audio signal showing DC offset (signal midpoint)

Measuring Temperature:

- **Problem:**
  - You want to display the temperature or use the value to control a device; for example, to switch something on when the temperature reaches a threshold.

- **Solution:**
  - **This recipe displays the temperature in Fahrenheit and Celsius (Centigrade) using the popular LM35 heat detection sensor. The sensor looks similar to a transistor and is connected as shown in Figure 6-11:**

Measuring Temperature:

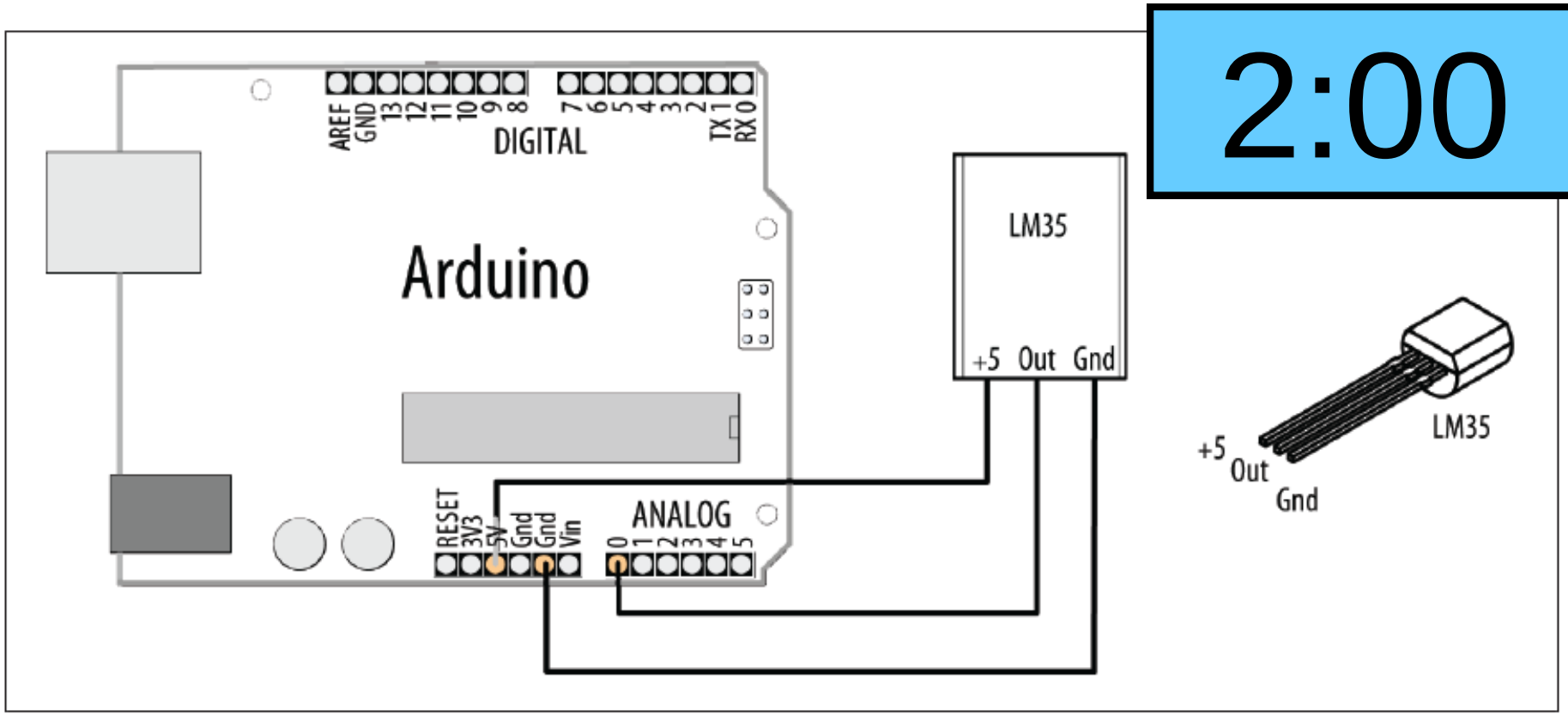


Figure 6-11. Connecting the LM35 temperature sensor

Measuring Temperature:

```
/*
 lm35 sketch
 prints the temperature to the Serial Monitor
 */

const int inPin = 0; // analog pin


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


void loop()
{
  int value = analogRead(inPin);
```

Measuring Temperature:

```
  Serial.print(value); Serial.print(" > ");
  float millivolts = (value / 1024.0) * 5000;
  float celsius = millivolts / 10;  // sensor output is 10mV per degree Celsius
  Serial.print(celsius);
  Serial.print(" degrees Celsius, ");

  Serial.print( (celsius * 9)/ 5 + 32 );  //  converts to fahrenheit
  Serial.println(" degrees Fahrenheit");

  delay(1000); // wait for one second

}
```

Measuring Temperature:

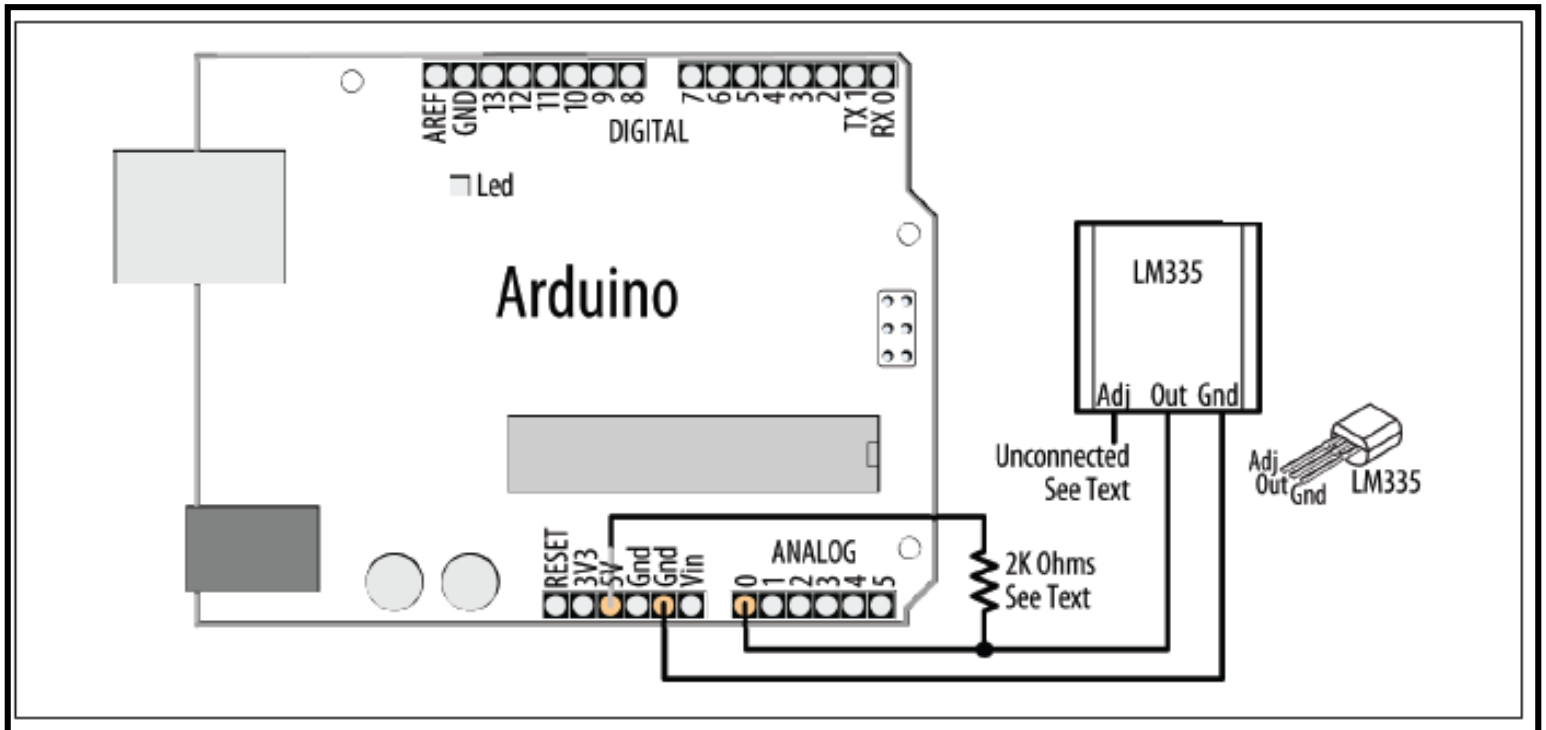


Figure 6-12. Connecting the LM335 temperature sensor

You can improve the accuracy by wiring the unconnected *adj* pin to the slider of a 10K potentiometer with the other leads connected to +5V and Gnd. Adjust the pot to get a reading to match a known accurate thermometer.

Measuring Temperature:

The LM335 output is 10mV per degree Kelvin, so zero degrees Celsius results in 2.731 volts. A series resistor is required to set the operating current. A 2K ohm resistor is often used, but 2.2K ohms can also be used. Here is a sketch that displays temperature using the LM335 (Figure 6-12 shows the connections):

```
/*
  lm335 sketch
  prints the temperature to the Serial Monitor
 */

const int inPin = 0; // analog pin


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

Measuring Temperature:

```
void loop()
{
  int value = analogRead(inPin);
  Serial.print(value); Serial.print(" > ");
  float millivolts = (value / 1024.0) * 5000;
  // sensor output is 10mV per degree Kelvin, 0 Celsius is 273.15
  float celsius = (millivolts / 10) - 273.15 ;

  Serial.print(celsius);
  Serial.print(" degrees Celsius, ");

  Serial.print( (celsius * 9)/ 5 + 32 );  //  converts to fahrenheit
  Serial.println(" degrees Fahrenheit");

  delay(1000); // wait for one second
}
```

Reading RFID Tags:

- **Problem:**
  - You want to read an RFID tag and respond to specific IDs.

- **Solution:**
  - Figure 6-13 shows a Parallax RFID (radio frequency identification) reader connected to the Arduino serial port. (You may need to disconnect the reader from the serial port when uploading the sketch.)

Reading RFID Tags:

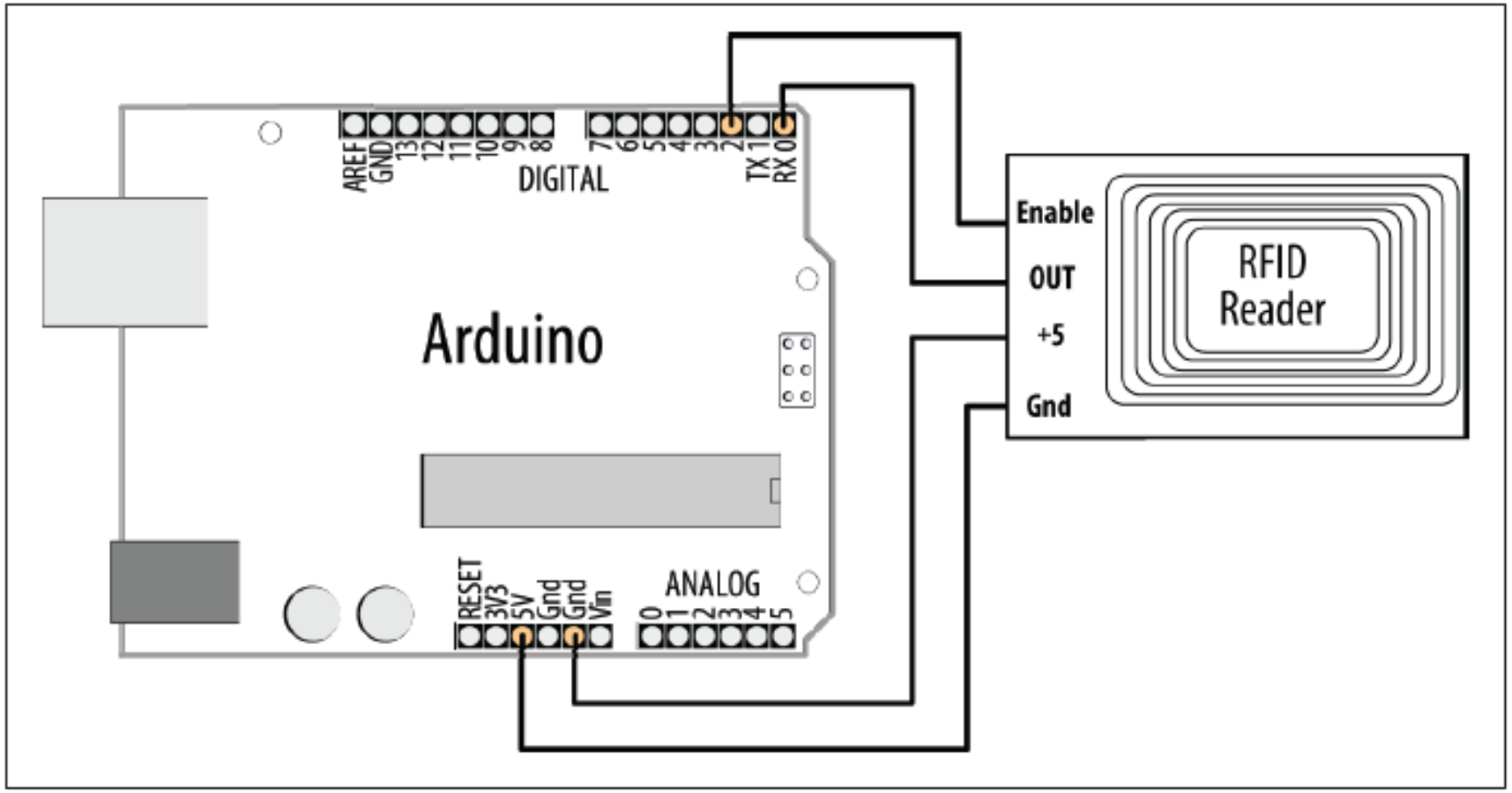


Figure 6-13. Serial RFID reader connected to Arduino

Reading RFID Tags:

The sketch reads and displays the value of an RFID tag:

```
/*
 RFID sketch
 Displays the value read from an RFID tag
 */

const int startByte   = 10;  // ASCII line feed precedes each tag
const int endByte     = 13;  // ASCII carriage return terminates each tag
const int tagLength   = 10;  // the number of digits in tag
const int totalLength = tagLength + 2; //tag length + start and end bytes
```

```
char tag[tagLength + 1];  // holds the tag and a terminating null

int bytesread = 0;

void setup()
{
  Serial.begin(2400);    // set this to the baud rate of your RFID reader
  pinMode(2,OUTPUT);     // connected to the RFID ENABLE pin
  digitalWrite(2, LOW);  // enable the RFID reader
}
```

Reading RFID Tags:

```
void loop()
{
  if(Serial.available() >= totalLength)  // check if there's enough data
  {
    if(Serial.read() == startByte)
    {
      bytesread = 0;                 // start of tag so reset count to 0
      while(bytesread < tagLength)  // read 10 digit code
      {
        int val = Serial.read();
        if((val == startByte)||(val == endByte))  // check for end of code
          break;
        tag[bytesread] = val;
         bytesread = bytesread + 1; // ready to read next digit
      }
      if( Serial.read() == endByte)  // check for the correct end character
      {
        tag[bytesread] = 0; // terminate the string
        Serial.print("RFID tag is: ");
        Serial.println(tag);
      }
    }
  }
}
```

Tracking Rotary Movement:

- **Problem:**
  - You want to measure and display the rotation of something to track its speed and/or direction.

- **Solution:**
  - **To sense rotary motion you can use a rotary encoder that is attached to the object you want to track. Connect the encoder as shown in Figure 6-14:**

- /*
- Read a rotary encoder
- This simple version polls the encoder pins
- The position is displayed on the Serial Monitor
- */

Tracking Rotary Movement:

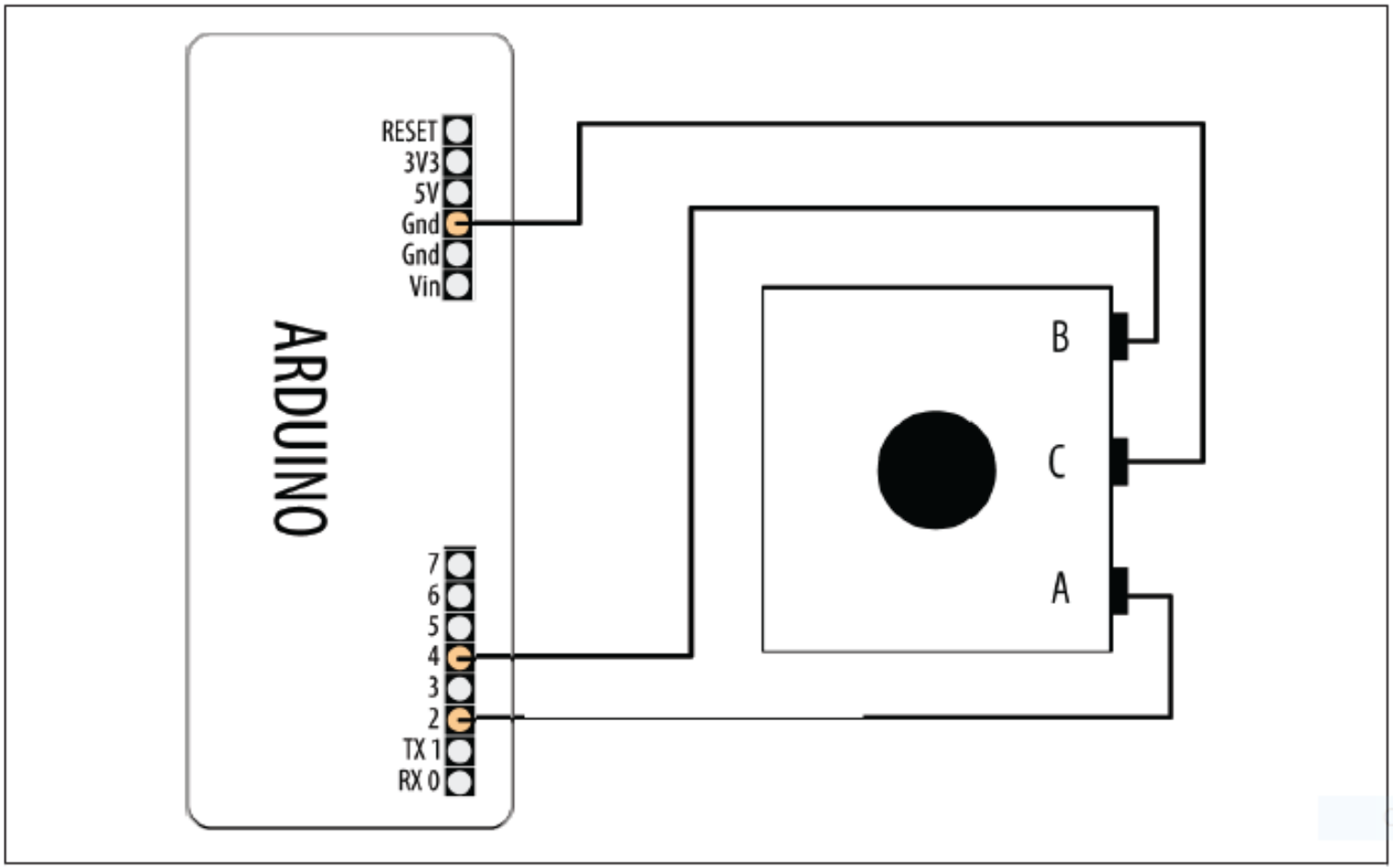
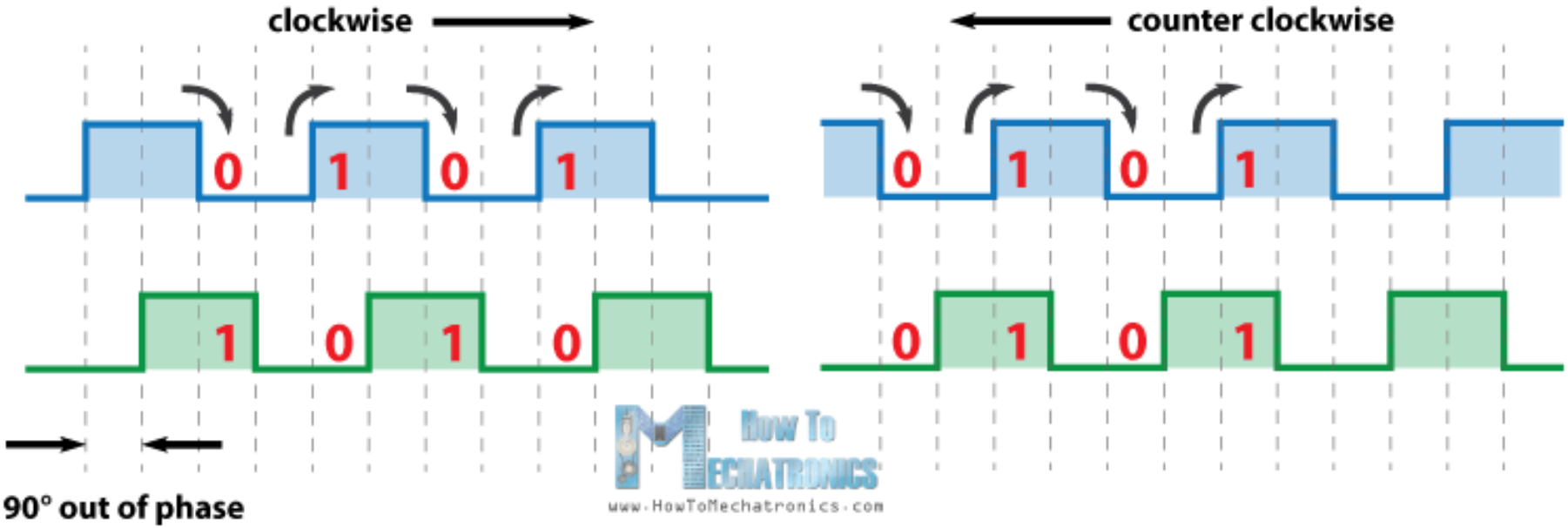


Figure 6-14. Rotary encoder

Tracking Rotary Movement:



https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/

Tracking Rotary Movement:

```
const int encoderPinA = 2;
const int encoderPinB = 4;
const int encoderStepsPerRevolution=16;
int angle = 0;

int val;

int encoderPos = 0;
boolean encoderALast = LOW;  // remembers the previous pin state


void setup()
{
  pinMode(encoderPinA, INPUT);
  pinMode(encoderPinB, INPUT);
  digitalWrite(encoderPinA, HIGH);
  digitalWrite(encoderPinB, HIGH);
  Serial.begin (9600);
}
```

Tracking Rotary Movement:

```
void loop()
{
  boolean encoderA = digitalRead(encoderPinA);

  if ((encoderALast == HIGH) && (encoderA == LOW))
  {
    if (digitalRead(encoderPinB) == LOW)
    {
      encoderPos--;
    }
    else
    {
      encoderPos++;
    }
    angle=(encoderPos % encoderStepsPerRevolution)*360/encoderStepsPerRevolution;
    Serial.print (encoderPos);
    Serial.print (" ");
    Serial.println (angle);
  }

  encoderALast = encoderA;
}
```

Tracking the Movement of More Than One Rotary Encoder:

- **Problem:**
  - You have two or more rotary encoders and you want to measure and display rotation.

- **Solution:**
  - **The circuit uses two encoders, connected as shown in Figure 6-15. You can read more about rotary encoders in Recipe 6.10:**

Tracking the Movement of More Than One Rotary Encoder:

```
/*
 RotaryEncoderMultiPoll
 This sketch has two encoders connected.
 One is connected to pins 2 and 3
 The other is connected to pins 4 and 5
 */

const int  ENCODERS = 2; // the number of encoders
```

```
const int encoderPinA[ENCODERS] = {2,4};     // encoderA pins on 2 and 4
const int encoderPinB[ENCODERS] = {3,5};     // encoderB pins on 3 and 5
int encoderPos[ ENCODERS] = { 0,0};          // initialize the positions to 0
boolean encoderALast[ENCODERS] = { LOW,LOW}; // holds last state of encoderA pin
```

Tracking the Movement of More Than One Rotary Encoder:

```
void setup()
{
  for (int i=2; i<6; i++){
    pinMode(i,INPUT);
    digitalWrite(i, HIGH);
  }
  Serial.begin (9600);
}
```

```
void loop()
{
  for(int i=0; i < ENCODERS;i++)
  {
    updatePosition(i);
  }
}
```

Tracking the Movement of More Than One Rotary Encoder:

```
int updatePosition( int encoderIndex)
{
  boolean encoderA = digitalRead(encoderPinA[encoderIndex]);
  if ((encoderALast[encoderIndex] == HIGH) && (encoderA == LOW))
  {
    if (digitalRead(encoderPinB[encoderIndex]) == LOW)
    {
      encoderPos[encoderIndex]--;
    }
    else
    {
      encoderPos[encoderIndex]++;
    }
    Serial.print("Encoder ");
    Serial.print(encoderIndex,DEC);
    Serial.print("=");
    Serial.print (encoderPos[encoderIndex]);
    Serial.println ("/");
  }
  encoderALast[encoderIndex] = encoderA;
}
```

Tracking the Movement of More Than One Rotary Encoder:

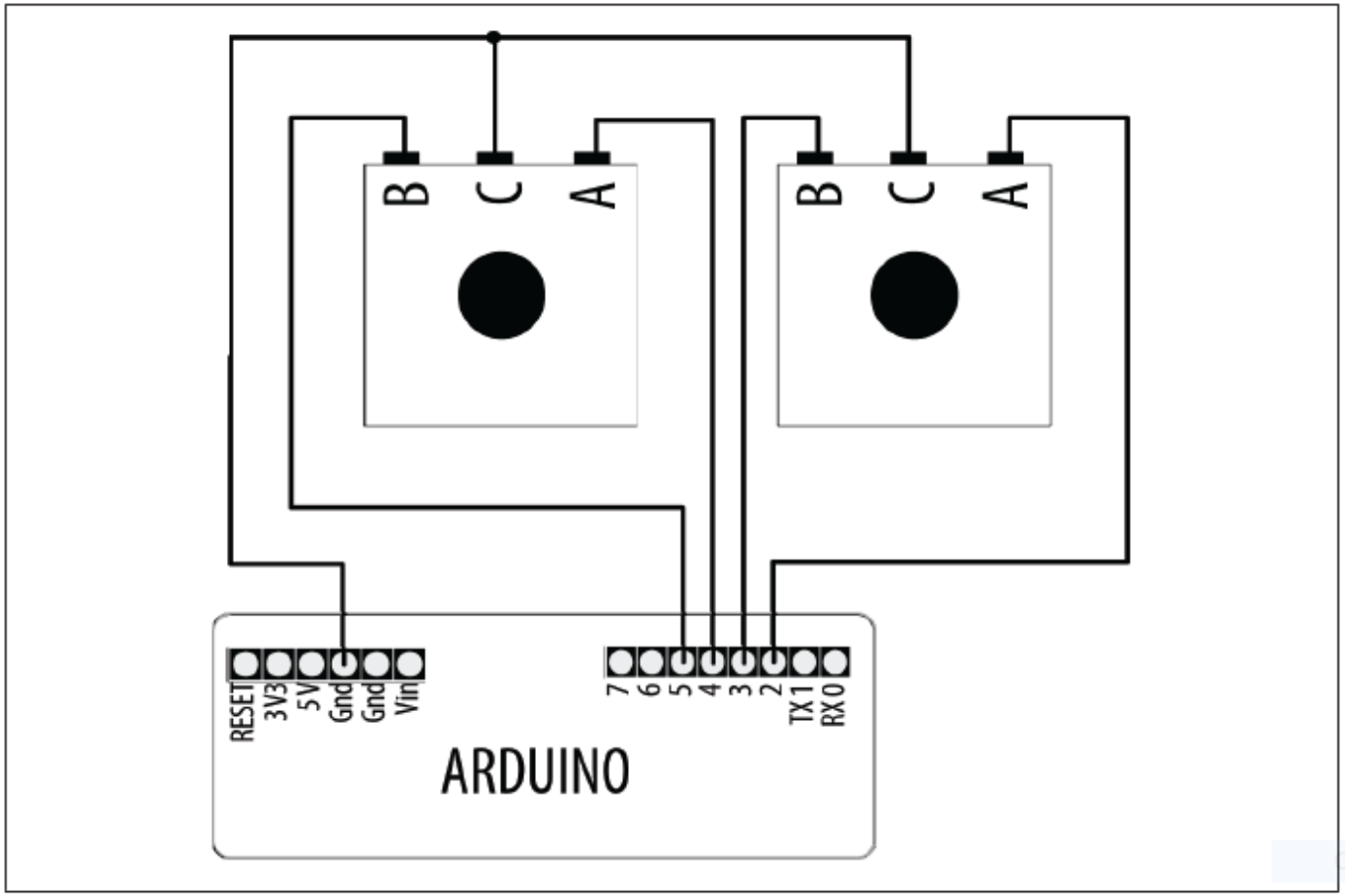


Figure 6-15. Connecting two rotary encoders

Tracking Rotary Movement in a Busy Sketch:

- **Problem:**
  - As you extend your code and it is doing other things in addition to reading the encoder, reading the encoder starts to get unreliable. This problem is particularly bad if the shaft rotates quickly.

- **Solution:**
  - **The circuit is the same as the one for Recipe 6.11. We will use an interrupt on the Arduino to make sure that every time a step happens, the code responds to it:**

Tracking Rotary Movement in a Busy Sketch:

```
/*
  RotaryEncoderInterrupt sketch
 */

const int encoderPinA = 2;
const int encoderPinB = 4;
int Pos, oldPos;
volatile int encoderPos = 0; // variables changed within interrupts are volatile


void setup()
{
  pinMode(encoderPinA, INPUT);
  pinMode(encoderPinB, INPUT);
  digitalWrite(encoderPinA, HIGH);
  digitalWrite(encoderPinB, HIGH);
  Serial.begin(9600);

  attachInterrupt(digitalPinToInterrupt(encoderPinA), doEncoder, FALLING);
}
```

Tracking Rotary Movement in a Busy Sketch:

attachInterrupt()

Description

Digital Pins With Interrupts

The first parameter to attachInterrupt is an interrupt number. Normally you should use digitalPinToInterrupt(pin) to translate the actual digital pin to the specific interrupt number. For example, if you connect to pin 3, use digitalPinToInterrupt(3) as the first parameter to attachInterrupt.

Board	Digital Pins Usable For Interrupts
Uno, Nano, Mini, other 328-based	2, 3
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	all digital pins, except 4
MKR1000 Rev.1	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Due	all digital pins
101	all digital pins

Tracking Rotary Movement in a Busy Sketch:

mode:	defines when the interrupt should be triggered. Four constants are predefined as valid values: - LOW to trigger the interrupt whenever the pin is low, - CHANGE to trigger the interrupt whenever the pin changes value	- RISING to trigger when the pin goes from low to high, - FALLING for when the pin goes from high to low.

Tracking Rotary Movement in a Busy Sketch:

```
void loop()
{
  uint8_t oldSREG = SREG;

  cli();
  Pos = encoderPos;
  SREG = oldSREG;
  if(Pos != oldPos)
  {
    Serial.println(Pos,DEC);
    oldPos = Pos;
  }
   delay(1000);
}


void doEncoder()
{
  if (digitalRead(encoderPinA) == digitalRead(encoderPinB))
```

Tracking Rotary Movement in a Busy Sketch:

```
      encoderPos++;    // count up if both encoder pins are the same
    else
      encoderPos--;    // count down if pins are different
  }
```

This code will only report the Pos value on the serial port, at most once every second (because of the delay), but the values reported will take into account any movement that may have happened while it was delaying.

Tracking Rotary Movement in a Busy Sketch:

To read this variable in the main loop, you should take special precautions to make sure the interrupt does not happen in the middle of reading it. This chunk of code does that:

```
uint8_t oldSREG = SREG;

  cli();
  Pos = encoderPos;
  SREG = oldSREG;
```

First you save the state of SREG (the interrupt registers), and then `cli` turns the interrupt off. The value is read, and then restoring SREG turns the interrupt back on and sets everything back as it was. Any interrupt that occurs when interrupts are turned off will wait until interrupts are turned back on. This period is so short that interrupts will not be missed (as long as you keep the code in the interrupt handler as short as possible).

Using a Mouse:

- **Problem:**
  - You want to detect movements of a PS/2-compatible mouse and respond to changes in the *x* and *y* coordinates.

- **Solution:**
  - **This solution uses LEDs to indicate mouse movement. The brightness of the LEDs changes in response to mouse movement in the *x* (left and right) and *y* (nearer and farther) directions. Clicking the mouse buttons sets the current position as the reference point (Figure 6-16 shows the connections):**

Using a Mouse:

```
/*
   Mouse
   an arduino sketch using ps2 mouse library
   see: http://www.arduino.cc/playground/ComponentLib/Ps2mouse
 */

// PS2 mouse library from :  http://www.arduino.cc/playground/ComponentLib/Ps2mouse
#define WProgram.h Arduino.h
#include <ps2.h>

const int dataPin =  5;
const int clockPin = 6;

const int xLedPin  = 9;
const int yLedPin  = 11;

const int mouseRange = 255;  // the maximum range of x/y values
```

Using a Mouse:

```
char x;                  // values read from the mouse
char y;
byte status;

int xPosition = 0;       // values incremented and decremented when mouse moves
int yPosition = 0;
int xBrightness = 128;   // values increased and decreased based on mouse position
int yBrightness = 128;

const byte REQUEST_DATA = 0xeb; // command to get data from the mouse

PS2 mouse(clockPin, dataPin);

void setup()
{
  mouseBegin();
}
```

Using a Mouse:

```
void loop()
{
  // get a reading from the mouse
  mouse.write(REQUEST_DATA); // ask the mouse for data
  mouse.read();       // ignore ack
  status = mouse.read(); // read the mouse buttons
  if(status & 1) // this bit is set if the left mouse btn pressed
    xPosition = 0; // center the mouse x position
  if(status & 2) // this bit is set if the right mouse btn pressed
    yPosition = 0; // center the mouse y position

  x = mouse.read();
  y = mouse.read();
  if( x != 0 || y != 0)
  {
    // here if there is mouse movement

    xPosition = xPosition + x; // accumulate the position
    xPosition = constrain(xPosition,-mouseRange,mouseRange);
```

Using a Mouse:

```
    xPosition = xPosition + x; // accumulate the position
    xPosition = constrain(xPosition,-mouseRange,mouseRange);

    xBrightness = map(xPosition, -mouseRange, mouseRange, 0,255);
    analogWrite(xLedPin, xBrightness);

    yPosition = constrain(yPosition + y, -mouseRange,mouseRange);
    yBrightness = map(yPosition, -mouseRange, mouseRange, 0,255);
    analogWrite(yLedPin, yBrightness);
  }
}
```

Using a Mouse:

```
void mouseBegin()
{
  // reset and initialize the mouse
  mouse.write(0xff);          // reset
  delayMicroseconds(100);
  mouse.read();               // ack byte
  mouse.read();               // blank
  mouse.read();               // blank

  mouse.write(0xf0);          // remote mode
  mouse.read();               // ack
  delayMicroseconds(100);
}
```

Using a Mouse:

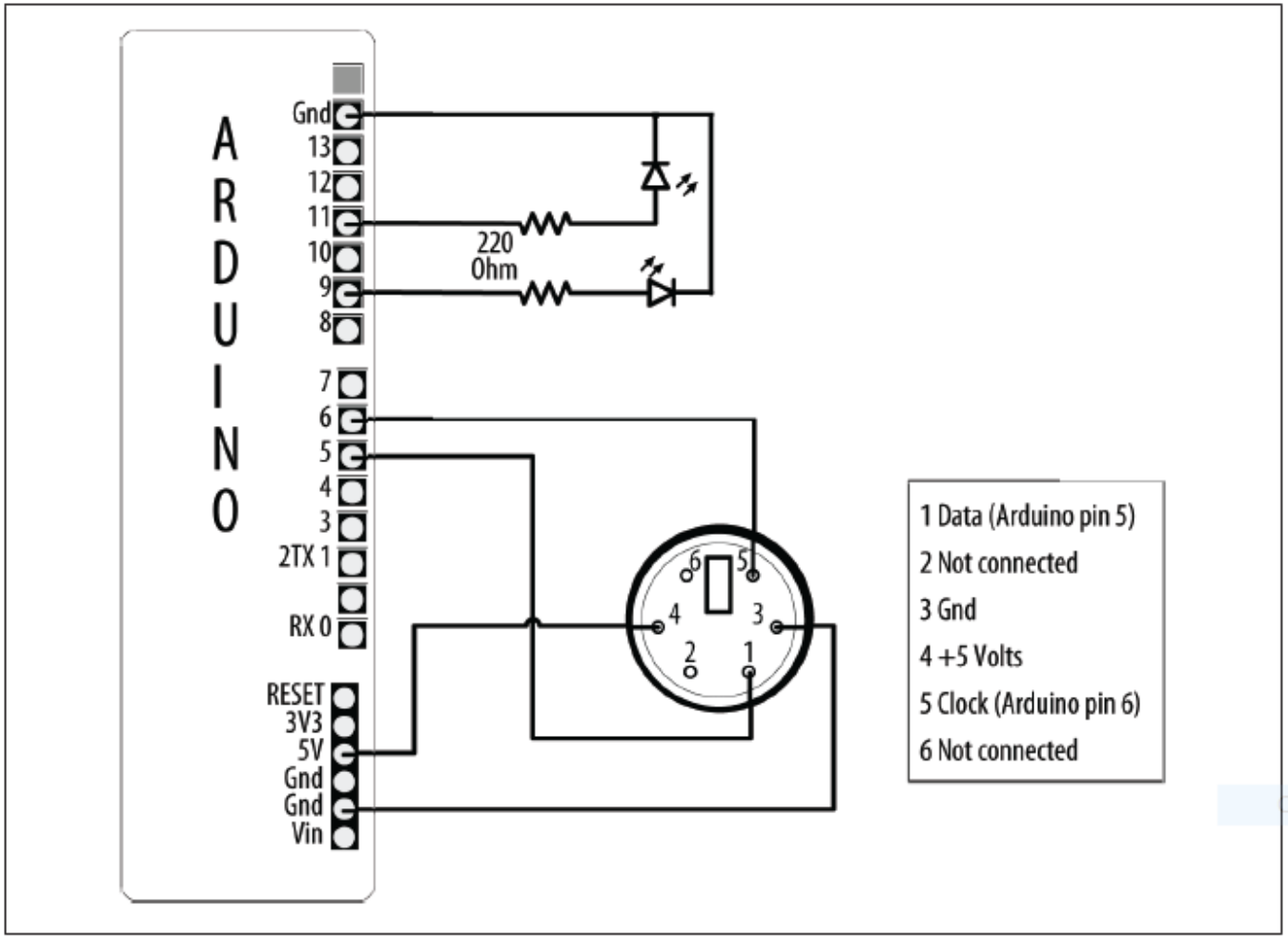


Figure 6-16. Connecting a mouse to indicate position and light LEDs