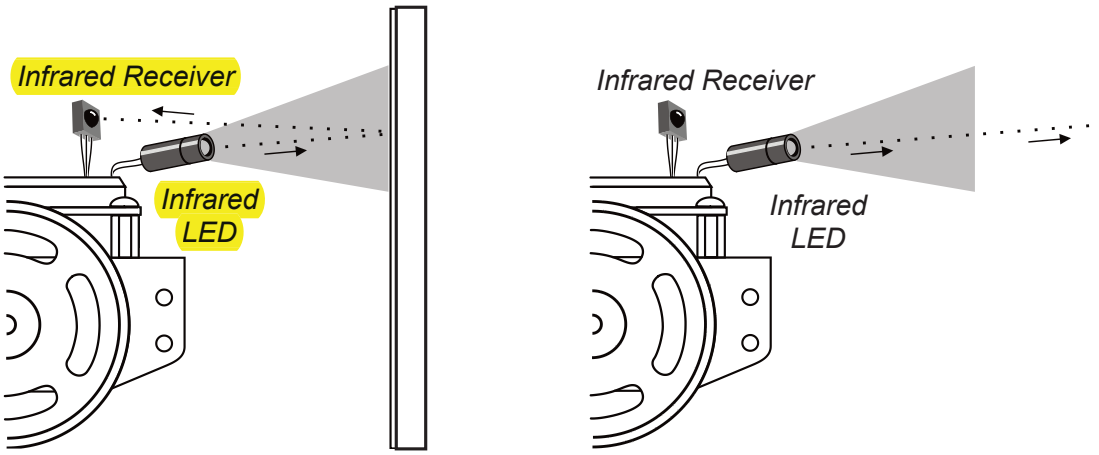
Chapter 7. Navigating with Infrared Headlights

The BOE Shield-Bot can already use whiskers to get around, but it only detects obstacles when it bumps into them. Wouldn't it be convenient if the BOE Shield-Bot could just "see" objects and then decide what to do about them? Well, that's what it can do with infrared headlights and eyes like the ones shown below. Each headlight is an infrared LED inside a tube that directs the light forward, just like a flashlight. Each eye is an infrared receiver that sends the Arduino high/low signals to indicate whether it detects the infrared LED's light reflected off an object.

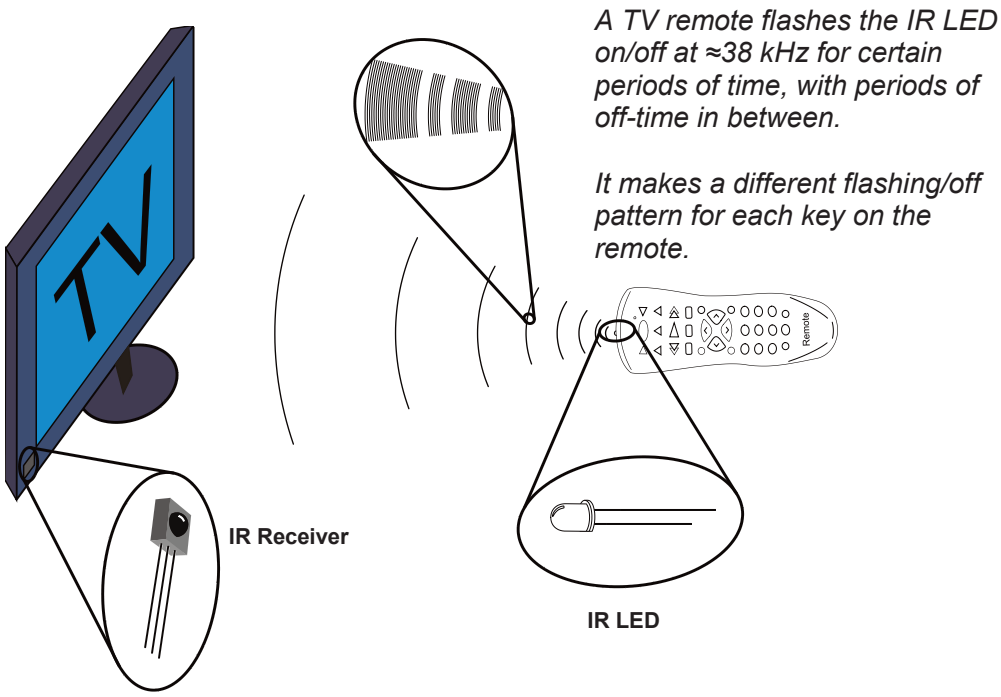


Infrared Light Signals

Infrared is abbreviated IR, and it is light the human eye cannot detect (for a color image of the visible light spectrum, see http://learn.parallax.com/lightspectrum). The IR LEDs introduced in this chapter emit infrared light, just like the red LEDs we've been using emit visible light.

The infrared receivers in this chapter detect infrared light, similar to the phototransistors in the last chapter. But, there's a difference—these infrared receivers are not just detecting ambient light, but they are designed to detect infrared light flashing on and off very quickly.

The infrared LED that the BOE Shield-Bot will use as a tiny headlight is actually the same kind you can find in just about any TV remote. The TV remote flashes the IR LED to send messages to your TV. The microcontroller in your TV picks up those messages with an infrared receiver like the one your BOE Shield-Bot will use.

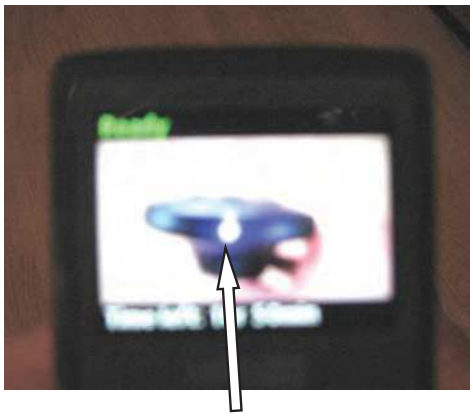


The TV remote sends messages by flashing the IR LED very fast, at a rate of about 38 kHz (about 38,000 times per second). The IR receiver only responds to infrared if it's flashing at this rate. This prevents infrared from sources like the sun and incandescent lights from being misinterpreted as messages from the remote. So, to send signals that the IR receiver can detect, your Arduino will have to flash the IR LED on/off at 38 kHz.

IR Interference: Some fluorescent lights do generate signals that can be detected by the IR receivers. These lights can cause problems for your BOE Shield-Bot's infrared headlights. One of the things you will do in this chapter is develop an infrared interference "sniffer" that you can use to test the fluorescent lights near your BOE Shield-Bot courses.

The light sensors inside many digital cameras, including some cell phones and webcams, can all detect infrared light. By looking through a digital camera, we can "see" if an infrared LED is on or off. These photos show an example with a digital camera and a TV remote. When you press and hold a button on the remote and point the IR LED into the digital camera's lens, it displays the infrared LED as a flashing, bright white light.

With a button pressed and held, the IR LED doesn't look any different.



Through a digital camera display, the IR LED appears as a bright white light.

The pixel sensors inside the digital camera detect red, green, and blue light levels, and the processor adds up those levels to determine each pixel's color and brightness. Regardless of whether a pixel sensor detects red, green, or blue, it detects infrared. Since all three pixel color sensors also detect infrared, the digital camera display mixes all the colors together, which results in white.

Infra means below, so infrared means below red.
The name refers to the fact that the frequency of infrared light waves is less than the frequency of red light waves. The wavelength our IR LED transmits is 980 nanometers (abbreviated nm), and that's the same wavelength our IR receiver detects. This wavelength is in the near-infrared range. The far-infrared range is 2000 to 10,000 nm, and certain wavelengths in this range are used for night-vision goggles and IR temperature sensing.

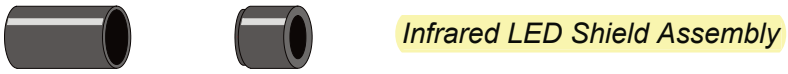
Activity 1: Build and Test the Object Detectors

In this activity, you will build and test infrared object detectors for the BOE Shield-Bot.

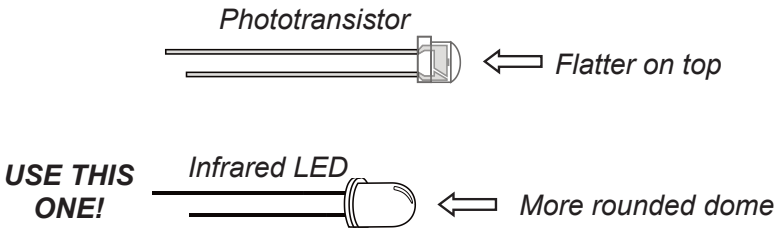
Parts List

(2) IR receivers
(2) IR LEDs (clear case)
(2) IR LED shield assemblies
(2) Resistors, 220 Ω (red-red-brown)
(2) Resistors, 2 kΩ (red-black-red)
(misc) Jumper wires

- ✓ Gather the parts in the Parts List, using the drawings below to help identify the infrared receivers, LEDs, and shield assembly parts.



✓ Check the figure below to make sure you have selected infrared LEDs and not phototransistors. The infrared LED has a taller and more rounded plastic dome, and is shown on the right side of this drawing.

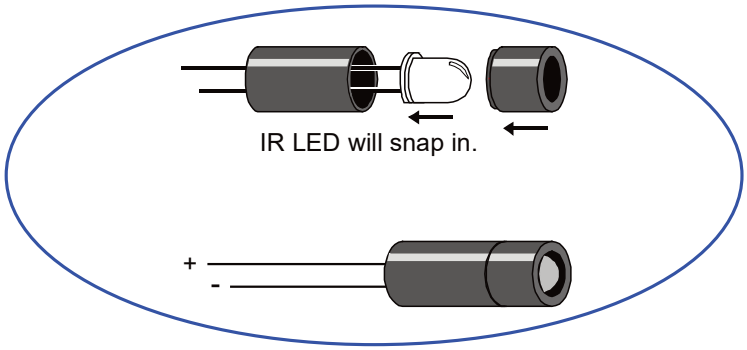


Assemble the IR Headlights

✓ Insert the infrared LED into the LED standoff base (the larger of the two pieces) as shown below. The standoff is shaped to fit the flat side of the LED.

✓ Make sure the IR LED snaps into the LED standoff.

✓ Slip the short tube over the IR LED's clear plastic case. The ring on one end of the tube should fit right into the LED standoff with a little twist.

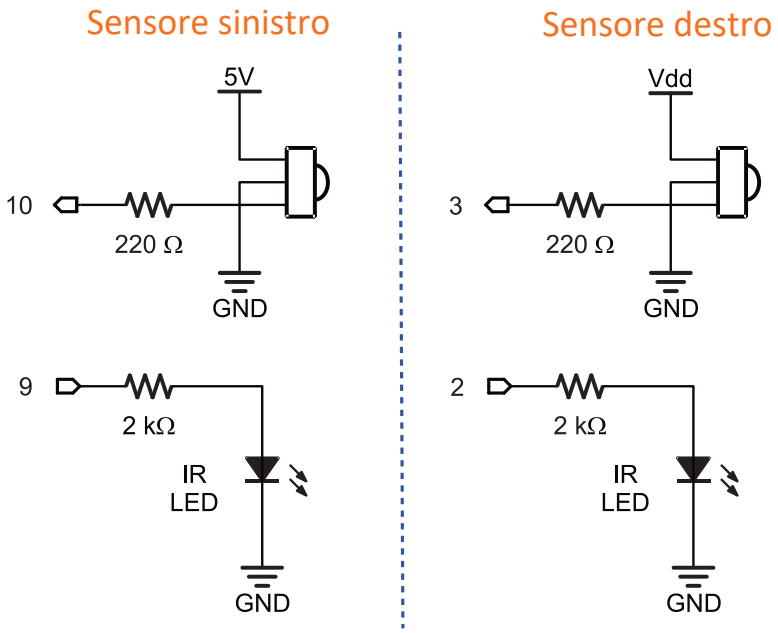


IR Object Detection Circuit

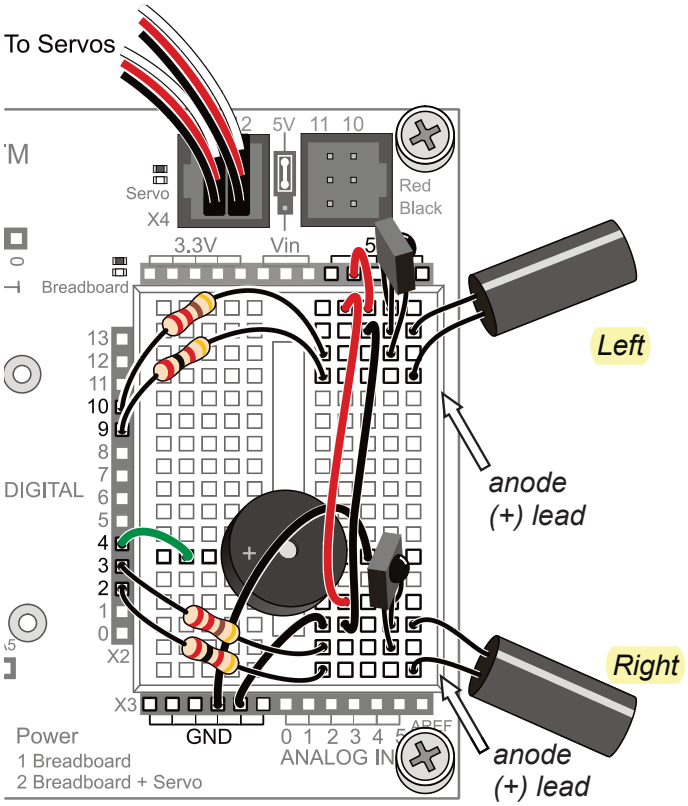
The next figures show the IR object detection schematic and wiring diagram. One IR object detector (IR LED and receiver pair) is mounted on each corner of the breadboard closest to the very front of the BOE Shield-Bot.

- ✓ Disconnect the power and programming cables.
- ✓ Build the circuit in the schematic below, using the wiring diagram as a reference for parts placement.

Note that the anode lead of each IR LED connects to a 2 kΩ resistor. The cathode lead plugs into the same breadboard row as an IR detector's center pin, and that row is connected to GND with a jumper wire.



Watch your IR LED anodes and cathodes!
The anode lead is the longer lead on an IR LED by convention. The cathode lead is shorter and mounted in the plastic case closer to its flat spot. These are the same conventions as the red LEDs we have been using.



Object Detection Test Code

Your BOE Shield-Bot's infrared receivers are designed to detect infrared light (in the 980 nanometer range) flashing at a rate near 38 kHz. To make the IR LED turn on/off at that rate, we can use the familiar **tone** function that makes the speaker beep at the beginning of each sketch.

Infrared detection takes three steps:

1. Flash the IR LED on/off at 38 kHz.
2. Delay for a millisecond or more to give the IR receiver time to send a low signal in response to sensing 38 kHz IR light reflecting off an object.
3. Check the state of the IR receiver for either a high signal (no IR detected), or a low signal (IR detected).

Here is an example of the three steps applied to the left IR LED (pin 9) and IR receiver (pin 10).

```
tone(9, 38000, 8);              // IRLED 38 kHz for at least 1 ms
delay(1);                       // Wait 1 ms
int ir = digitalRead(10);       // IR receiver -> ir variable
```

The tone actually lasts about 1.1 ms. Even though we would expect `tone(9, 38000, 8)` to generate a 38 kHz signal for 8 ms, the signal actually only lasts for about 1.1 ms as of Arduino software v1.0. Given the name `tone`, the function may have been designed for and tested with audible tones. If played on a speaker, a 38 kHz tone will not be audible. It's in the ultrasonic range, meaning it's pitch is so high that it's above the audible range for humans, which is about 20 Hz to 20 kHz.

The `tone` function generates a square wave that repeats its high/low cycle 38000 times per second. Through experimentation with Arduino software v1.0, the author discovered that a call to `tone` with a duration of 8 ms resulted in a 38 kHz square wave that lasted about 1.1 ms.

Remember that the `tone` function does its processing in the background. Since the IR receiver needs a few tenths of a millisecond to respond to the 38 kHz signal, `delay(1)` prevents the `ir = digitalRead(10)` call from copying the IR receiver's output until it's ready.

The next sketch defines a function named `irDetect` with three parameters: one to specify the IR LED pin, one to specify the IR receiver pin, and one to set the frequency for flashing the IR LED.

```
int irDetect(int irLedPin, int irReceiverPin, long frequency)
```

In the `loop` function you'll see a call to `irDetect`:

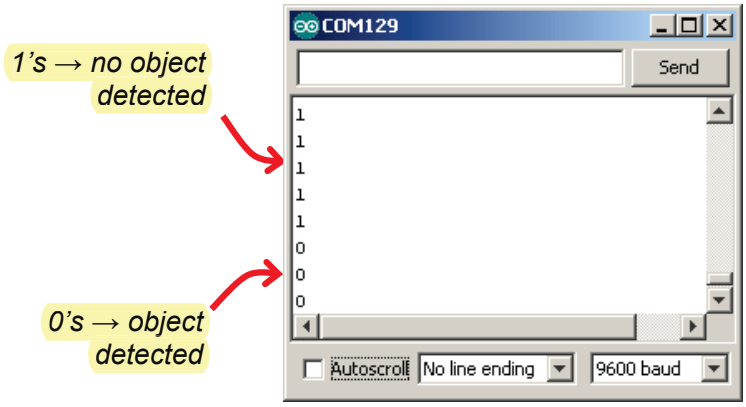
```
int irLeft = irDetect(9, 10, 38000);       // Check for object
```

This call passes 9 to the `irLedPin` parameter, 10 to `irReceiverPin`, and 38000 to the *`frequency`* parameter. The function performs those three steps for infrared detection and returns 1 if no object is detected, or 0 if an object is detected. That return value gets stored in `irLeft`.

Example Sketch: TestLeftIr

This sketch only tests the BOE Shield-Bot's left IR detector. This helps simplify trouble-shooting because you are focusing on only one of the two circuits. This is yet another example of subsystem testing. After the subsystems check out, we can move to system integration. But first, you've got to make sure to test and correct any wiring or code entry errors that might have crept in.

- ✓ Reconnect the battery pack to the Arduino.
- ✓ Set the 3-position switch to position 1.
- ✓ Create, save, and run the sketch the sketch TestLeftIr.




```
/*
 * Robotics with the BOE Shield - TestLeftIR
 * Display 1 if the left IR detector does not detect an object,
 * or 0 if it does.
 */

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

  pinMode(10, INPUT);  pinMode(9, OUTPUT);   // Left IR LED & Receiver

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

void loop()                                 // Main loop auto-repeats
{
  int irLeft = irDetect(9, 10, 38000);  // Check for object

  Serial.println(irLeft);                   // Display 1/0 no detect/detect

  delay(100);                               // 0.1 second delay
}

// IR Object Detection Function
int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);             // IRLED 38 kHz at least 1 ms
  delay(1);                                 // Wait 1 ms
  int ir = digitalRead(irReceiverPin);  // IR receiver -> ir variable
  delay(1);                                 // Down time before recheck
  return ir;                                // Return 1 no detect, 0 detect
}
```

✓ Leave the BOE Shield-Bot connected to its programming cable, and open the Serial Monitor when the sketch is done loading.

- ✓ Place an object, such as your hand or a sheet of paper, about an inch (2 to 3 cm) from the left IR object detector.
- ✓ Verify that the Serial Monitor displays a 0 when you place an object in front of the IR object detector, and a 1 when you remove the object.
- ✓ If the Serial Monitor displays the expected values, go ahead and test the right IR Object Detector (below). If not, go to the Troubleshooting section for help.

Your Turn – Test the Right IR Object Detector

Modifying the sketch to test the right IR object detector is a matter of replacing **`irLeft`** with **`irRight`**, and passing the correct pin numbers to the **`irDetect`** parameters to test the other circuit. Here's a checklist of the changes:

- ✓ Save the sketch TestLeftIr as TestRightIr.
- ✓ Change **`pinMode(10, INPUT); pinMode(9, OUTPUT)`** to **`pinMode(3, INPUT); pinMode(2, OUTPUT).`**
- ✓ Change **`int irLeft = irDetect(9, 10, 38000)`** to **`int irRight = irDetect(2, 3, 38000).`**
- ✓ Change **`Serial.println(irLeft)`** to **`Serial.println(irRight).`**
- ✓ Repeat the testing steps in this activity for the BOE Shield-Bot's right IR object detector.
- ✓ If necessary, trouble-shoot any circuit or code entry errors.

Activity 2: Field Testing

In this activity, you will build and test indicator LEDs that will tell you if an object is detected without the help of the Serial Monitor. This is handy if you are not near your computer, and you need to troubleshoot your IR detector circuits.

You will also write a sketch to "sniff" for infrared interference from fluorescent lights. Some fluorescent lights send signals that resemble the signal sent by your infrared LEDs. The device inside a fluorescent light fixture that controls voltage for the lamp is called the *ballast*. Some ballasts operate in the same frequency range of your IR detector, causing the lamp to emit a 38.5 kHz infrared signal. When using IR object detection for navigation, ballast interference can cause some bizarre BOE Shield-Bot behavior!

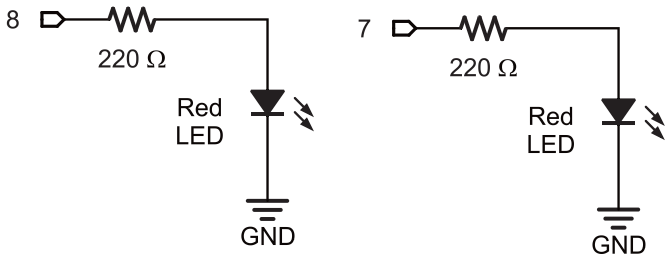
Adding LED Indicator Circuits

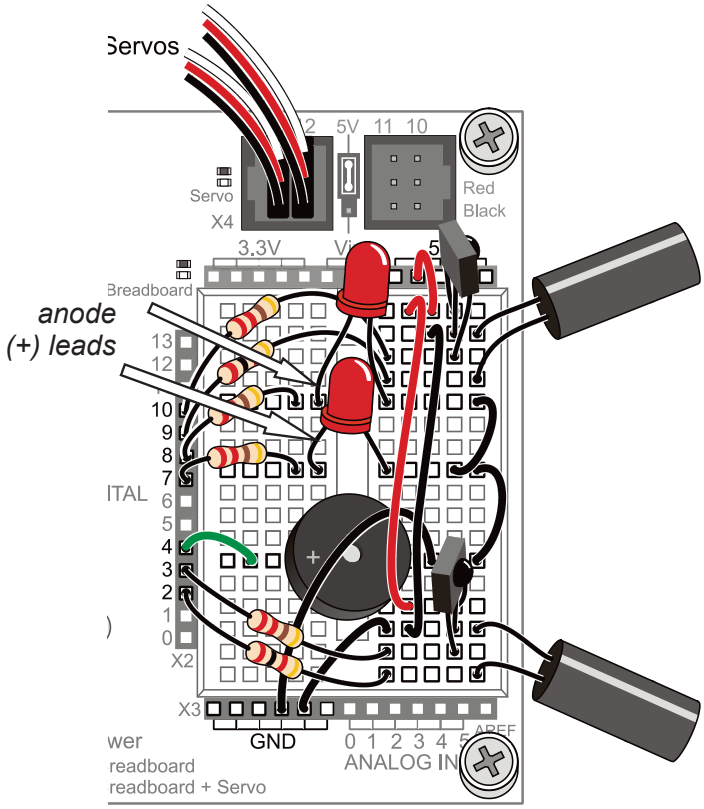
LED indicator circuits are similar to the ones you used with the whiskers. Make sure to be careful about your cathodes and anodes when you connect them.

Parts List

(2) Red LEDs
(2) Resistors, 220 Ω (red-red-brown)
(misc) Jumper wires

- ✓ Disconnect the programming and power cables.
- ✓ Add the red LED circuits shown in the schematic to the BOE Shield, using the wiring diagram to help with parts placement.





Testing the System

There are quite a few components involved in this system, and this increases the likelihood of a wiring error. That's why it's important to have a test sketch that shows you what the infrared detectors are sensing. Use this sketch to verify that all the circuits are working before unplugging the BOE Shield-Bot from its programming cable.

Example Sketch – TestBothIrAndIndicators

- ✓ Reconnect the programming and power cables, and set the Power switch to 1.
- ✓ Create, save, and run the sketch TestBothIrAndIndicators.
- ✓ Verify that the speaker makes a clear, audible tone after the sketch loads.
- ✓ Use the Serial Monitor to verify that the Arduino still receives a zero from each IR detector when an object is placed in front of it.
- ✓ Verify that when you place an object in front of the left IR detector, the left (pin 8) red LED lights up.
- ✓ Repeat that test for the right IR detector and pin 7 LED.

```
/*
 * Robotics with the BOE Shield - TestBothIrAndIndicators
 * Test both IR detection circuits with the Serial Monitor.  Displays 1 if
 * the left IR detector does not detect an object, or 0 if it does.
 * Also displays IR detector states with indicator LEDs.
 */

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

  pinMode(10, INPUT);  pinMode(9, OUTPUT);    // Left IR LED & Receiver
  pinMode(3, INPUT);  pinMode(2, OUTPUT);     // Right IR LED & Receiver
  pinMode(8, OUTPUT); pinMode(7, OUTPUT);     // Indicator LEDs

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

void loop()                               // Main loop auto-repeats
{
  int irLeft = irDetect(9, 10, 38000);  // Check for object on left
  int irRight = irDetect(2, 3, 38000);  // Check for object on right

  digitalWrite(8, !irLeft);               // LED states opposite of IR
  digitalWrite(7, !irRight);

  Serial.print(irLeft);                   // Display 1/0 no detect/detect
  Serial.print("  ");                     // Display 1/0 no detect/detect
  Serial.println(irRight);                // Display 1/0 no detect/detect
```

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

// IR Object Detection Function

int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);           // IRLED 38 kHz for at least 1 ms
  delay(1);                               // Wait 1 ms
  int ir = digitalRead(irReceiverPin);    // IR receiver -> ir variable
  delay(1);                               // Down time before recheck
  return ir;                              // Return 1 no detect, 0 detect
}
```

Inside TestBothIrAndIndicators

After the two calls to the **irDetect** function, **irLeft** and **irRight** each store a 1 if an object is not detected, or 0 if an object is detected. The sketch could have used an **if…else** statement to turn the indicator lights on/off depending on the value of **irLeft** and **irRight**. But, there are other ways!

This approach uses the values of **irLeft** and **irRight** directly. There's only one catch: when (for example) **irLeft** stores 0, we want its red indicator LED to turn on, and if it stores 1, we want its LED to turn off. Since 1 makes the indicator LED turn on and 0 turns it off, using **digitalWrite(8, irLeft)** would give us the opposite of what we want. So, the sketch uses the not operator (**!**) to invert the value that **irLeft** stores. Now, **digitalWrite(8, !irLeft)** inverts the value of **IrLeft** so that when it stores a zero, the LED turns on, and when it stores 1, the LED turns off. The same technique is applied for the right IR detector and indicator LED.

The Not (!) Operator inverts all the binary values in a variable. There's more going on with **digitalWrite(8, !irLeft)** here than meets the eye. You're probably used to passing the **digitalWrite** function's ***value*** parameter either **HIGH** (constant for 1) to turn the light on, or **LOW** (constant for 0) to turn the light off. When you use variables, the **digitalWrite** function uses the variable's rightmost binary digit: 1 to turn the light on, or 0 to turn the light off. So, when **irLeft** is 0 (object detected) **!irLeft** changes its binary value 00000000 to 11111111. It has a 1 in the rightmost digit, so the light turns on. When **irLeft** is 1, it's really binary 00000001. The **!irLeft** statement results in binary 11111110. Rightmost digit is 0, so the light turns off.

Your Turn – Remote Testing and Range Testing

With the indicator LEDs working, you can now unplug the BOE Shield-Bot from its programming cable and test detection with a variety of objects. Since certain objects reflect infrared better than others, the IR object detectors will be able to see some objects further away, and others only when very close.

- ✓ Make sure the battery pack is connected to your Arduino and the BOE Shield's Power switch is set to 1.
- ✓ Unplug your BOE Shield-Bot from the programming cable, and test again with objects in the area where you plan to make it navigate.
- ✓ Test the detection range with different colored objects. What colors, materials, and surface textures can it detect only at closest range? What can it detect from farther away?

Sniffing for IR Interference

You might have found that your BOE Shield-Bot said it detected something even though nothing was in range. That may mean a nearby light is generating some IR light at a frequency close to 38.5 kHz. It might also mean that direct sunlight streaming through a window is causing false detections. If you try to have a BOE Shield-Bot contest or demonstration near one of these light sources, your infrared systems could end up performing very poorly! So, before any public demo, make sure to check the prospective navigation area with this IR interference "sniffer" sketch ahead of time.

The concept behind this sketch is simple: don't transmit any IR through the IR LEDs, just monitor to see if any IR is detected. If IR is detected, sound the alarm using the piezospeaker.

You can use a handheld remote for just about any piece of equipment to generate IR interference. TVs, VCRs, CD/DVD players, and projectors all use the same type of IR detectors you have on your BOE Shield-Bot right now. So, the remotes you use to control these devices all use the same kind of IR LED that's on your BOE Shield-Bot to transmit messages to your TV, VCR, CD/DVD player, etc. All you'll have to do to generate IR interference is point the remote at your BOE Shield-Bot and repeatedly press/release one of the remote's buttons.

Example Sketch – IrInterferenceSniffer

With this sketch, your BOE Shield-Bot should play a tone, turn on its indicator LEDs, and display a warning in the Serial Monitor any time it detects infrared. Again, since it's not transmitting any IR, it means the 38 kHz infrared has to be coming from an outside source.

- ✓ Create, save, and run the sketch IrInterferenceSniffer.
- ✓ Test to make sure the BOE Shield-Bot sounds the alarm when it detects IR interference. If you are in a classroom, you can do this with a separate BOE Shield-Bot that's running TestBothIrAndIndicators. Just point its IR LEDs into the IrInterferenceSniffer bot's IR receivers. If you don't have a second BOE Shield-Bot, just use a handheld remote for a TV, VCR, CD/DVD player, or projector. Simply point the remote at the BOE Shield-Bot and repeatedly press and release

one of its buttons. If the BOE Shield-Bot responds by sounding the alarm, you know your IR interference sniffer is working.

```
/*
 * Robotics with the BOE Shield - IrInterferenceSniffer
 * Test for external sources of IR interference.  If IR interference is
 * detected: Serial Monitor displays warning, piezospeaker plays alarm,
 * and indicator lights flash.
 */

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

  pinMode(10, INPUT);                     // Left IR Receiver
  pinMode(3, INPUT);                      // Right IR Receiver
  pinMode(8, OUTPUT);                     // Left indicator LED
  pinMode(7, OUTPUT);                     // Right indicator LED

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

void loop()                               // Main loop auto-repeats
{
  int irLeft = digitalRead(10);           // Check for IR on left
  int irRight = digitalRead(3);           // Check for IR on right

  if((irLeft == 0) || (irRight == 0))     // If left OR right detects
  {
    Serial.println("IR interference!!!");    // Display warning
    for(int i = 0; i < 5; i++)            // Repeat 5 times
    {
      digitalWrite(7, HIGH);              // Turn indicator LEDs on
      digitalWrite(8, HIGH);
      tone(4, 4000, 10);                  // Sound alarm tone
      delay(20);                          // 10 ms tone, 10 between tones
      digitalWrite(7, LOW);               // Turn indicator LEDs off
      digitalWrite(8, LOW);
    }
  }
}
```

Your Turn – Testing for Fluorescent Lights that Interfere

- ✓ Disconnect your BOE Shield-Bot from its programming cable, and point it at any fluorescent light near where you plan to operate it.
- ✓ Especially if you get frequent alarms, turn off that fluorescent light before trying to use IR object detection under it.

- ✓ If the source turns out to be sunlight streaming in through a window, close the blinds and retest, or move your obstacle course to a location that's well away from the window.

Always use this IrInterferenceSniffer to make sure that any area where you are using the BOE Shield-Bot is free of infrared interference.

Activity 3: Detection Range Adjustments

You may have noticed that brighter car headlights (or a brighter flashlight) can be used to see objects that are further away when it's dark. By making the BOE Shield-Bot's infrared LED headlights brighter, you can also increase its detection range. A smaller resistor allows more current to flow through an LED. More current through an LED is what causes it to glow more brightly. In this activity, you will examine the effect of different resistance values with both the red and infrared LEDs.

Parts List:

You will need some extra parts for this activity:

(2) Resistors, 470 Ω (yellow-violet-brown)
(2) Resistors, 220 Ω (red-red-brown)
(2) Resistors, 1 kΩ (brown-black-red)
(2) Resistors, 4.7 kΩ (yellow-violet-red)

Series Resistance and LED Brightness

First, let's use one of the red LEDs to see the difference that a resistor makes in how brightly an LED glows. All we need to test the LED is a sketch that sends a high signal to the LED.

Example Sketch – P1LedHigh

- ✓ Create, save, and run the sketch LeftLedOn.
- ✓ Run the sketch and verify that the LED in the circuit connected to P8 emits light.

```
// Robotics with the BOE Shield - LeftLedOn
// Turn on left LED for brightness testing

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

  pinMode(8, OUTPUT);                     // Left indicator LED
  digitalWrite(8, HIGH);
}

void loop()                               // Main loop auto-repeats
{
}
```

Testing LED Brightness with Different Resistors

Remember to disconnect power and the programming cable before you make changes to a circuit. Remember also that the same sketch will run again when you reconnect power, so you can pick up right where you left off with each test.

- ✓ Replace the 220 Ω resistor that goes from pin 8 to the right LED's cathode with a 470 Ω resistor. Note now how brightly the LED glows.
- ✓ Repeat for a 1 kΩ resistor.
- ✓ Repeat once more with a 4.7 kΩ resistor.
- ✓ Replace the 4.7 kΩ resistor with the 220 Ω resistor before moving on to the next portion of this activity.
- ✓ Explain in your own words the relationship between LED brightness and series resistance.

Series Resistance and IR Detection Range

We now know that less series resistance will make an LED glow more brightly. A reasonable hypothesis would be that brighter IR LEDs can make it possible to detect objects that are further away.

- ✓ Re-open and run TestBothIrAndIndicators.
- ✓ Verify that both LED indicator circuits are working properly before continuing.

Your Turn – Testing IR LED Range

- ✓ With a ruler, measure the furthest distance from the IR LED that a sheet of paper can be detected when using 2 kΩ resistors, and record your data in the next table.

- ✓ Replace the 2 kΩ resistors that connect pin 2 and pin 9 to the IR LED anodes with 4.7 kΩ resistors.
- ✓ Determine the furthest distance at which the same sheet of paper is detected, and record your data.
- ✓ Repeat with 1 kΩ resistors, 470 Ω resistors, and 220 Ω resistors. (For the smaller resistor values, they may end up making the detectors so sensitive that they see the table surface in front of your robot. If that happens, set the robot on the edge of the table with the IR detectors pointing off the end and try the distance measurements again.)

Detection Distances vs. Resistance	
IRELD Series Resistance (Ω)	**Maximum Detection Distance**
4700	
2000	
1000	
470	
220	

- ✓ Before moving on to the next activity, restore your IR object detectors to their original circuit, with 2 kΩ resistors in series with each IR LED.
- ✓ Also, before moving on, make sure to test this last change with TestBothIrAndIndicators to verify that both IR object detectors are working properly.

Activity 4: Object Detection and Avoidance

An interesting thing about these IR detectors is that their outputs are just like the whiskers. When no object is detected, the output is high; when an object is detected, the output is low. In this activity, the example sketch RoamingWithWhiskers is modified so that it works with the IR detectors; all it takes is few simple modifications. Here are the steps:

1. Save the sketch RoamingWithWhiskers as RoamingWithIr
2. Add the **`irDetect`** function.

```
int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);
  delay(1);
  int ir = digitalRead(irReceiverPin);
  delay(1);
```

```
  return ir;
}
```

3. Replace these **digitalRead** calls:

```
  byte wLeft = digitalRead(5);
  byte wRight = digitalRead(7);
```

4. …with these calls to **irDetect**:

```
  int irLeft = irDetect(9, 10, 38000);
  int irRight = irDetect(2, 3, 38000);
```

5. Replace all instances of **wLeft** with **irLeft** and **wRight** with **irRight**.
6. Update the **/*…*/** and **//** comments.

Example Sketch – RoamingWithIr

- ✓ Open RoamingWithWhiskers.
- ✓ Save it as RoamingWithIr.
- ✓ Modify it so that it matches the sketch below.
- ✓ Save the sketch and run it on the Arduino.
- ✓ Disconnect the BOE Shield-Bot from its programming cable.
- ✓ Reconnect the battery pack and move the 3-position switch to position 2.
- ✓ Place your BOE Shield-Bot somewhere where it can roam and avoid obstacles.
- ✓ Verify that it behaves like RoamingWithWhiskers (aside from the fact that there's no contact required).

```
/*
 * Robotics with the BOE Shield - RoamingWithIr
 * Adaptation of RoamingWithWhiskers with IR object detection instead of
 * contact switches.
 */

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

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

void setup()                            // Built-in initialization block
{
  pinMode(10, INPUT);  pinMode(9, OUTPUT);   // Left IR LED & Receiver
  pinMode(3, INPUT);  pinMode(2, OUTPUT);    // Right IR LED & Receiver
```

```
  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
{
  int irLeft = irDetect(9, 10, 38000);  // Check for object on left
  int irRight = irDetect(2, 3, 38000);  // Check for object on right

  if((irLeft == 0) && (irRight == 0))   // If both sides detect
  {
    backward(1000);                     // Back up 1 second
    turnLeft(800);                      // Turn left about 120 degrees
  }
  else if(irLeft == 0)                  // If only left side detects
  {
    backward(1000);                     // Back up 1 second
    turnRight(400);                     // Turn right about 60 degrees
  }
  else if(irRight == 0)                 // If only right side detects
  {
    backward(1000);                     // Back up 1 second
    turnLeft(400);                      // Turn left about 60 degrees
  }
  else                                  // Otherwise, no IR detected
  {
    forward(20);                        // Forward 1/50 of a second
  }
}

int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);         // IRLED 38 kHz for at least 1 ms
  delay(1);                             // Wait 1 ms
  int ir = digitalRead(irReceiverPin);  // IR receiver -> ir variable
  delay(1);                             // Down time before recheck
  return ir;                            // Return 1 no detect, 0 detect
}

void forward(int time)                  // Forward function
{
  servoLeft.writeMicroseconds(1700);    // Left wheel counterclockwise
  servoRight.writeMicroseconds(1300);   // Right wheel clockwise
  delay(time);                          // Maneuver for time ms
}

void turnLeft(int time)                 // Left turn function
{
  servoLeft.writeMicroseconds(1300);    // Left wheel clockwise
  servoRight.writeMicroseconds(1300);   // Right wheel clockwise
  delay(time);                          // Maneuver for time ms
```

```
}

void turnRight(int time)                // Right turn function
{
  servoLeft.writeMicroseconds(1700);    // Left wheel counterclockwise
  servoRight.writeMicroseconds(1700);   // Right wheel counterclockwise
  delay(time);                          // Maneuver for time ms
}

void backward(int time)                 // Backward function
{
  servoLeft.writeMicroseconds(1300);    // Left wheel clockwise
  servoRight.writeMicroseconds(1700);   // Right wheel counterclockwise
  delay(time);                          // Maneuver for time ms
}
```

Activity 5: High-performance IR Navigation

The style of pre-programmed maneuvers from the last activity were fine for whiskers, but are unnecessarily slow when using the IR detectors. With whiskers, the BOE Shield-Bot had to make contact and then back up to navigate around obstacles. With infrared, your BOE Shield-Bot will detect most obstacles before it runs into them, and can just find a clear path around the obstacle.

Increase the Sampling Rate to Avoid Collisions

You can reduce all your maneuver durations to 20 ms, which means your sketch will check and recheck for objects almost 50 times per second. (It's actually a little slower, more like 40 times per second, because the code execution and IR detection all takes time too.) As your BOE Shield-Bot navigates, it will execute a series of small turns to avoid an obstacle before it ever runs into it. With that approach, it never turns further than it has to, and it can neatly find its way around obstacles and successfully navigate more complex courses. After experimenting with this next sketch, you'll likely agree that it's a much better way for the BOE Shield-Bot to roam.

Example Sketch – FastIrRoaming

- ✓ Create, save, and run the sketch FastIrRoaming, then test it with the same obstacles you used from the previous activity.

```
/*
 * Robotics with the BOE Shield - FastIrRoaming
 * Adaptation of RoamingWithWhiskers with IR object detection instead of
 * contact switches
 */

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

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

void setup()                            // Built-in initialization block
{
  pinMode(10, INPUT);  pinMode(9, OUTPUT);   // Left IR LED & Receiver
  pinMode(3, INPUT);  pinMode(2, OUTPUT);    // Right IR LED & Receiver

  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
{

  int irLeft = irDetect(9, 10, 38000);  // Check for object on left
  int irRight = irDetect(2, 3, 38000);  // Check for object on right

  if((irLeft == 0) && (irRight == 0))   // If both sides detect
  {
    maneuver(-200, -200, 20);           // Backward 20 milliseconds
  }
  else if(irLeft == 0)                  // If only left side detects
  {
    maneuver(200, -200, 20);            // Right for 20 ms
  }
  else if(irRight == 0)                 // If only right side detects
  {
    maneuver(-200, 200, 20);            // Left for 20 ms
  }
  else                                  // Otherwise, no IR detects
  {
    maneuver(200, 200, 20);             // Forward 20 ms
  }
}

int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);         // IRLED 38 kHz for at least 1 ms
  delay(1);                             // Wait 1 ms
  int ir = digitalRead(irReceiverPin);  // IR receiver -> ir variable
  delay(1);                             // Down time before recheck
  return ir;                            // Return 1 no detect, 0 detect
}

void maneuver(int speedLeft, int speedRight, int msTime)
{
  // speedLeft, speedRight ranges: Backward Linear Stop Linear Forward
  //                                -200    -100....0.....100   200
  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
}
```

How FastIrRoaming Works

This sketch uses the **`maneuver`** function from the TestManeuverFunction sketch. The **`maneuver`** function expects three parameters: **`speedLeft`**, **`speedRight`**, and **`msTime`**. Recall that both speed parameters use 200 for full speed forward, -200 for full speed backward, and values between -100 and +100 for linear speed control. Also, remember **`msTime`** values are 20, so each maneuver executes for 20 ms before returning to the **`loop`** function.

```
void loop()                               // Main loop auto-repeats
{
  int irLeft = irDetect(9, 10, 38000);  // Check for object on left
  int irRight = irDetect(2, 3, 38000);  // Check for object on right

  if((irLeft == 0) && (irRight == 0))   // If both sides detect
  {
    maneuver(-200, -200, 20);           // Backward 20 milliseconds
  }
  else if(irLeft == 0)                  // If only left side detects
  {
    maneuver(200, -200, 20);            // Right for 20 ms
  }
    else if(irRight == 0)               // If only right side detects
  {
    maneuver(-200, 200, 20);            // Left for 20 ms
  }
  else                                  // Otherwise, no IR detects
  {
    maneuver(200, 200, 20);             // Backward 20 ms
  }
}
```

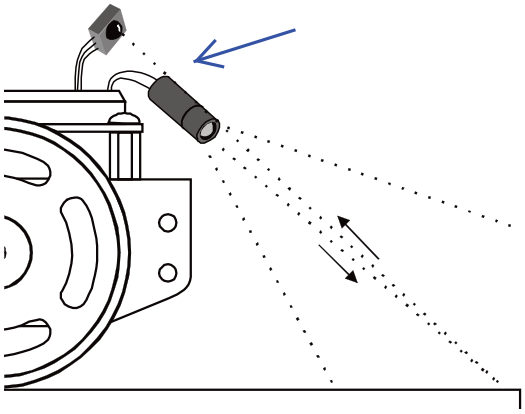
Your Turn

- ✓ Save FastIrRoaming as FastIrRoamingYourTurn.
- ✓ Add code to make the LEDs indicate that the BOE Shield-Bot has detected an object.
- ✓ Try modifying the values that **`speedLeft`** and **`speedRight`** are set to so that the BOE Shield-Bot does everything at half speed. (Remember that 200 and -200 are overkill, so try 50 for half speed forward and –50 for half speed backward).

Activity 6: Drop-off Detector

Up until now, your robot has mainly been programmed to take evasive maneuvers when an object is detected. There are also applications where the BOE Shield-Bot must take evasive action when an object is not detected. For example, if the BOE Shield-Bot is roaming on a table, its IR detectors might be looking down at the table surface. The sketch should make it continue forward so long as both IR detectors can "see" the surface of the table.

- ✓ Disconnect power and programming cable.
- ✓ Point your IR object detectors downward and outward as shown below.



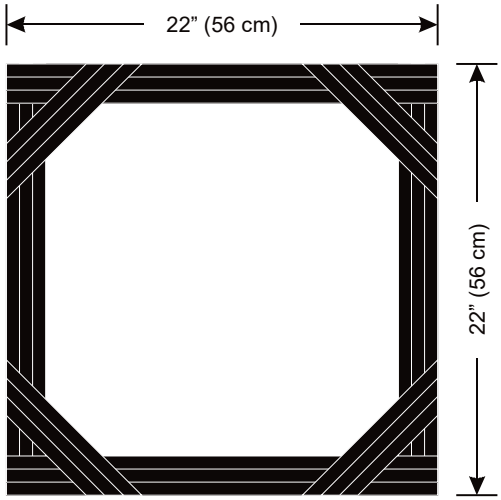
Recommended Materials:

(1) Roll of black vinyl electrical tape, ¾″ (19 mm) wide, or black tempera paint and brush.
(1) Sheet of white poster board, 22 x 28 in (56 x 71 cm).

Simulating a Drop-Off with Poster Board

A sheet of white poster board with a border made of electrical tape or black tempera paint makes for a handy way to simulate the drop-off presented by a table edge, with much less risk to your BOE Shield-Bot.

- ✓ Build a course like one shown below. For electrical tape, make a small test patch on a piece of paper first to make sure the infrared detectors cannot see a reflection from it. If it does not reflect infrared, use three strips edge to edge on the poster board with no paper visible between the strips.



- ✓ Run the sketch IrInterferenceSniffer to make sure that nearby fluorescent lighting will not interfere with your BOE Shield-Bot's IR detectors.
- ✓ Use TestBothIrAndIndicators to make sure that the BOE Shield-Bot detects the poster board but does not detect the electrical tape or paint.

If the BOE Shield-Bot still "sees" the electrical tape too clearly:

- ✓ Try adjusting the IR detectors and LEDs downward at shallower angles.
- ✓ Try a different brand of vinyl electrical tape.
- ✓ Try replacing the 2 kΩ resistors with 4.7 kΩ (yellow-violet-red) resistors to make the BOE Shield-Bot more nearsighted.
- ✓ Adjust the `tone` command with different `frequency` arguments. Here are some arguments that will make the BOE Shield-Bot more nearsighted: 39000, 40000, 41000.

If the Boe Shield-Bot seems nearsighted:

If you are using older IR LEDs, the BOE Shield-Bot might actually be having problems with being too nearsighted**.** Here are some remedies that will increase the BOE Shield-Bot's sensitivity to objects and make it more far sighted:

- ✓ Try replacing the 2 kΩ (red-black-red) resistors with 1 kΩ (red-black-brown) or even 470 Ω (yellow-violet-brown) resistors in series with the IR LEDs instead of 2 kΩ.
- ✓ Try pointing the IR LEDs and detectors downward at a steeper angle so that the bot is looking at the surface right in front of it.

If you want to try a tabletop:

- ✓ Try an electrical tape or paint course first!
- ✓ Remember to follow the same steps you followed before running the BOE Shield-Bot in the electrical tape or paint delimited course!
- ✓ Always be ready to pick your BOE Shield-Bot up from above as it approaches the edge of the table it's navigating. If the BOE Shield-Bot tries to drive off the edge, pick it up before it takes the plunge. Otherwise, your Shield-Bot might become a Not-Bot!
- ✓ Your BOE Shield-Bot may detect you if you are standing in its line of sight. Its current sketch has no way to differentiate you from the table below it, so it might try to continue forward and off the edge of the table. So, stay out of its detector's line of sight as you spot.

Example Sketch: AvoidTableEdge

For the most part, programming your BOE Shield-Bot to navigate around a tabletop without going over the edge is a matter of adjusting the **`if...else if...else`** statements from FastIrRoaming. First of all, instead of backing up, it will need to go forward 20 ms at a time when it sees objects with both detectors. It will also need to turn toward objects instead of away from them, and it will need to turn for more than 20 ms when it sees the drop-off. 375 ms turns seem to work well, but it will be up to you to adjust that value for best performance.

- ✓ Open FastIrNavigation and save it as AvoidTableEdge.
- ✓ Modify the sketch so that it matches the Example Sketch. Pay close attention to the conditions and **`maneuver`** calls in the **`loop`** function. The condition that used to go forward for 20 ms now backs up for 250 ms. Likewise, the condition that used to back up now goes forward for 20 ms. Also, the condition that used to call for a 20 ms right turn now calls for a 375 ms left turn, and the condition that used to call for a 20 ms left turn now calls for a 375 ms right turn.
- ✓ Test the sketch on your electrical tape or paint delimited course.
- ✓ If you decide to try a tabletop, remember to follow the testing and spotting tips discussed earlier.

```
/*
 * Robotics with the BOE Shield - AvoidTableEdge
 * Adaptation of FastIrRoaming for table edge avoidance
 */

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

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

```
void setup()                              // Built-in initialization block
{
  pinMode(10, INPUT);  pinMode(9, OUTPUT);   // Left IR LED & Receiver
  pinMode(3, INPUT);  pinMode(2, OUTPUT);    // Right IR LED & Receiver

  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
{

  int irLeft = irDetect(9, 10, 38000);  // Check for object on left
  int irRight = irDetect(2, 3, 38000);  // Check for object on right

  if((irLeft == 0) && (irRight == 0))   // Both sides see table surface
  {
    maneuver(200, 200, 20);             // Forward 20 milliseconds
  }
  else if(irLeft == 0)                  // Left OK, drop-off on right
  {
    maneuver(-200, 200, 375);           // Left for 375 ms
  }
  else if(irRight == 0)                 // Right OK, drop-off on left
  {
    maneuver(200, -200, 375);           // Right for 375 ms
  }
  else                                  // Drop-off straight ahead
  {
    maneuver(-200, -200, 250);          // Backward 250 ms before retry
  }
}

int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);         // IRLED 38 kHz for at least 1 ms
  delay(1);                             // Wait 1 ms
  int ir = digitalRead(irReceiverPin);  // IR receiver -> ir variable
  delay(1);                             // Down time before recheck
  return ir;                            // Return 1 no detect, 0 detect
}

void maneuver(int speedLeft, int speedRight, int msTime)
{
  // speedLeft, speedRight ranges: Backward Linear Stop Linear Forward
  //                                -200    -100....0....100    200
  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
}
```

How AvoidTableEdge Works

Since AvoidTableEdge is just FastIrRoaming with a modified **`if`**…**`else if`**…**`else`** statement in its **`loop`** function, let's look at the two statements side by side.

```
// From FastIrRoaming

  if((irLeft == 0) && (irRight == 0))
  {
    maneuver(-200, -200, 20);
  }
  else if(irLeft == 0)
  {
    maneuver(200, -200, 20);
  }
  else if(irRight == 0)
  {
    maneuver(-200, 200, 20);
  }
  else
  {
    maneuver(200, 200, 20);
  }
```

```
//From AvoidTableEdge

  if((irLeft == 0) && (irRight == 0))
  {
    maneuver(200, 200, 20);
  }
  else if(irLeft == 0)
  {
    maneuver(-200, 200, 375);
  }
  else if(irRight == 0)
  {
    maneuver(200, -200, 375);
  }
  else
  {
    maneuver(-200, -200, 250);
  }
}
```

In response to **`if((irLeft == 0) && (irRight == 0))`**, FastIrRoaming backs up with **`maneuver(-200, -200, 20)`** because both IR detectors see an obstacle. In contrast, AvoidTableEdge goes forward with **`maneuver(200, 200, 20)`** because both IR detectors see the table, which means it's safe to move forward for another 20 ms.

In response to else **`if(irLeft == 0)`**, FastIrRoaming turns right for 20 ms, taking a step toward avoiding an obstacle on the left with **`maneuver(200, -200, 20)`**. AvoidTableEdge instead turns away from a drop-off that must be on its right. That's because the first **`if`** statement where both IR detectors could see the table was not true. If it was, the **`if`**…**`else if`**…**`else`** statement would not have made it to this **`else if`** condition. It means

the left IR detector does see the table, but the right one does not. So the code makes the robot turn left for 0.375 seconds with `maneuver(-200, 200, 375)`. This should make it avoid a drop-off that must have been detected on the right.

In response to `else if (irRight == 0)`, FastIrRoaming turns left for 20 ms, taking an incremental step toward avoiding an obstacle on the right. At this point in the `if`…`else if`…`else` statement, we know that the right IR detector does see an object, but the left one does not. Again, that's because, it would have handled the condition where both IR detectors see the table with the first if statement. So, the left IR detector does not see the object and turns right for 0.375 seconds with `maneuver(200, -200, 375)`.

Your Turn

The 375 ms turns to avoid the table edge can be adjusted for different applications. For example, if the BOE Shield-Bot is supposed to hug the edge of the table, smaller turns might be useful. In a contest where the BOE Shield-Bot is supposed to push objects out of an area, a larger turn (but not too large) would be better so that it zigzags back and forth across the table.

You can modify the code to make shallower turns by using a smaller `msTime` parameter value in the `maneuver` function calls. For example, if you change the 375 in `maneuver(-200, 200, 375)` to 300, it will make shallower left turns. If you change it to 450, it will make sharper left turns.

- ✓ Modify AvoidTableEdge so that it closely follows the edge of the simulated drop-off course. Adjust the `msTime` parameter values in calls to `maneuver` to make the BOE Shield-Bot execute smaller turns when it sees the drop-off. How small can you make the turn before it tries to fall off?
- ✓ Experiment with pivoting away from the table edge to make the BOE Shield-Bot roam inside the perimeter instead of following the edge.

Chapter 7 Summary

This chapter focused on using a pair of infrared LED emitters and receivers to broadcast and detect reflections of 38.5 kHz modulated infrared light. Using this sensor system for robot navigation used these concepts and skills:

Electronics

- What an infrared LED is, and how to identify its anode and cathode
- What a modulated infrared receiver is, and how to use it in a microcontroller circuit
- What effect resistor values have on LEDs when connected in series

Programming

- How to use the familiar Arduino `tone` function for a new purpose: to modulate an infrared light signal, instead of control the pitch of a sound
- How to repurpose an existing sketch from one sensor system for use with a different sensor system, so long as both sensors have a similar output signal
- How to repurpose an existing sketch designed for a specific behavior (avoiding obstacles) to a new behavior (detecting a drop-off)

Robotics Skills

- Using a pair of infrared emitters and receivers to detect objects for non-contact autonomous sensor navigation

Engineering Skills

- Using human-visible indicators (red LEDs) to broadcast the state of sensors detecting human-invisible conditions (reflected modulated infrared light)
- More practice with subsystem testing and troubleshooting

Chapter 7 Challenges

Questions

1. What is the frequency of the signal sent by `tone(9, 38000, 8)`? How long does this call send the signal for? What I/O pin does the IR LED circuit have to be connected to in order to broadcast this signal?
2. What has to happen after the tone command to find out if the IR receiver detects reflected infrared?
3. What does it mean if the IR detector sends a low signal? What does it mean when the detector sends a high signal?
4. What happens if you change the value of a resistor in series with a red LED? What happens if you change the value of a resistor in series with an infrared LED??

Exercises

1. Modify a line of code in IrInterferenceSniffer so that it only monitors one of the IR detectors.
2. Modify AvoidTableEdge so that when it makes the BOE Shield-Bot back up, it also turns a little so that it doesn't get stuck going straight back at the same drop-off.