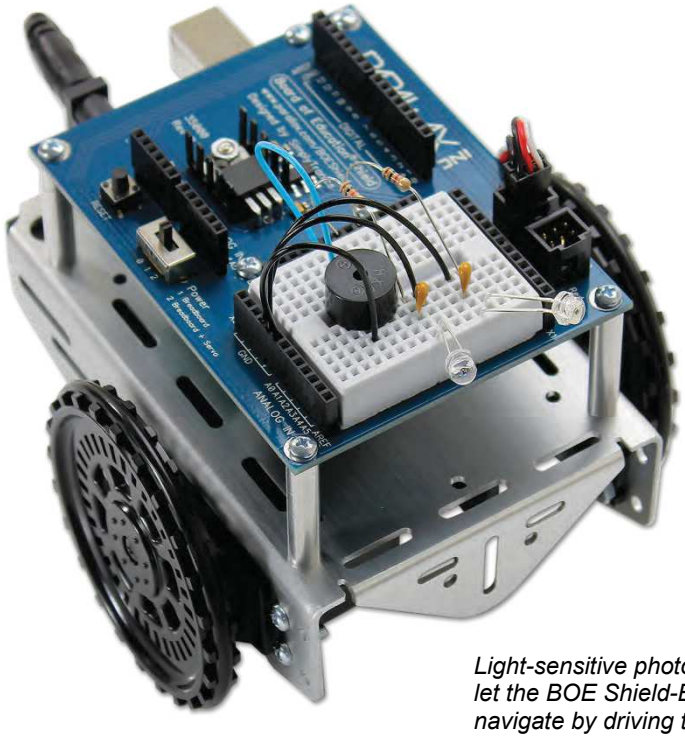
Chapter 6. Light-Sensitive Navigation with Phototransistors

Light sensors have many applications in robotics and industrial control: finding the edge of a roll of fabric in a textile mill, determining when to activate streetlights at different times of the year, when to take a picture, or when to deliver water to a crop of plants.

The light sensors in your Robotics Shield Kit respond to visible light, and also to an invisible type of light called *infrared*. These sensors can be used in different circuits that the Arduino can monitor to detect variations in light level. With this information, your sketch can be expanded to make the BOE Shield-Bot navigate by light, such as driving toward a flashlight beam or an open doorway letting light into a dark room.

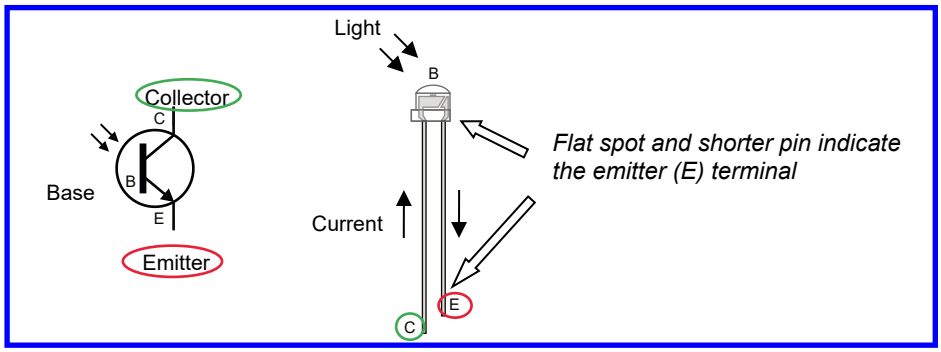


Light-sensitive phototransistors let the BOE Shield-Bot navigate by driving towards a bright light source.

Introducing the Phototransistor

A *transistor* is like a valve that regulates the amount of electric current that passes through two of its three terminals. The third terminal controls just how much current passes through the other two. Depending on the type of transistor, the current flow can be controlled by voltage, current, or in the case of the *phototransistor*, by light.

The drawing below shows the schematic and part drawing of the phototransistor in your Robotics Shield Kit. The brightness of the light shining on the phototransistor's base (B) terminal determines how much current it will allow to pass into its collector (C) terminal, and out through its emitter (E) terminal. Brighter light results in more current; less-bright light results in less current.

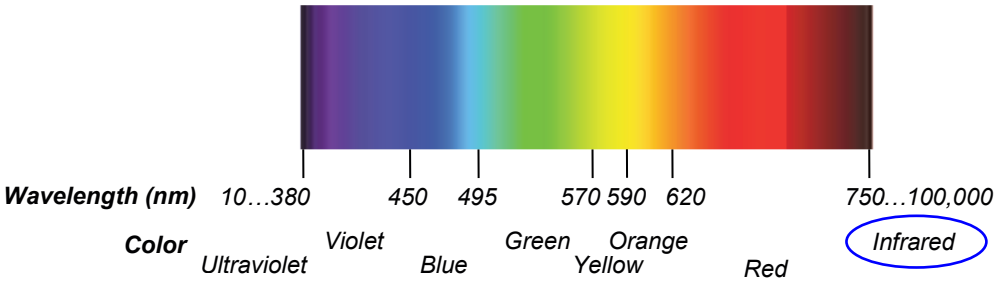


The phototransistor looks a little bit like an LED. The two devices do have two similarities. First, if you connect the phototransistor in the circuit backwards, it won't work right. Second, it also has two different length pins and a flat spot on its plastic case for identifying its terminals. The longer of the two pins indicates the phototransistor's collector terminal. The shorter pin indicates the emitter, and it connects closer to a flat spot on the phototransistor's clear plastic case.

Light Waves

In the ocean, you can measure the distance between the peaks of two adjacent waves in feet or meters. With light, which also travels in waves, the distance between adjacent peaks is measured in *nanometers* (nm) which are billionths of meters. The figure below shows the wavelengths for colors of light we are familiar with, along with some the human eye cannot detect, such as ultraviolet and infrared.

You can see this image in color in the free PDF version of this book, availabe for download from the #122-32335 product page at www.parallax.com.



The phototransistor in the Robotics Shield Kit is most sensitive to 850 nm wavelengths, which is in the infrared range. Infrared light is not visible to the human eye, but many different light sources emit considerable amounts of it, including halogen and incandescent lamps, and especially the sun. This phototransistor also responds to visible light, though it's less sensitive, especially to wavelengths below 450 nm.

The phototransistor circuits in this chapter are designed to work well indoors, with fluorescent or incandescent lighting. Make sure to avoid direct sunlight and direct halogen lights; they would flood the phototransistors with too much infrared light.

In your robotics area, close window blinds to block direct sunlight, and point any halogen lamps upward so that the light is reflected off the ceiling.

Activity 1: Simple Light to Voltage Sensor

Imagine that your BOE Shield-Bot is navigating a course, and there's a bright light at the end. Your robot's final task in the course is to stop underneath that bright light. There's a simple phototransistor circuit you can use that lets the Arduino know it detected bright light with a binary-1, or ambient light with a binary-0. Incandescent bulbs in desk lamps and flashlights make the best bright-light sources. Compact fluorescent and LED light sources are not as easy for the circuit in this activity to recognize.

Ambient means 'existing or present on all sides' according to Merriam Webster's dictionary. For the light level in a room, think about ambient light as the overall level of brightness.

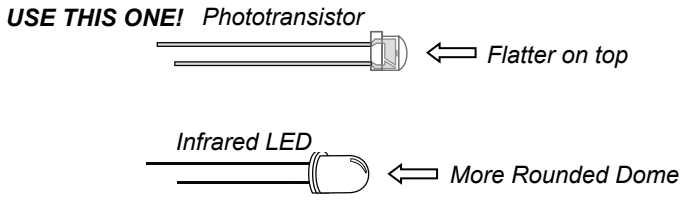
Parts List

(1) phototransistor
(2) jumper wires
(1) resistor, 2 kΩ (red-black-red)
(1) incandescent or fluorescent flashlight or desk lamp

After some testing, and depending on the light conditions in your robotics area, you might end up replacing the 2 kΩ resistor with one of these resistors, so keep them handy:

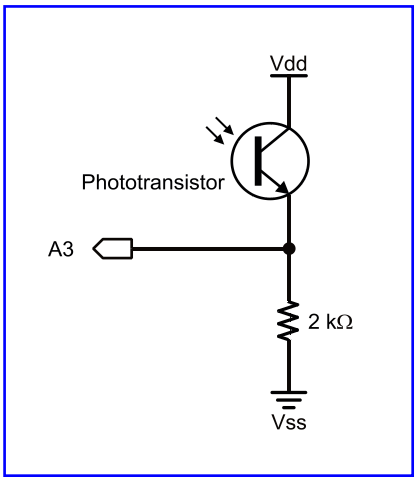
(1) resistor, 220 Ω (red-red-brown)
(1) resistor, 470 Ω (yellow-violet-brown)
(1) resistor, 1 kΩ (brown-black-red)
(1) resistor, 4.7 kΩ (yellow-violet-red)
(1) resistor, 10 kΩ (brown-black-orange)

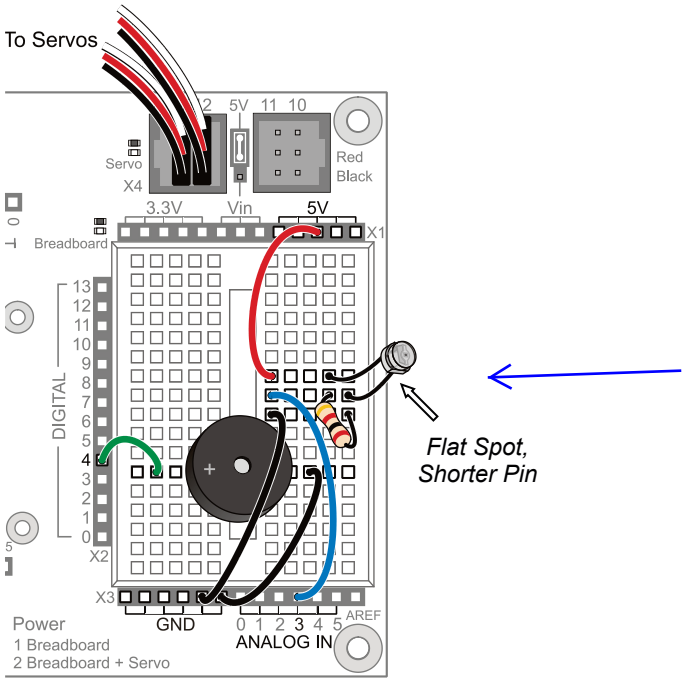
The next drawing will help you tell apart the phototransistor and infrared LED, since they look similar.



Building the Bright Light Detector

✓ The schematic and wiring diagram below show a circuit very similar to the ones in streetlights that turn on automatically at night. The circuit outputs a voltage that varies depending on how much light shines on the phototransistor. The Arduino will monitor the voltage level with one of its analog input pins.

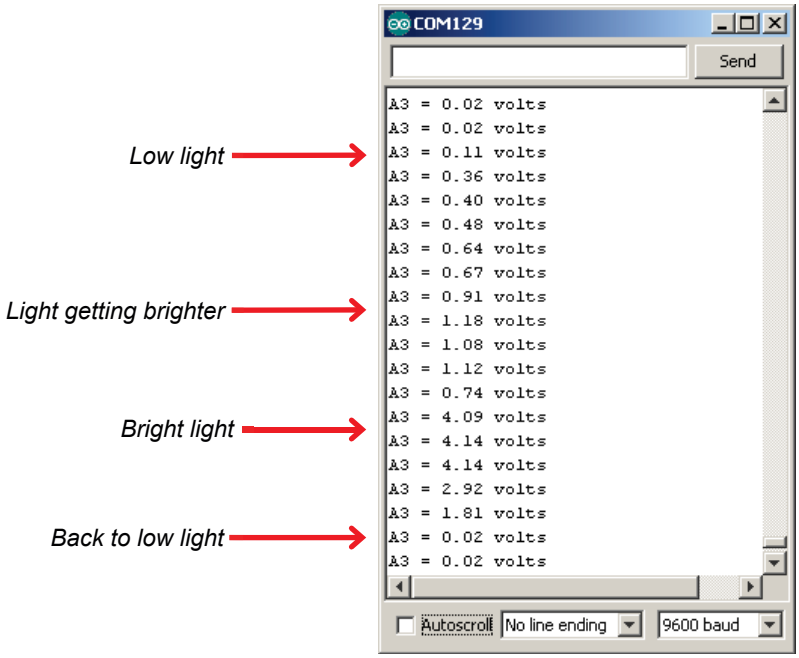




- ✓ Disconnect the battery pack and programming cable from your Arduino, and set the BOE Shield's switch to 0.
- ✓ Remove the whisker circuits, but leave the piezospeaker circuit in place.
- ✓ Build the circuit shown, using the 2 kΩ resistor.
- ✓ Double-check to make sure you connect the phototransistor's emitter lead (by the flat spot) to the resistor, and its collector to 5V.
- ✓ Also double-check that the phototransistor's leads are not touching each other.

Example Sketch: PhototransistorVoltage

The PhototransistorVoltage sketch makes the Serial Monitor display the voltage measured at A3—one of the Arduino's five analog input channels that are accessible through the BOE Shield. In the circuit you just built, a wire connects A3 to the row where the phototransistor's emitter and resistor meet. The voltage at this part of the circuit will change as the light level sensed by the phototransistor changes. The Serial Monitor screen capture below shows some example voltage measurements.



- ✓ Reconnect programming cable and battery pack power to your board.
- ✓ Put the BOE Shield's power switch in position 1.
- ✓ Create and save the PhototransistorVoltage sketch, and run it on your Arduino.
- ✓ Slowly move the flashlight or lamp over the phototransistor, and watch the value of A3 in the Serial Monitor. Brighter light should cause larger voltage values and dimmer light should cause smaller voltages.
- ✓ If the ambient light is brighter than just fluorescent lights, and you have a bright flashlight, you may need to replace the 2 kΩ resistor with a smaller value. Try 1 kΩ, 470 Ω, or even 220 Ω for really bright lights.
- ✓ If the ambient light is low, and you are using a fluorescent desk lamp bulb or an LED flashlight for your bright light, you may need to change the 2 kΩ resistor to 4.7 kΩ, or even 10 kΩ.
- ✓ Record values for ambient light (your normal room light levels), and then bright light, like when you shine a flashlight right on the phototransistor. Make a note of them—you'll need them for the sketch after this one.

```
/*
 * Robotics with the BOE Shield - PhototransistorVoltage
 * Display voltage of phototransistor circuit output connected to A3 in
 * the serial monitor.
 */
```

```
void setup()                              // Built-in initialization block
{
  Serial.begin(9600);                     // Set data rate to 9600 bps
}

void loop()                               // Main loop auto-repeats
{
  Serial.print("A3 = ");                  // Display "A3 = "
  Serial.print(volts(A3));                // Display measured A3 volts
  Serial.println(" volts");               // Display " volts" & newline
  delay(1000);                            // Delay for 1 second
}

float volts(int adPin)                    // Measures volts at adPin
{                                         // Returns floating point voltage
 return float(analogRead(adPin)) * 5.0 / 1024.0;
}
```

Halt Under the Bright Light

The sketch HaltUnderBrightLight will make the BOE Shield-Bot go forward until the phototransistor detects light that's bright enough to make the voltage applied to A3 exceed 3.5 V. You can change the 3.5 V value to one that's halfway between the high and low voltage values you recorded from the last sketch.

- ✓ Calculate the half-way point between the ambient and bright light voltages you recorded from the last sketch.
- ✓ In the HaltUnderBrightLight sketch, substitute your half way point value in place of 3.5 in the statement **`if(volts(A3) > 3.5`**.
- ✓ Run your modified version of HaltUnderBrightLight on the Arduino.
- ✓ Hold your flashlight or lamp about a foot off of the floor, and put the BOE Shield-Bot on the floor a couple feet away but pointed so it will go straight under the light.
- ✓ Move the power switch to position 2 so the BOE Shield-Bot will drive forward. How close did it get to stopping directly under the light?
- ✓ Try making adjustments to the threshold you set in the **`if(volts(A3) >…)`** statement to get the BOE Shield-Bot to park right underneath that bright light.

```
/*
 * Robotics with the BOE Shield - HaltUnderBrightLight
 * Display voltage of phototransistor circuit output connected to A3 in
 * the serial monitor.
 */
```

```
#include <Servo.h>                      // Include servo library

Servo servoLeft;                        // Declare left and right servos
Servo servoRight;

void setup()                            // Built-in initialization block
{
  tone(4, 3000, 1000);                  // Play tone for 1 second
  delay(1000);                          // Delay to finish tone

  servoLeft.attach(13);                 // Attach left signal to pin 13
  servoRight.attach(12);                // Attach right signal to pin 12

  servoLeft.writeMicroseconds(1700);    // Full speed forward
  servoRight.writeMicroseconds(1300);
}

void loop()                             // Main loop auto-repeats
{
  if(volts(A3) > 3.5)                   // If A3 voltage greater than 3.5
  {
    servoLeft.detach();                 // Stop servo signals
    servoRight.detach();
  }
}

float volts(int adPin)                  // Measures volts at adPin
{                                       // Returns floating point voltage
 return float(analogRead(adPin)) * 5.0 / 1024.0;
}
```

How the Volts Function Works

The Arduino's A0, A1…A5 sockets are connected to Atmel microcontroller pins that are configured for *analog to digital conversion*. It's how microcontrollers measure voltage: they split a voltage range into many numbers, with each number representing a voltage. Each of the Arduino's analog inputs has a 10-bit *resolution*, meaning that it uses 10 binary digits to describe its voltage measurement. With 10 binary digits, you can count from 0 to 1023; that's a total of 1024 voltage levels if you include zero.

By default, the Arduino's **`analogRead`** function is configured to use the 0…1023 values to describe where a voltage measurement falls in a 5 V scale. If you split 5 V into 1024 different levels, each level is 5/1024^{ths} of a volt apart. 5/1024^{ths} of a volt is approximately 0.004882813 V, or about 4.89 thousandths of a volt. So, to convert a value returned by **`analogRead`** to a voltmeter-style value, all the volts function has to do is multiply by 5 and divide by 1024.

Example: the **analogRead** function returns 645; how many volts is that? Answer:

$$V = 645 \times \frac{5V}{1024}$$
$$= 3.1494140625\ V$$
$$\approx 3.15\ V$$

The sketches have been calling the **volts** function with **volts(A3)**. When they do that, they pass **A3** to its ***adPin*** parameter. Inside the function, **analogRead(*adPin*)** becomes **analogRead(A3)**. It returns a value in the 0 to 1023 range that represents the voltage applied to A3. The **analogRead** call returns an integer, but since it is nested in **float(analogRead(*adPin*)**, that integer value gets converted to floating point. Then, it's multiplied by the floating point value 5.0 and divided by 1024.0, which converts it to a voltmeter value (just like we converted 645 to 3.15 V).

```
float volts(int adPin)                  // Measures volts at adPin
{                                       // Returns floating point voltage
 return float(analogRead(adPin)) * 5.0 / 1024.0;
}
```

Since **return** is to the left of the calculation in the **volts** function block, the result gets returned to the function call. The sketch PhototransistorVoltage displays the value returned by the **volts** function with **Serial.print(volts(A3))**.

HaltUnderBrightLight uses that value in the **if(volts(A3) > 3.5)** expression to bring the BOE Shield-Bot to a halt under the light.

Binary vs. Analog and Digital

A binary sensor can transmit two different states, typically to indicate the presence or absence of something. For example, a whisker sends a high signal if it is not pressed, or a low signal if it is pressed.

An analog sensor sends a continuous range of values that correspond to a continuous range of measurements. The phototransistor circuits in this activity are examples of analog sensors. They provide continuous ranges of values that correspond to continuous ranges of light levels.

A *digital value* is a number expressed by digits. Computers and microcontrollers store analog measurements as digital values. The process of measuring an analog sensor and storing that measurement as a digital value is called analog to digital conversion. The measurement is called a digitized measurement. Analog to digital conversion documents will also call them quantized measurements.

The Arduino Map Function

In the PhototransistorVoltage sketch, we converted measurements from the 0 to 1023 range to the 0.0 to 4.995 volt range for display. For other applications, you might need to convert the value to some other range of integers so that your sketch can pass it to another function, maybe for motor control, or maybe for more analysis.

That's where the Arduino's **`map`** function comes in handy. It's useful for "mapping" a value in one range of integers to an equivalent value in some other range. For example, let's say you want to map a measurement in the 0 to 1024 range to a range of 1300 to 1700 for servo control. Here is an example of how you could use the **`map`** function to do it:

```
int adcVal = analogRead(A3);
int newAdcVal = map(adcVal, 0, 1023, 1300, 1700);
```

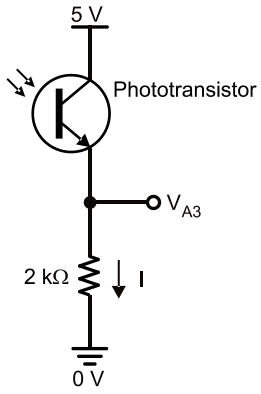
In this example, if the value of **`adcVal`** is 512 , the result of the **`map`** function for the **`newAdcVal`** call would be 1500 . So, the measurement got mapped from a point about half way through the 0..1023 range to its equivalent point in the 1300...1700 range.

How the Phototransistor Circuit Works

A resistor "resists" the flow of current. Voltage in a circuit with a resistor can be likened to water pressure. For a given amount of electric current, more voltage (pressure) is lost across a larger resistor than a smaller resistor that has the same amount of current passing through it. If you instead keep the resistance constant and vary the current, you can measure a larger voltage (pressure drop) across the same resistor with more current, or less voltage with less current.

The Arduino's analog inputs are invisible to the phototransistor circuit. So, A3 monitors the circuit but has no effect on it. Take a look at the circuit below. With 5 volts (5 V) at the top and GND (0 V) at the bottom of the circuit, 5 V of electrical pressure (voltage) makes the supply of electrons in the BOE Shield-Bot's batteries want to flow through it.

The reason the voltage at A3 (V_{A3}) changes with light is because the phototransistor lets more current pass when more light shines on it, or less current pass with less light. That current, which is labeled I in the circuit below, also has to pass through the resistor. When more current passes through a resistor, the voltage across it will be higher. When less current passes, the voltage will be lower. Since one end of the resistor is tied to GND = 0 V, the voltage at the V_{A3} end goes up with more current and down with less current.



If you replace the 2 kΩ resistor with a 1 kΩ resistor, V_{A3} will see smaller values for the same currents. In fact, it will take twice as much current to get V_{A3} to the same voltage level, which means the light will have to be twice as bright to reach the 3.5 V level, the default voltage in HaltUnderBrightLight to make the BOE Shield-Bot stop.

So, a smaller resistor in series with the phototransistor makes the circuit less sensitive to light. If you instead replace the 2 kΩ resistor with a 10 kΩ resistor, V_{A3} will be 5 times larger with the same current, and it'll only take $1/5^{th}$ the light to generate $1/5^{th}$ the current to get V_{A3} past the 3.5 V level. So, a larger resistor makes the circuit more sensitive to light.

Connected in Series: When two or more elements are connected end-to-end, they are *connected in series*. The phototransistor and resistor in this circuit are connected in series.

Ohm's Law

Two properties affect the voltage at V_{A3}: current and resistance, and Ohm's Law explains how it works. Ohm's Law states that the voltage (V) across a resistor is equal to the current (I) passing through it multiplied by its resistance (R). So, if you know two of these values, you can use the Ohm's Law equation to calculate the third:

$$\mathbf{V = I \times R}$$

E = I x R: In some textbooks, you will see E = I × R instead. E stands for electric potential, which is another way to say "volts."

Voltage (V) is measured in units of volts, which are abbreviated with an upper-case V. Current (I) is measured in amperes, or amps, which are abbreviated A. Resistance (R) is

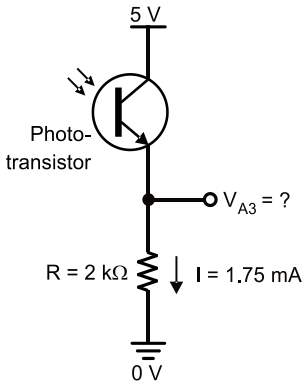
measured in ohms which is abbreviated with the Greek letter omega (Ω). The current levels you are likely to see through this circuit are in milliamps (mA). The lower-case m indicates that it's a measurement of thousandths of amps. Similarly, the lower-case k in kΩ indicates that the measurement is in thousands of ohms.

Let's use Ohm's Law to calculate V_{A3} in with the phototransistor, letting two different amounts of current flow through the circuit:

- 1.75 mA, which might happen as a result of fairly bright light
- 0.25 mA, which would happen with less bright light

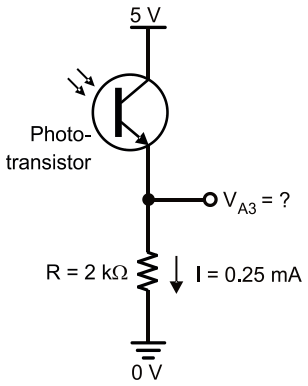
The examples below show the conditions and their solutions. When you try these calculations, remember that milli (m) is thousand*ths* and kilo (k) is thousand*s* when you substitute the numbers into Ohm's Law.

Example 1: I = 1.75 mA and R = 2 kΩ



$$
\begin{aligned}
V_{A3} &= I \times R \\
&= 1.75 mA \times 2 k\Omega \\
&= \frac{1.75}{1000} A \times 2000 \Omega \\
&= 1.75 A \times 2 \Omega \\
&= 3.5 A\Omega \\
&= 3.5\ V
\end{aligned}
$$

Example 2: 1 = 0.25 mA and R = 2 kΩ



$$
\begin{aligned}
V_{A3} &= I \times R \\
&= 0.25mA \times 2k\Omega \\
&= \frac{0.25}{1000} A \times 2000\Omega \\
&= 0.25A \times 2\Omega \\
&= 0.5A\Omega \\
&= 0.5\ V
\end{aligned}
$$

Your Turn – Ohm's Law and Resistor Adjustments

Let's say that the ambient light in your room is twice as bright as the light that resulted in V_{A3} = 3.5 V for bright light and 0.5 V for shade. Another situation that could cause higher current is if the ambient light is a stronger source of infrared. In either case, the phototransistor could allow twice as much current to flow through the circuit, which could lead to measurement difficulties.

- **Question**: What could you do to bring the circuit's voltage response back down to 3.5 V for bright light and 0.5 V for dim?
- **Answer**: Cut the resistor value in half; make it 1 kΩ instead of 2 kΩ.
- ✓ Try repeating the Ohm's Law calculations with R = 1 kΩ, bright current I = 3.5 mA, and dim current I = 0.5 mA. Does it bring V_{A3} back to 3.5 V for bright light and 0.5 V for dim light with twice the current? (It should; if it didn't for you, check your calculations.)

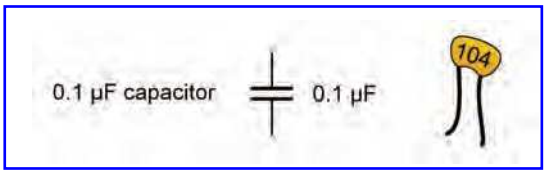
Activity 2: Measure Light Levels Over a Larger Range

The circuit in the previous activity only works over a limited light range. You might get the Activity #1 circuit all nice and calibrated in one room, then take it to a brighter room and find that all the voltage measurements will sit at the maximum value. Or, maybe you'll take it into a darker room, and the voltages will end up never making it past 0.1 V.

This activity introduces a different phototransistor circuit that the Arduino can use to measure a much wider range of light levels. This circuit and sketch can return values ranging from 0 to over 75,000. **Be aware: this time the smaller values indicate bright light, and large values indicate low light.**

Introducing the Capacitor

A *capacitor* is a device that stores charge, and it is a fundamental building block of many circuits. Batteries are also devices that store charge, and for these activities it will be convenient to think of capacitors as tiny batteries that can be charged, discharged, and recharged.



How much charge a capacitor can store is measured in farads (F). A *farad* is a very large value that's not practical for use with these BOE Shield-Bot circuits. The capacitors in your kit store fractions of millionths of farads. A millionth of a farad is called a *microfarad*, and it is abbreviated µF. This one stores one tenth of one millionth of a farad: 0.1 µF.

Common Capacitance Measurements

microfarads:	(millionths of a farad), abbreviated µF	1 µF = 1×10^{-6} F
nanofarads:	(billionths of a farad), abbreviated nF	1 nF = 1×10^{-9} F
picofarads:	(trillionths of a farad), abbreviated pF	1 pF = 1×10^{-12} F

The 104 on the 0.1 µF capacitor's case is a measurement in picofarads (pF). In this labeling system, 104 is the number 10 with four zeros added, so the capacitor is 100,000 pF, or 0.1 µF.

$(100{,}000) \times (1 \times 10^{-12})$ F = $(100 \times 10^{3}) \times (1 \times 10^{-12})$ F
$= 100 \times 10^{-9}$ F = 0.1×10^{-6} F
= 0.1 µF.

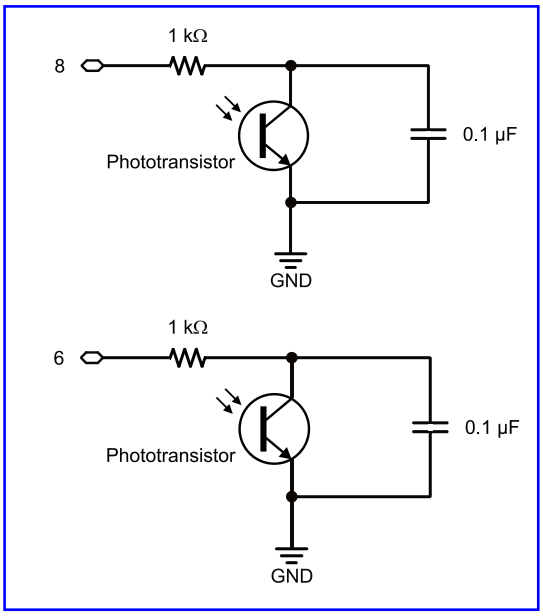
Building the Photosensitive Eyes

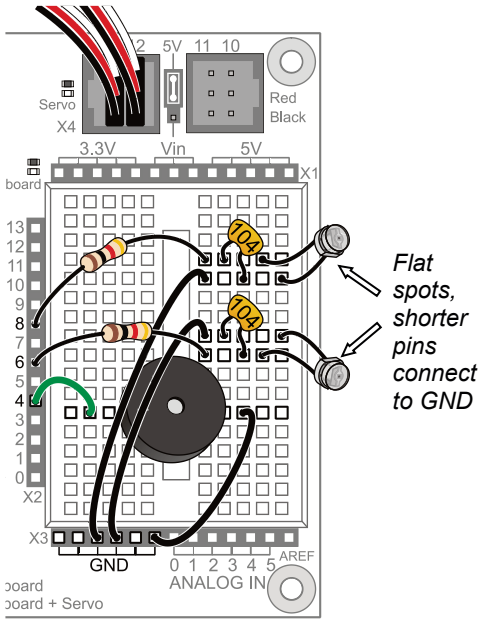
These circuits can respond independently to the light level reaching each phototransistor. They will be pointing upward at about 45°, one forward-left and the other forward-right. This way, a sketch monitoring the values of both phototransistors can determine which side of the BOE Shield-Bot sees brighter light. Then, this information can be used for navigation decisions.

Parts List

(2) phototransistors
(2) capacitors, 0.1 µF (104)
(2) resistors, 1 kΩ (brown-black-red)
(2) jumper wires

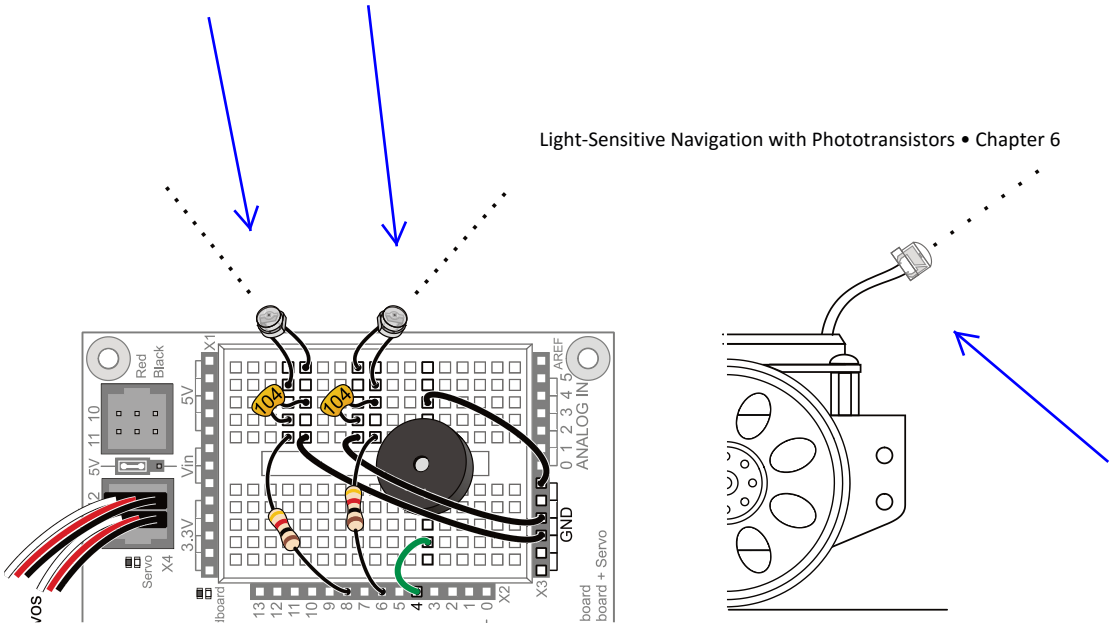
- ✓ Disconnect batteries and programming cable from your board.
- ✓ Remove the old phototransistor circuit, and build the circuits shown below.
- ✓ Double-check your circuits against the wiring diagram to make sure your phototransistors are not plugged in backwards, and that the leads are not touching.





The roaming examples in this chapter will depend on the phototransistors being pointed upward and outward to detect differences in light levels from different directions.

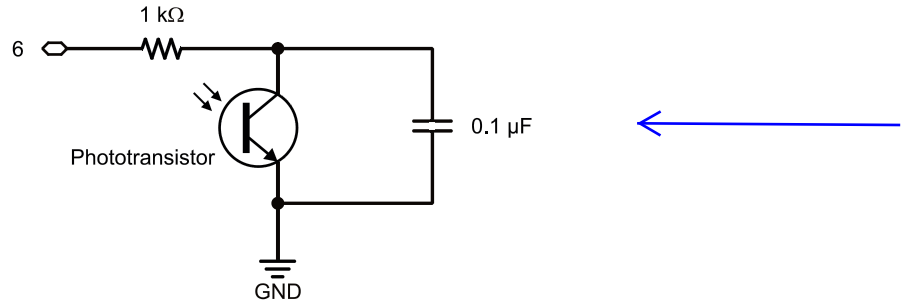
- ✓ Adjust the phototransistors to point upward at a 45° from the breadboard, and outward about 90° apart, as shown next.



About Charge Transfer and the Phototransistor Circuit

Think of each capacitor in this circuit as a tiny rechargeable battery, and think of each phototransistor as a light-controlled current valve. Each capacitor can be charged to 5 V and then allowed to drain through its phototransistor. The rate that the capacitor loses its charge depends on how much current the phototransistor (current valve) allows to pass, which in turn depends on the brightness of the light shining on the phototransistor's base. Again, brighter light results in more current passing. Shadows result in less current.

This kind of phototransistor/capacitor circuit is called a *charge transfer* circuit. The Arduino will determine the rate at which each capacitor loses its charge through its phototransistor by measuring how long it takes the capacitor's voltage to *decay*, that is, to drop below a certain voltage value. The decay time corresponds to how wide open that current valve is, which is controlled by the brightness of the light reaching the phototransistor's base. More light means faster decay, less light means slower decay.

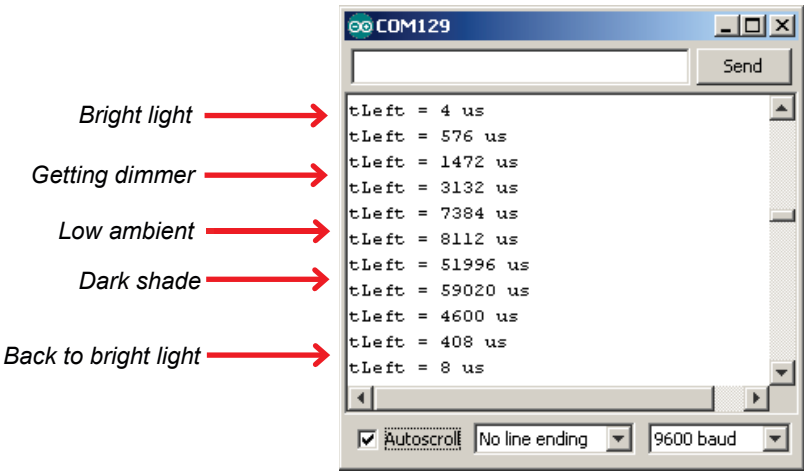


QT Circuit: A common abbreviation for charge transfer is QT. The letter Q refers to electrical charge (an accumulation of electrons), and T is for transfer.

Connected in Parallel: The phototransistor and capacitor shown above are connected in parallel; each of their leads are connected to common terminals (also called nodes). The phototransistor and the capacitor each have one lead connected to GND, and they also each have one lead connected to the same 1 kΩ resistor lead.

Test the Phototransistor Circuit

The sketch LeftLightSensor charges the capacitor in the pin 8 QT circuit, measures the voltage decay time, and displays it in the Serial Monitor. Remember, with this circuit and sketch, lower numbers mean brighter light.



We'll be using this light-sensing technique for the rest of the chapter, so you can take the BOE Shield-Bot from one room to another without having to worry about finding the right resistors for different ambient light levels.

- ✓ If there is direct sunlight shining in through the windows, close the blinds.
- ✓ Create, save, and run LeftLightSensor, and open the Serial Monitor.
- ✓ Make a note of the value displayed in the Serial Monitor.
- ✓ If the Serial Monitor does not display values or seems to get stuck after just one or two, it may mean that there's an error in your circuit. If you see these symptoms, check your wiring and try again.
- ✓ Use your hand or a book to cast a shadow over the pin 8 phototransistor circuit.
- ✓ Check the measurement in the Serial Monitor again. The value should be larger than the first one. Make a note of it too.

- ✓ If there's no output to the Serial Monitor, or if it is just stuck at one value regardless of light level, there could be a wiring error. Double-check your circuit (and your code too, if you hand-entered it.)
- ✓ Move the object casting the shadow closer to the top of the phototransistor to make the shadow darker. Make a note of the measurement.
- ✓ Experiment with progressively darker shadows, even cupping your hand over the phototransistor. (When it's really dark you may have to wait a few seconds for the measurement to finish.)

```
/*
 * Robotics with the BOE Shield - LeftLightSensor
 * Measures and displays microsecond decay time for left light sensor.
 */

void setup()                              // Built-in initialization block
{
  tone(4, 3000, 1000);                    // Play tone for 1 second
  delay(1000);                            // Delay to finish tone

  Serial.begin(9600);                     // Set data rate to 9600 bps
}

void loop()                               // Main loop auto-repeats
{
  long tLeft = rcTime(8);                 // Left rcTime -> tLeft

  Serial.print("tLeft = ");               // Display tLeft label
  Serial.print(tLeft);                    // Display tLeft value
  Serial.println(" us");                  // Display tLeft units + newline

  delay(1000);                            // 1 second delay
}
                                          // rcTime function at pin
long rcTime(int pin)                      // ..returns decay time
{
  pinMode(pin, OUTPUT);                   // Charge capacitor
  digitalWrite(pin, HIGH);                // ..by setting pin ouput-high
  delay(1);                               // ..for 5 ms
  pinMode(pin, INPUT);                    // Set pin to input
  digitalWrite(pin, LOW);                 // ..with no pullup
  long time  = micros();                  // Mark the time
  while(digitalRead(pin));                // Wait for voltage < threshold
  time = micros() - time;                 // Calculate decay time
  return time;                            // Return decay time
}
```

Your Turn: Test the Other Phototransistor Circuit

Before moving on to navigation, you'll need to run the same test on the right (pin 6) light sensor circuit. Both circuits have to be working well before you can move on to using them for navigation—there's that subsystem testing again!

- ✓ In the **`rcTime`** call, change the ***`pin`*** parameter from 8 to 6.
- ✓ Change all instances of **`tLeft`** to **`tRight`**.
- ✓ Run the sketch, and verify that the pin 6 light sensor circuit is working.
- ✓ It would also be nice to have a third sketch that tests both phototransistor circuits.
- ✓ Re-save the sketch as BothLightSensors, and update the comments.
- ✓ Replace the **`loop`** function with the one below.

Try rotating your BOE Shield-Bot until one side is pointing toward the brightest light source in the room and the other is pointing away from it. What is the largest difference you can get between **`tLeft`** and **`tRight`** in the Serial Monitor?

```
void loop()                             // Main loop auto-repeats
{
  long tLeft = rcTime(8);               // Left rcTime -> tLeft
  Serial.print("tLeft = ");             // Display tLeft label
  Serial.print(tLeft);                  // Display tLeft value
  Serial.print(" us      ");            // Display tLeft units

  long tRight = rcTime(6);              // Left rcTime -> tRight
  Serial.print("tRight = ");            // Display tRight label
  Serial.print(tRight);                 // Display tRight value
  Serial.println(" us");                // Display tRight units + newline

  delay(1000);                          // 1 second delay
}
```

rcTime and Voltage Decay

When light levels are low, the custom **`rcTime`** function might take time measurements too large for **`int`** or even **`word`** variables to store. The next step up in storage capacity is a **`long`** variable, which can store values from -2,147,483,648 to 2,147,483,647. So, the function definition **`long rcTime(int pin)`** is set up to make the function return a **`long`** value when it's done. It also needs to know which pin to measure.

```
long rcTime(int pin)
```

A charge transfer measurement takes seven steps:

(1) Set the I/O pin high to charge the capacitor.
(2) Wait long enough for the capacitor to charge.
(3) Change the I/O pin to input.
(4) Check the time.
(5) Wait for the voltage to decay and pass below the Arduino's 2.1 V threshold.
(6) Check the time again.
(7) Subtract the step-3 time from the step-6 time. That's the amount of time the decay took.

```
{
  pinMode(pin, OUTPUT);          // Step 1, part 1
  digitalWrite(pin, HIGH);       // Step 1, part 2
  delay(1);                      // Step 2
  pinMode(pin, INPUT);           // Step 3  part 1
  digitalWrite(pin, LOW);        // Step 3, part 2
  long time  = micros();         // Step 4
  while(digitalRead(pin));       // Step 5
  time = micros() - time;        // Step 6 & 7
  return time;
}
```

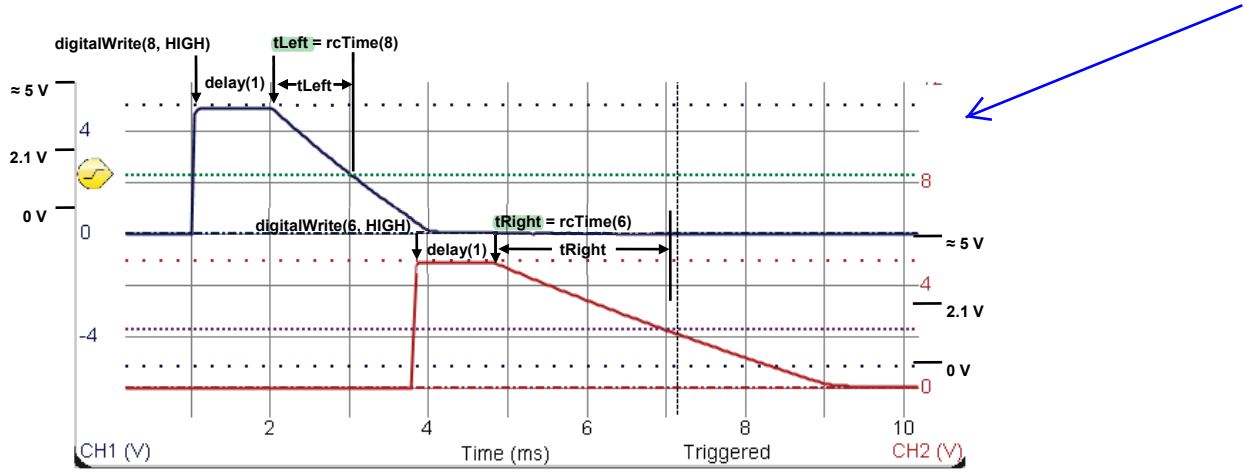
In this sketch, Step 1 has two sub-steps. First, **`pinMode(pin, OUPUT)`** sets the I/O pin to an output, then **`digitalWrite(pin, HIGH)`** makes it supply 5 V to the circuit.

Step 3 also has two sub-steps, because the I/O pin is sending a high signal. When the sketch changes the I/O pin's direction from output-high to input, it adds 10 kΩ of resistance to the circuit, which must be removed. Adding **`digitalWrite(pin, LOW)`** after **`pinMode(pin, INPUT)`** removes that resistance and allows the capacitor to drain its charge normally through the phototransistor.

Optional Advanced Topic: Voltage Decay Graphs

The next graph shows the BOE Shield-Bot's left and right QT circuit voltage responses while the BothLightSensors sketch is running. The device that measures and graphs these voltage responses over time is called an *oscilloscope*.

The two lines that graph the two voltage signals are called *traces*. The voltage scale for the upper trace is along the left, and the voltage scale for the lower trace is along the right. The time scale for both traces is along the bottom. Labels above each trace show when each command in BothLightSensors executes, so that you can see how the voltage signals respond.



The upper trace in the graph plots the capacitor's voltage in the pin 8 QT circuit; that's the left light sensor. In response to `digitalWrite(8, HIGH)`, the voltage quickly rises from 0 V to almost 5 V at about the 1 ms mark. The signal stays at around 5 V for the duration of `delay(1)`. Then, at the 2 ms mark, the `rcTime` call causes the decay to start.
The `rcTime function` measures the time it takes the voltage to decay to about 2.1 V and stores it in the `tLeft` variable. In the plot, it looks like that decay took about 1 ms, so the `tLeft` variable should store a value close to 1000.

The lower trace in the graph plots the pin 6 QT circuit's capacitor voltage—the right light sensor. This measurement starts after the left sensor measurement is done. The voltage varies in a manner similar to the upper trace, except the decay time takes about 2 ms. We would expect to see `tRight` store a value in the 2000 neighborhood. This larger value corresponds to a slower decay, which in turn corresponds to a lower light level.

Activity 3: Light Measurements for Roaming

We now have circuits that can work under a variety of lighting conditions. Now we need some code that can adapt as well. An example of sketch code that cannot adapt to change would be:

```
if(tLeft > 2500)…          // Not good for navigation.
```

Maybe that statement would work well for turning away from shadows in one room, but take it to another with brighter lights, and it might never detect a shadow. Or, take it to a darker room, and it might think it's seeing shadows all the time. For navigation, what matters is not an actual number reporting the light level over each sensor. What matters is the *difference* in how much light the two sensors detect, so the robot can turn toward the sensor seeing brighter light (or away from it, depending on what you want.)

The solution is simple. Just divide the right sensor measurement into the sum of both. Your result will always be in the 0 to 1 range. This technique is an example of a *normalized differential* measurement Here's what it looks like as an equation:

$$normalized\ differential\ shade = \frac{tRight}{tRight + tLeft}$$

For example, a normalized differential measurement of 0.25 would mean "the light is 1/2 as bright over the right sensor as it is over the left." The actual values for **`tRight`** and **`tLeft`** might be small in a bright room or large in a dark room, but the answer will still be 0.25 if the light is 1/2 as bright over the right sensor. A measurement of 0.5 would mean that the **`tRight`** and **`tLeft`** values are equal. They could both be large, or both be small, but if the result is 0.5, it means the sensors are detecting the same level of brightness.

Here's another trick: subtract 0.5 from the normalized differential shade measurement. That way, the results range from –0.5 to +0.5 instead of 0 to 1, and a measurement of 0 means equal brightness. The result is *a zero-justified* normalized differential shade measurement.

$$zero\ justified\ normalized\ differential\ shade = \frac{tRight}{tRight + tLeft} - 0.5$$

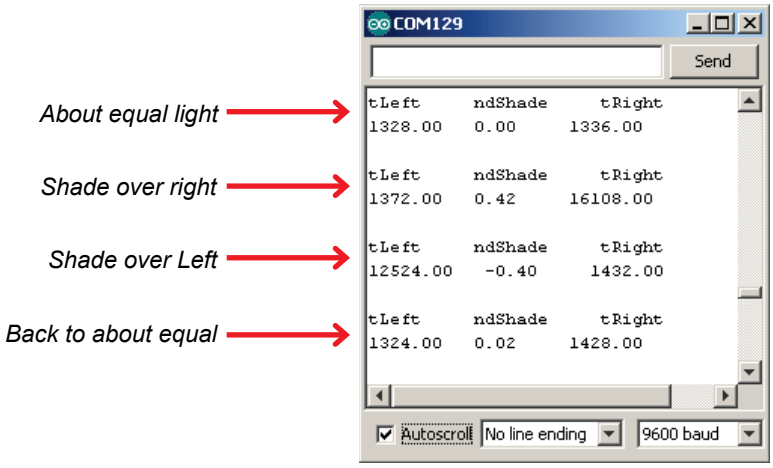
But why do it? The value range –0.5 to +0.5 is great for navigation sketches because the positive and negative values can be used to scale the wheels speeds. Here is how the zero-justified normalized differential shade equation appears in the next sketch:

```
float ndShade;                          // Normalized differential shade
ndShade = tRight / (tLeft + tRight) - 0.5; // Calculate & subtract 0.5
```

The final measurement will be stored in a floating point variable named **ndShade**, so that gets declared first. Then, the next line does the zero-justified normalized differential shade math. The result will be a value in the –0.5 to +0.5 range that represents the fraction of total shade that **tRight** detects, compared to **tLeft**. When **ndShade** is 0, it means **tRight** and **tLeft** are the same values, so the sensors are detecting equally bright light. The closer **ndShade** gets to –0.5, the darker the shade over the right sensor. The closer **ndShade** gets to 0.5 the darker the shade over the left sensor. This will be very useful for navigation. Let's test it first with the Serial Monitor.

Example Sketch: LightSensorValues

This screen capture shows a Serial Monitor example with the LightSensorValues sketch running. With shade over the right sensor, the **ndShade** value is about 0.4. With shade over the left sensor, it's about –0.4.

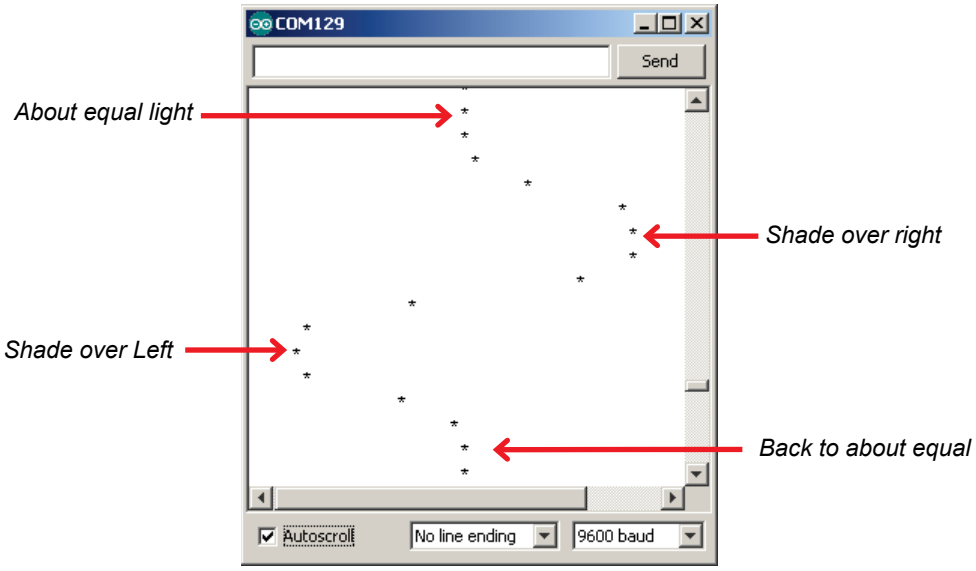


- ✓ Make sure there is no direct sunlight streaming in nearby windows. Indoor lighting is good, but direct sunlight will still flood the sensors.
- ✓ Verify that when you cast shade over the BOE Shield-Bot's left sensor, it results in negative values, with darker shade resulting in larger negative values.
- ✓ Verify that when you cast shade over the BOE Shield-Bot's right sensor, it results in positive values, with darker shade resulting in larger positive values.
- ✓ Verify that when both sensors see about the same level of light or shade, that **ndShade** reports values close to 0.
- ✓ Try casting equal shade over both sensors. Even though the overall light level dropped, the value of **ndShade** should still stay close to zero.

```
/*
 * Robotics with the BOE Shield - LightSensorValues
 * Displays tLeft, ndShade and tRight in the Serial Monitor.
 */

void setup()                              // Built-in initialization block
{
  tone(4, 3000, 1000);                    // Play tone for 1 second
  delay(1000);                            // Delay to finish tone

  Serial.begin(9600);                     // Set data rate to 9600 bps
}

void loop()                               // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));         // Get left light & make float
  float tRight = float(rcTime(6));        // Get right light & make float

  float ndShade;                          // Normalized differential shade
  ndShade = tRight / (tLeft + tRight) - 0.5; // Calculate & subtract 0.5

  // Display heading
  Serial.println("tLeft     ndShade     tRight");

  Serial.print(tLeft);                    // Display tLeft value
  Serial.print("   ");                    // Display spaces
  Serial.print(ndShade);                  // Display ndShade value
  Serial.print("     ");                  // Display more spaces
  Serial.println(tRight);                 // Display tRight value
  Serial.println(' ');                    // Add an extra newline

  delay(1000);                            // 1 second delay
}

long rcTime(int pin)                      // rcTime measures decay at pin
{
  pinMode(pin, OUTPUT);                   // Charge capacitor
  digitalWrite(pin, HIGH);                // ..by setting pin ouput-high
  delay(5);                               // ..for 5 ms
  pinMode(pin, INPUT);                    // Set pin to input
  digitalWrite(pin, LOW);                 // ..with no pullup
  long time  = micros();                  // Mark the time
  while(digitalRead(pin));                // Wait for voltage < threshold
  time = micros() - time;                 // Calculate decay time
  return time;                            // Returns decay time
}
```

Light Measurement Graphic Display

The Serial Monitor screen capture below shows an example of a graphical display of the **ndShade** variable. The asterisk will be in the center of the -0.5 to +0.5 scale if the light or shade is the same over both sensors. If the shade is darker over the BOE Shield-Bot's right sensor, the asterisk will position to the right in the scale. If it's darker over the left, the asterisk will position toward the left. A larger shade/light contrast (like darker shade over one of the sensors) will result in the asterisk positioning further from the center.



- ✓ Load the LightSensorDisplay sketch into the Arduino.
- ✓ Open the Serial Monitor and make sure Autoscroll is checked.

Try casting different levels of shade over each light sensor, and watch how the asterisk in the Serial Monitor responds. Remember that if you cast equal shade over both sensors, the asterisk should still be in the middle; it only indicates which sensor sees more shade if there's a difference between them.

```
/*
 * Robotics with the BOE Shield - LightSensorDisplay
 * Displays a scrolling graph of ndShade.  The asterisk positions ranges
 * from 0 to 40 with 20 (middle of the display) indicating same light on
 * both sides.
 */

void setup()                            // Built-in initialization block
{
```

```
  tone(4, 3000, 1000);                  // Play tone for 1 second
  delay(1000);                          // Delay to finish tone
  Serial.begin(9600);                   // Set data rate to 9600 bps
}

void loop()                             // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));       // Get left light & make float
  float tRight = float(rcTime(6));      // Get right light & make float

  float ndShade;                        // Normalized differential shade
  ndShade = tRight / (tLeft+tRight) - 0.5;   // Calculate & subtract 0.5

  for(int i = 0; i<(ndShade * 40) + 20; i++) // Place asterisk in 0 to 40
  {
    Serial.print(' ');                  // Pad (ndShade * 40) + 20 spaces
  }
  Serial.println('*');                  // Print asterisk and newline

  delay(100);                           // 0.1 second delay
}

long rcTime(int pin)                    // rcTime measures decay at pin
{
  pinMode(pin, OUTPUT);                 // Charge capacitor
  digitalWrite(pin, HIGH);              // ..by setting pin ouput-high
  delay(5);                             // ..for 5 ms
  pinMode(pin, INPUT);                  // Set pin to input
  digitalWrite(pin, LOW);               // ..with no pullup
  long time  = micros();                // Mark the time
  while(digitalRead(pin));              // Wait for voltage < threshold
  time = micros() - time;               // Calculate decay time
  return time;                          // Returns decay time
}
```

How LightSensorDisplay Works

The **`loop`** function starts by taking the two **`rcTime`** measurements for the left and right light sensors, and stores them in **`tLeft`** and **`tRight`**.

```
void loop()                             // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));       // Get left light & make float
  float tRight = float(rcTime(6));      // Get right light & make float
```

After declaring **`ndShade`** as a floating-point variable, **`tLeft`** and **`tRight`** are used in an expression to get that zero-justified normalized differential measurement. The result will be between –0.5 and +0.5, and gets stored in **`ndShade`**.

```
  float ndShade;                         // Normalized differential shade
```

```
  ndShade = tRight / (tLeft+tRight) - 0.5;    // Calculate & subtract 0.5
```

Next, this **for** loop places the cursor in the right place for printing an asterisk. Take a close look at the **for** loop's condition. It takes **ndShade** and multiples it by 40. It also has to add 20 to the value because if **ndShade** is –0.5, we want that to print with zero leading spaces. So (–0.5 × 40) + 20 = 0. Now, if **ndShade** is 0, we want it to print 20 spaces over: (0 × 40) + 20 = 20. If it's +0.5 we want it to print 40 spaces over: (0.5 × 40) + 20 = 40. Of course, if it's something in between, like 0.25, we have (0.25 × 40) + 20 = 30. So, it'll print half way between center and far right.

```
  for(int i = 0; i<(ndShade * 40) + 20; i++) // Place asterisk in 0 to 40
  {
    Serial.print(' ');                       // Pad (ndShade * 40) + 20 spaces
  }
```

After printing the spaces, a single asterisk prints on the line. Recall that **println** prints and also adds a newline so that the next time through the loop, the asterisk will display on the next line.

```
  Serial.println('*');                       // Print asterisk and newline

  delay(100);                                // 0.1 second delay
}
```

Activity 4: Test a Light-Roaming Routine

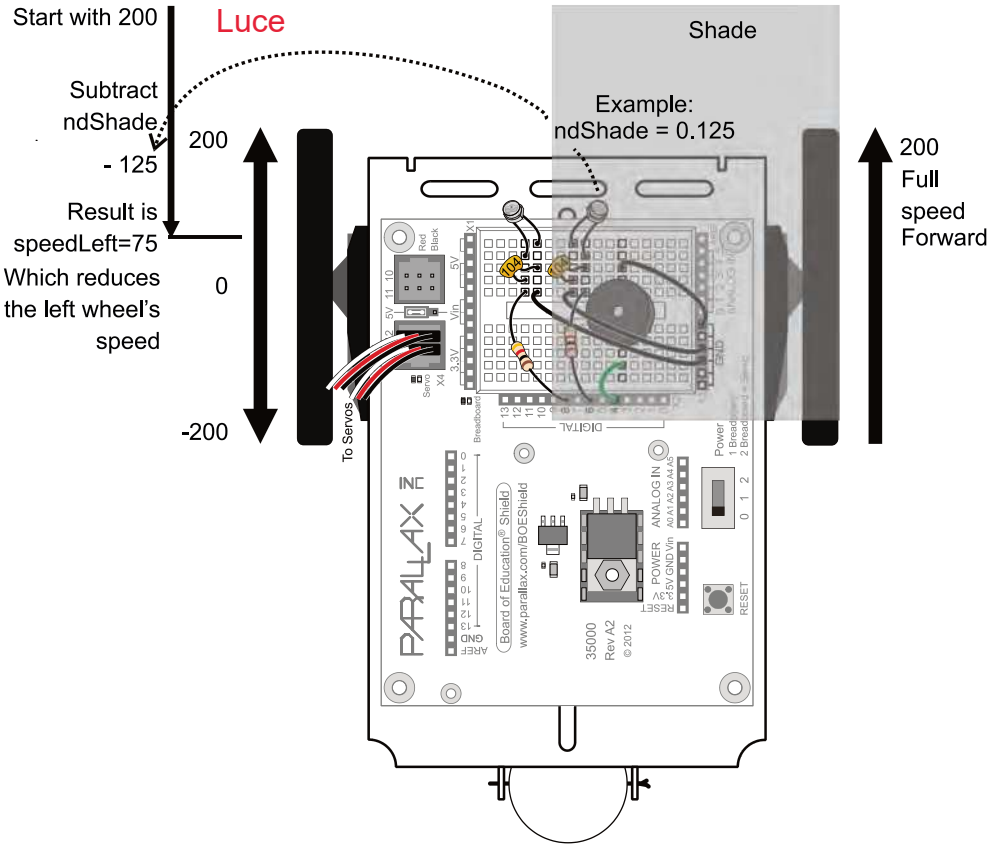
One approach toward making the Boe-Bot roam toward light sources is to make it turn away from shade. You can use the **ndShade** variable to make the BOE Shield-Bot turn a little or a lot when the contrast between the light detected on each side is a little or a lot.

Shady Navigation Decisions

Here is an **if** statement that works well for turning away from shade on the right side of the BOE Shield-Bot. It starts by declaring two **int** variables, **speedLeft** and **speedRight**. They are not declared within the **if**…**else** block because other blocks in the **loop** function will need to check their values too. Next, **if(ndShade > 0.0)** has a code block that will be executed if shade is detected on the robot's right side, slowing down the left wheel to make the BOE Shield-Bot turn away from the dark. To do this, **ndShade * 1000.0** is subtracted from 200. Before assigning the result to **speedLeft**, **int(200.0–(ndShade×1000.0)** converts the answer from a floating point value back to an integer. We're doing this to make the value compatible with our custom **maneuver** function , which needs an **int** value.

```
int speedLeft, speedRight;                  // Declare speed variables

if (ndShade > 0.0)                          // Shade on right? Luce a sinistra
{                                           // Slow down left wheel
  speedLeft = int(200.0 - (ndShade * 1000.0));
  speedLeft = constrain(speedLeft, -200, 200);
  speedRight = 200;                         // Full speed right wheel
}
```

This diagram shows an example of how this works when **`ndShade`** is 0.125. The left wheel slows down because 200 – (0.125×1000) = 75. Since linear speed control is in the 100 to -100 range, it puts the wheel at about ¾ of full speed. Meanwhile, on the other side, **`speedRight`** is set to 200 for full speed forward.



The larger **ndShade** is, the more it subtracts from 200. That's not a problem in this example, but if **ndShade** were 0.45, it would try to store –250 in the **speedLeft** variable. Since the speeds we'll want to pass to the **maneuver** function need to be in the -200 to 200 range, we'll use the Arduino's **constrain** function to prevent **speedLeft** from going out of bounds: **speedLeft = constrain(speedLeft, –200, 200).**

Here is an **else** statement that works well for turning away from shade on the left. It slows down the right wheel and keeps the left wheel going full speed forward. Notice that it adds **(ndShade*1000)** to 200. Reason being, this is the **else** statement for **if(ndShade > 0.0)**, so it will get used when **ndShade** is equal to or smaller than zero. So, if **ndShade** is –0.125, **speedRight = int(200.0 + (ndShade * 1000.0))** would evaluate to 200 + (–1.25 × 1000) = 200 – 125 = 75. The **constrain** function is used again, to limit **speedRight**.

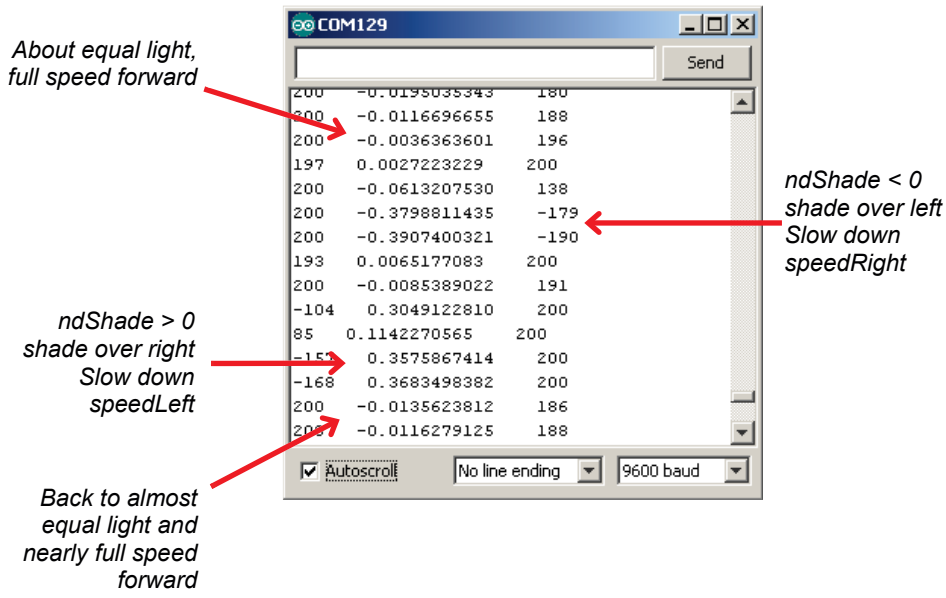
```
  else                                    // Shade on Left?
  {                                       // Slow down right wheel
    speedRight = int(200.0 + (ndShade * 1000.0));
    speedRight = constrain(speedRight, -200, 200);
    speedLeft = 200;                      // Full speed left wheel
  }
```

Test Navigation Decisions with Serial Monitor

Before actually testing out these navigation decisions, it's best to take a look at the variable values with the Serial Monitor. So, instead of a call to the **maneuver** function, first, let's use some **Serial.print** calls to see if we got it right.

```
  Serial.print(speedLeft, DEC);           // Display speedLeft
  Serial.print("   ");                    // Spaces
  Serial.print(ndShade, DEC);             // Display ndShade
  Serial.print("    ");                   // More spaces
  Serial.println(speedRight, DEC);        // Display speedRight

  delay(2000);                            // 1 second delay
}
```

The **print** and **println** calls should result in a display like the one below. It shows the value of **speedLeft** in the left column, **speedRight** in the right column, and **ndShade** between them. Watch it carefully. The side with brighter light will always display 200 for full-speed-forward, and the other will be slowing down with values less than 200—the darker the shade, the smaller the number.



Example Sketch – Light Seeking Display

- ✓ Make sure the power switch is set to 1.
- ✓ Create, save, and run the sketch LightSeekingDisplay.
- ✓ Open the Serial Monitor.
- ✓ Try casting different levels of shade over the BOE Shield-Bot's right light sensor. Does **`speedLeft`** (in the left column) slow down or even go into reverse with lots of shade?
- ✓ Try the same thing with the left light sensor to verify the right wheel slows down.
- ✓ Try casting more shade over both. Again, since the shade is the same for both, **`ndShade`** should stay close to zero, with little if any slowing of the wheels. (Remember, **`speedLeft`** and **`SpeedRight`** would have to drop by 100 before they'll start to slow down.)

```
/*
 * Robotics with the BOE Shield - LightSeekingDisplay
 * Displays speedLeft, ndShade, and speedRight in Serial Monitor. Verifies
 * that wheel speeds respond correctly to left/right light/shade conditions.
 */

void setup()                              // Built-in initialization block
{
  tone(4, 3000, 1000);                    // Play tone for 1 second
  delay(1000);                            // Delay to finish tone
```

```
  Serial.begin(9600);                   // Set data rate to 9600 bps
}

void loop()                             // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));       // Get left light & make float
  float tRight = float(rcTime(6));      // Get right light & make float

  float ndShade;                        // Normalized differential shade
  ndShade = tRight / (tLeft+tRight) - 0.5;   // Calculate & subtract 0.5

  int speedLeft, speedRight;            // Declare speed variables

  if (ndShade > 0.0)                    // Shade on right?
  {                                     // Slow down left wheel
    speedLeft = int(200.0 - (ndShade * 1000.0));
    speedLeft = constrain(speedLeft, -200, 200);
    speedRight = 200;                   // Full speed right wheel
  }
  else                                  // Shade on Left?
  {                                     // Slow down right wheel
    speedRight = int(200.0 + (ndShade * 1000.0));
    speedRight = constrain(speedRight, -200, 200);
    speedLeft = 200;                    // Full speed left wheel
  }

  Serial.print(speedLeft, DEC);         // Display speedLeft
  Serial.print("   ");                  // Spaces
  Serial.print(ndShade, DEC);           // Display ndShade
  Serial.print("    ");                 // More spaces
  Serial.println(speedRight, DEC);      // Display speedRight

  delay(1000);                          // 1 second delay
}

long rcTime(int pin)                    // rcTime measures decay at pin
{
  pinMode(pin, OUTPUT);                 // Charge capacitor
  digitalWrite(pin, HIGH);              // ..by setting pin ouput-high
  delay(5);                             // ..for 5 ms
  pinMode(pin, INPUT);                  // Set pin to input
  digitalWrite(pin, LOW);               // ..with no pullup
  long time  = micros();                // Mark the time
  while(digitalRead(pin));              // Wait for voltage < threshold
  time = micros() - time;               // Calculate decay time
  return time;                          // Returns decay time
}
```

Activity 5: Shield-Bot Navigating by Light

At this point, the LightSeekingDisplay sketch needs four things to take it from displaying what it's going to do to actually doing it:

- ✓ Remove the **`Serial.print`** calls.
- ✓ Add servo code.
- ✓ Add the **`maneuver`** function.
- ✓ Add a call to the **`loop`** function to pass **`speedLeft`** and **`speedRight`** to the **`maneuver`** function.

The result is the LightSeekingShieldBot sketch.

- ✓ Create, save, and run the sketch LightSeekingShieldBot.
- ✓ Connect the battery pack, put the BOE Shield-Bot on the floor, and set the power switch to 2.
- ✓ Let the BOE Shield-Bot roam and try casting shadows over its left and right light sensors. It should turn away from the shadow.

```
/*
 * Robotics with the BOE Shield - LightSeekingShieldBot
 * Roams toward light and away from shade.
 */

#include <Servo.h>                      // Include servo library

Servo servoLeft;                        // Declare left and right servos
Servo servoRight;

void setup()                            // Built-in initialization block
{
  tone(4, 3000, 1000);                  // Play tone for 1 second
  delay(1000);                          // Delay to finish tone

  servoLeft.attach(13);                 // Attach left signal to pin 13
  servoRight.attach(12);                // Attach right signal to pin 12
}

void loop()                             // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));       // Get left light & make float
  float tRight = float(rcTime(6));      // Get right light & make float

  float ndShade;                        // Normalized differential shade
  ndShade = tRight / (tLeft+tRight) - 0.5;   // Calculate & subtract 0.5

  int speedLeft, speedRight;            // Declare speed variables

  if (ndShade > 0.0)                    // Shade on right?
  {                                     // Slow down left wheel
```

```
    speedLeft = int(200.0 - (ndShade * 1000.0));
    speedLeft = constrain(speedLeft, -200, 200);
    speedRight = 200;                   // Full speed right wheel
  }
  else                                  // Shade on Left?
  {                                     // Slow down right wheel
    speedRight = int(200.0 + (ndShade * 1000.0));
    speedRight = constrain(speedRight, -200, 200);
    speedLeft = 200;                    // Full speed left wheel
  }

  maneuver(speedLeft, speedRight, 20);  // Set wheel speeds
}

long rcTime(int pin)                    // rcTime measures decay at pin
{
  pinMode(pin, OUTPUT);                 // Charge capacitor
  digitalWrite(pin, HIGH);              // ..by setting pin ouput-high
  delay(5);                             // ..for 5 ms
  pinMode(pin, INPUT);                  // Set pin to input
  digitalWrite(pin, LOW);               // ..with no pullup
  long time  = micros();                // Mark the time
  while(digitalRead(pin));              // Wait for voltage < threshold
  time = micros() - time;               // Calculate decay time
  return time;                          // Returns decay time
}

// maneuver function
void maneuver(int speedLeft, int speedRight, int msTime)
{
  servoLeft.writeMicroseconds(1500 + speedLeft);   // Left servo speed
  servoRight.writeMicroseconds(1500 - speedRight); // Right servo speed
  if(msTime==-1)                                   // If msTime = -1
  {
    servoLeft.detach();                            // Stop servo signals
    servoRight.detach();
  }
  delay(msTime);                                   // Delay for msTime
}
```

Your Turn – Light/Shade Sensitivity Adjustments

- ✓ If you want more sensitivity to light, change 1000 to a larger value in these two commands:

```
    speedLeft = int(200.0 - (ndShade * 1000.0));
    speedRight = int(200.0 + (ndShade * 1000.0));
```

Want less light sensitivity? Change 1000 to a smaller value.

- ✓ Try it.

Here are several more light-sensing navigation ideas for your BOE Shield-Bot that can be made with adjustments to the **`loop`** function:

- ✓ To make your BOE Shield-Bot follow shade instead of light, place **`ndShade = -ndShade`** right before the **`if`**…**`else`** statement. Curious about how or why this works? Check out Project 2 at the end of this chapter.
- ✓ End roaming under a bright light or in a dark cubby by detecting very bright or very dark conditions. Add **`tLeft`** and **`tRight`** together, and compare the result to either a really high (dark) threshold value or a really low (bright) threshold value.
- ✓ Make your BOE Shield-Bot function as a light compass by remaining stationary and rotating toward bright light sources.
- ✓ Incorporate whiskers into the roaming toward light activity so that the BOE Shield-Bot can detect and navigate around objects in its way.

Chapter 6 Summary

This chapter focused on using a pair of light sensors to detect bright light and shade for robot navigation. Lots of interesting electronics concepts and programming techniques come into play.

Electronics

- What a phototransistor is, and how to identify its base, emitter and collector
- What wavelengths are in the ultraviolet, visible, and infrared spectrums
- What is meant by ambient light
- What the difference is between a binary sensor and an analog sensor
- What Ohm's Law is, and how to use it to select a resistor to adjust the phototransistor circuit's voltage response
- Using a phototransistor as a simple binary sensor in a voltage output circuit
- What a capacitor is, and how small capacitors for breadboard circuits are labeled in units of picofarads
- Using a phototransistor as an analog sensor in a resistor-capacitor charge-transfer circuit, also called a QT circuit
- What it means when components are connected in series vs. connected in parallel
- What voltage decay is, and how it's used in a resistor-capacitor circuit

Programming

- How to use the Arduino's **`analogRead`** function to take a voltage measurement from an analog sensor's output
- How a sketch can measure voltage decay time from a resistor-capacitor circuit to take a measurement from an analog sensor

- How to use the Arduino's **`constrain`** function to set upper and lower limits for a variable value

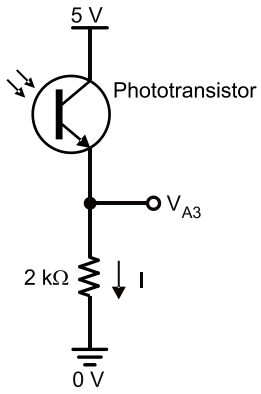
Robotics Skills

- Using a pair of phototransistors for autonomous sensor navigation in response to light level

Engineering Skills

- Subsystem testing of circuits and code routines
- The concept of resolution, in the context of the Arduino reporting a 10-bit value from an analog input
- Using an equation to get a zero-justified normalized differential measurement

Chapter 6 Challenges



Questions

1. What does a transistor regulate?
2. Which phototransistor terminals have leads?
3. How can you use the flat spot on the phototransistor's plastic case to identify its terminals?
4. Which color would the phototransistor be more sensitive to: red or green?
5. How does V_{A3} in the circuit above respond if the light gets brighter?
6. What does the phototransistor in the circuit above do that causes V_{A3} to increase or decrease?
7. How can the circuit above be modified to make it more sensitive to light?