Chapter 6: Light Sensitive Navigation with Photoresistors

Light has many applications in robotics and industrial control. Some examples include sensing the edge of a roll of fabric in the textile industry, 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.

There are many different light sensors that serve unique functions. The light sensor in your Boe-Bot kit is designed to detect visible light, and it can be used to make your Boe-Bot detect variations in light level. With this ability, your Boe-Bot can be programmed to recognize areas with light or dark perimeters, report the overall brightness and darkness level it sees, and seek out light sources such as flashlight beams and doorways letting light into dark rooms.

INTRODUCING THE PHOTORESISTOR

The resistors you worked with in previous chapters had fixed values, such as 220 Ω and 10 kΩ. The photoresistor, on the other hand, is a light dependent resistor (LDR). This means that its resistance value depends on the brightness, or illuminance, of the light that shines on its light detecting surface. Figure 6-1 shows the schematic symbol and part drawing for the photoresistor you will use to make the Boe-Bot able to detect variations in light levels.
A photoresistor is a light-dependent resistor (LDR) that covers the spectral sensitivity similar to that of the human eye. In other words, the kind of light that your eye detects is the same kind of light that affects the photoresistor’s resistance. The active elements of these photoresistors are made of Cadmium Sulfide (CdS). Light enters into the semiconductor layer applied to a ceramic substrate and produces free charge carriers. A defined electrical resistance is produced that is inversely proportional to the illumination intensity. In other words, darkness causes more resistance, and brightness causes less resistance.

Illuminance is a scientific name for the measurement of incident light. One way to understand incident light is to think about shining a flashlight at a wall. The focused beam that you see shining on the wall is incident light. The unit of measurement of illuminance is commonly the “foot-candle” in the English system or the “lux” in the metric system. While using the photoresistors we won’t be concerned about lux levels, just whether illuminance is higher or lower in certain directions. The Boe-Bot can be programmed to use the relative light intensity information to make navigation decisions.

ACTIVITY #1: BUILDING AND TESTING PHOTORESISTOR CIRCUITS

In this activity, you will build and test light level sensor circuits with photoresistors. Your light level sensor circuits will be able to detect the difference between shade and no shade. The PBASIC commands for determining whether a shadow is cast over the photoresistor will be very similar to those used to determine whether or not a whisker has contacted an object.

Parts List:

(2) Photoresistors - CdS
(2) Resistors – 2 kΩ (red-black-red)
(2) Resistors – 220 Ω (red-red-brown)
(4) Jumper wires
(2) Resistors – 470 Ω (yellow-violet-brown)
(2) Resistors – 1 kΩ (brown-black-red)
(2) Resistors – 4.7 kΩ (yellow-violet-red)
(2) Resistors – 10 kΩ (brown-black-orange)

Building the Photosensitive Eyes

Figure 6-2 shows the schematic and Figure 6-3 shows the wiring diagram for the photoresistor circuits you will use in this and the next two activities.

- √ Disconnect power from your board and servos.
- √ Build the circuit shown in Figure 6-2, using Figure 6-3 as a reference.
Chapter 6: Light Sensitive Navigation with Photoresistors

Figure 6-2
Schematic – First Light Detection Circuit

Figure 6-3
Wiring Diagrams for the First Light Detection Circuit

Board of Education (left) and HomeWork Board (right).
How the Photoresistor Circuit Works

A BASIC Stamp I/O pin can function as an output or an input. As an output, the I/O pin can send a high (5 V) or low (0 V) signal. Up to this point, high and low signals have been used to turn LED circuits on and off, control servos, and send tones to a speaker.

A BASIC Stamp I/O pin can also function as an input. As an input, the I/O pin does not apply any voltage to the circuit it is connected to. Instead, it just quietly listens without any actual effect on the circuit. In the previous chapter, these input registers stored values that indicated whether or not the whiskers were pressed. For example, the IN7 input register stored a 1 when it sensed 5 V (whisker not pressed), or a 0 when it sensed 0 V (whisker pressed).

An I/O pin set to input doesn't actually need to have 5 V applied to it to make its input register store a 1. Anything above 1.4 V will make the input register for an I/O pin store a 1. Likewise, an I/O pin doesn't need 0 V to make its input register store a 0. Any voltage below 1.4 V will make an input register for an I/O pin store a 0.

When a BASIC Stamp I/O pin is an input, the circuit behaves as though neither the I/O pin nor 220 Ω resistor is present. Figure 6-4 shows the equivalent circuit. The resistance of the photoresistor is shown as the letter R. It could be a few Ω if the light is very bright, or it could be in the neighborhood of 50 kΩ in complete darkness. In a well lit room with fluorescent ceiling fixtures, the resistance could be as small as a 1 kΩ (full light exposure) or as large as 25 kΩ (shade cast around most of the object).

As the photoresistor's resistance changes with light exposure, so does the voltage at Vo; as R gets larger, Vo gets smaller, and as R gets smaller, Vo gets larger. Vo is what the BASIC Stamp I/O pin is detecting when it is functioning as an input. If this circuit is connected to IN6, when the voltage at Vo is above 1.4 V, IN6 will store a 1. If Vo falls below 1.4 V, IN6 will store a 0.
When resistors are connected end-to-end as shown in Figure 6-4 they are **connected in series**, and they can be referred to as series resistors.

When two resistors are connected in series to set a voltage at $V_o$, the circuit is called a **voltage divider**. In this circuit, the value of $V_o$ can be anywhere between $V_{dd}$ and $V_{ss}$. The exact value of $V_o$ is determined by the ratio of $R$ to 2 kΩ. When $R$ is larger than 2 kΩ, $V_o$ will be closer to $V_{ss}$. When $R$ is smaller than 2 kΩ, $V_o$ will be closer to $V_{dd}$. When $R$ is equal to 2 kΩ, $V_o$ will be 2.5 V. If you measure one of the two values ($R$ or $V_o$), you can calculate the other value using one of these two equations.  

$$V_o = 5V \times \frac{2000\Omega}{2000\Omega + R}$$  

$$R = \left( 5V \times \frac{2000\Omega}{V_o} \right) - 2000\Omega$$

1.4 V is called the BASIC Stamp I/O pin’s **threshold voltage**, also known as the I/O pin’s **logic threshold**. When voltage sensed by an I/O pin is above that threshold, the I/O pin’s input register will store a 1. If it is below that value, the I/O pin’s input register will store a 0.

**Detecting Shadows**

Casting a shadow makes the photoresistor’s resistance value ($R$) larger, which in turn makes $V_o$ smaller. The 2 kΩ resistors were chosen to make the value of $V_o$ reside slightly above the BASIC Stamp I/O pin’s 1.4 V threshold in a well lit room. When you cast a shadow over it with your hand, it should send $V_o$ below the 1.4 V threshold.

In a well lit room, both $\text{IN6}$ and $\text{IN3}$ will store the value 1. If you cast a shadow over the photoresistor divider connected to P6, it will then store a 0. Likewise, if you cast a shadow over the photoresistor divider connected to P3, it will cause $\text{IN3}$ to store a 0.
Example Program: TestPhotoresistorsDividers.bs2

This example program is TestWhiskers.bs2 adapted to the photoresistor dividers. Instead of monitoring P5 and P7 as we did with the whiskers, we are now monitoring P3 and P6, which are connected to the photoresistor divider circuits. This program should display a value of 1 on both sides in a well-lit room. When you cast a shadow over one or both of the photoresistors, their corresponding values should change to 0.

√ Reconnect power to your board.
√ Enter, save, and run TestPhotoresistorDividers.bs2.
√ Verify that with no shade, both IN6 and IN3 store the value 1.
√ Verify that you can use your hand to cast a shadow over each of the photoresistors and cause its input register to change from 1 to 0.
√ If you are having difficulty, either with getting a shadow to change the input register to 0 or if the input registers store 0 regardless of whether or not you cast a shadow over them, see the Photoresistor Divider Troubleshooting box after the program listing. Work on it until your hand casting a shadow over the photoresistor reliably changes the state from 1 to 0.

```
' Robotics with the Boe-Bot - TestPhotoresistorDividers.bs2
' Display what the I/O pins connected to the photoresistor voltage dividers sense.
' {$STAMP BS2}
' {$PBASIC 2.5}
DEBUG "PHOTOSENSOR STATES", CR,
  "Left       Right", CR,
  "-------    --------"
DO
  DEBUG CRSRXY, 0, 3,
    "P6 = ", BIN1 IN6,
    "     P3 = ", BIN1 IN3
PAUSE 100
LOOP
```
Chapter 6: Light Sensitive Navigation with Photoresistors

Photoresistor Divider Troubleshooting

General things to verify:

- ✔ Check your wiring and program for errors.
- ✔ Make sure that each component is firmly plugged into its socket.
- ✔ Check the color codes on your resistors. The resistors that connect between Vss and the photoresistors should be 2 kΩ (red-black-red). The resistors connecting P6 and P3 to the photoresistors should be 220 Ω (red-red-brown).

If either the IN3 or IN6 registers showed 0 regardless of whether or not you cast a shadow over them:

- ✔ If the room is dimly lit, consider bringing in some extra lamps. Alternately, you can replace the 2 kΩ resistors with 4.7 kΩ resistors (Yellow Violet Red). This will give your resistor divider better performance under lower lighting conditions. For really low lighting conditions, you can even use the 10 kΩ resistors (brown-black-orange).

If either the IN3 or IN6 registers showed 1 regardless of whether or not you cast a shadow over them:

- ✔ If the room is very brightly lit, and you find yourself having to cup your hand over the photoresistor’s light collecting surface to make the 1 go to 0, you may need to substitute a lower value resistor in place of the 2 kΩ. Try 1 kΩ resistor (brown-black-red), or even a 470 Ω resistor (yellow-violet-brown) if you are outdoors.

Your Turn – Experimenting with Different Voltage Dividers

Depending on the lighting conditions in your robotics area, larger or smaller series resistors in place of the 2 kΩ resistors may improve the performance of your shadow detectors.

- ✔ Remember to disconnect power to your board during each circuit modification.
- ✔ Try replacing the 2 kΩ (red-black-red) resistors with each of the other resistor values you have gathered: 470 Ω, 1 kΩ, 4.7 kΩ, and 10 kΩ.
- ✔ Test each voltage divider combination with TestPhotoresistorDividers.bs2 and determine which resistors work best under your lighting conditions. The best combination is one that’s not overly sensitive, but doesn’t require you to cup your hand over the photoresistor either.
- ✔ Use the resistor combination that you think works best in the next two activities.
ACTIVITY #2: ROAM AND AVOID SHADOWS LIKE OBJECTS

Since the photoresistor dividers behave similarly to whiskers, it’s worth examining what’s involved in adapting RoamingWithWhiskers.bs2 so that it functions with the photoresistor dividers.

Adapting RoamingWithWhiskers.bs2 for the Photoresistor Dividers

All you really have to do is adjust the IF…THEN statements so that they monitor IN6 and IN3, instead of IN7 and IN5. Figure 6-5 demonstrates how to make these changes.

Figure 6-5: Modify RoamingWithWhiskers.bs2 for Use with Photoresistor Dividers

Example Program – RoamingWithPhotoresistorDividers.bs2

Open the program RoamingWithWhiskers.bs2 from page 179, and save it as RoamingWithPhotoresistorDividers.bs2.

Make the modifications shown in Figure 6-5.

Reconnect power to your board and servos.

Run and test the program.
Casting shadows over both photoresistors at the same time can be difficult. When the Boe-Bot is going forward, it is checking the photoresistors around 40 times/second. You will have to move quickly to cast a shadow over both photoresistors between pulses. It helps to move your hand rapidly from no shade to full shade to trigger both photoresistors at once. Alternately, just leave your hand casting shade over both photoresistors while it executes a maneuver. When it returns from the maneuver and checks the photoresistors again, it should recognize that both photoresistors are in shade.

✓ Verify that the Boe-Bot avoids shadows by using your hand to cast a shadow over the photoresistors. Try no shadow, a shadow over the right photoresistor divider (circuit connected to P3), a shadow over the left photoresistor divider (circuit connected to P7), and a shadow over both photoresistor dividers.

✓ Update the comments such as the title and descriptions of reactions to whisker presses to reflect the photoresistor circuit behavior. It should resemble the program below when you are done.

```plaintext
' -----[ Title ]---------------------------------------------------------------
' Robotics with the Boe-Bot - RoamingWithPhotoresistorDividers.bs2
' Boe-Bot detects shadows photoresistors voltage divider circuit and turns
' away from them.
' {$STAMP BS2}                             ' Stamp directive.
' {$PBASIC 2.5}                             ' PBASIC directive.
DEBUG "Program Running!"

' -----[ Variables ]----------------------------------------------------------
pulseCount      VAR     Byte                  ' FOR...NEXT loop counter.

' -----[ Initialization ]-----------------------------------------------------
FREQOUT 4, 2000, 3000                        ' Start/restart signal.

' -----[ Main Routine ]-------------------------------------------------------
DO
  IF (IN6  = 0) AND (IN3 = 0) THEN   ' Both photoresistors detects
    GOSUB Back_Up                    ' shadow, back up & U-turn
    GOSUB Turn_Left                  ' (left twice).
  ELSEIF (IN6  = 0) THEN            ' Left photoresistor detects
    GOSUB Back_Up                    ' shadow, back up & turn right.
    GOSUB Turn_Right
  ELSEIF (IN3  = 0) THEN             ' Right photoresistor detects
    GOSUB Back_Up                    ' shadow, back up & turn left.
    GOSUB Turn_Left
  ELSEIF (IN3  = 0) THEN             ' Right photoresistor detects
    GOSUB Back_Up                    ' shadow, back up & turn left.
    GOSUB Turn_Left
```

" verified"
ELSE
   GOSUB Forward_Pulse
ENDIF

LOOP

' ------[ Subroutines ]--------------------------------------------------------

Forward_Pulse:                     ' Send a single forward pulse.
   PULSOUT 12,650
   PULSOUT 13,850
   PAUSE 20
   RETURN

Turn_Left:                        ' Left turn, about 90-degrees.
   FOR pulseCount = 0 TO 20
      PULSOUT 12, 650
      PULSOUT 13, 650
      PAUSE 20
   NEXT
   RETURN

Turn_Right:                      ' Right turn, about 90-degrees.
   FOR pulseCount = 0 TO 20
      PULSOUT 12, 850
      PULSOUT 13, 850
      PAUSE 20
   NEXT
   RETURN

Back_Up:                           ' Back up.
   FOR pulseCount = 0 TO 40
      PULSOUT 12, 850
      PULSOUT 13, 650
      PAUSE 20
   NEXT
   RETURN

Your Turn – Improving performance

You can improve your Boe-Bot’s performance by commenting some of the subroutine calls that were designed to help the Boe-Bot back away from obstacles and then turn to avoid them. Figure 6-6 shows an example where the two Turn_Left subroutine calls are commented from the IF...THEN statement when the condition is that both photoresistors detect a shadow. Then, when only individual photoresistors detect shadows, the Back_Up subroutine calls are commented so that the Boe-Bot only turns in response to a shadow.
Chapter 6: Light Sensitive Navigation with Photoresistors

Figure 6-6: Modify RoamingWithPhotoresistorDividers.bs2

```
' Excerpt from RoamingWithPhotoresistor Dividers.bs2
' IF (IN6 = 0) AND (IN3 = 0) THEN
  '   GOSUB Back_Up
  '   GOSUB Turn_Left
  '   GOSUB Turn_Left
' ELSEIF (IN6 = 0) THEN
  '   GOSUB Back_Up
  '   GOSUB Turn_Right
' ELSEIF (IN3 = 0) THEN
  '   GOSUB Back_Up
  '   GOSUB Turn_Left
' ELSE
  GOSUB Forward_Pulse
ENDIF
```

```
' Modified excerpt from RoamingWithPhotoresistor Dividers.bs2
' IF (IN6 = 0) AND (IN3 = 0) THEN
  GOSUB Back_Up
  GOSUB Turn_Left
  GOSUB Turn_Left
' ELSEIF (IN6 = 0) THEN
  GOSUB Back_Up
  GOSUB Turn_Right
' ELSEIF (IN3 = 0) THEN
  GOSUB Back_Up
  GOSUB Turn_Left
' ELSE
  GOSUB Forward_Pulse
ENDIF
```

√ Modify RoamingWithPhotoresistorDividers.bs2 as shown in the right side of Figure 6-6.
√ Run the program, and compare the performance.

**ACTIVITY #3: A MORE RESPONSIVE SHADOW CONTROLLED BOE-BOT**

By eliminating the `FOR...NEXT` loops in the navigation subroutines, you can make the Boe-Bot significantly more responsive. This wasn’t really possible with the whiskers, because the Boe-Bot had to back up before turning since it had already made physical contact with the obstacle. When you are using shadows to guide the Boe-Bot, it can check between each pulse to see if the shadow is still detected regardless of whether it’s moving forward or executing a maneuver.

**A Simple Shadow Controlled Boe-Bot**

One interesting form of remote control is to have the Boe-Bot sit still in normal light, then follow a shadow you cast over the photoresistors. It’s kind of a user-friendly way of guiding the Boe-Bot’s motion.

**Example Program – ShadowGuidedBoeBot.bs2**

When you run this next program, the Boe-Bot should stay still when no shadow is cast over its photoresistor dividers. When you cast a shadow over both photoresistors, the
Boe-Bot should move forward. If you cast a shadow over one of the photoresistors, the Boe-Bot should turn in the direction of the photoresistor that senses the shadow.

√ Enter, save, and run ShadowGuidedBoeBot.bs2.
√ Use your hand to cast shadows over the photoresistor dividers.
√ Study this program carefully and make sure you understand how it works. It is very short, yet very powerful.

```
' Robotics with the Boe-Bot - ShadowGuidedBoeBot.bs2
' Boe-Bot detects shadows cast by your hand and tries to follow them.

' {$STAMP BS2}                               ' Stamp directive.
' {$PBASIC 2.5}                              ' PBASIC directive.
DEBUG "Program Running!"
FREQOUT 4, 2000, 3000                        ' Start/restart signal.
DO
  IF (IN6  = 0) AND (IN3 = 0) THEN           ' Both detect shadows, forward.
    PULSOUT 13, 850
    PULSOUT 12, 650
  ELSEIF (IN6  = 0) THEN                     ' Left detects shadow,
    PULSOUT 13, 750                          ' pivot left.
    PULSOUT 12, 650
  ELSEIF (IN3  = 0) THEN                     ' Right detects shadow,
    PULSOUT 13, 850                          ' pivot right.
    PULSOUT 12, 750
  ELSE                                       ' No shadow, sit still
    PULSOUT 13, 750                          ' No shadow, sit still
    PULSOUT 12, 750
  ENDIF
  PAUSE 20                                   ' Pause between pulses.
LOOP
```

**How ShadowGuidedBoeBot.bs2 Works**

The **IF...THEN** statement in the **DO...LOOP** looks for one of the four possible shadow conditions: both, left, right, neither. Depending on which condition is detected, **PULSOUT** commands deliver pulses for one of the following maneuvers: forward, pivot right, pivot left, or sit still. Regardless of the condition, one of the four sets of pulses will be delivered each time through the **DO...LOOP**. After the **IF...THEN** statement, it’s important to remember to include the **PAUSE 20** to ensure the low time between each pair of servo pulses.
Your Turn – Condensing the Program

This program does not need the `ELSE` condition or the two `PULSOUT` commands that follow. If you deliver no pulses, the Boe-Bot will sit still, just as it should when you deliver pulses using 750 for the `PULSOUT Duration` argument.

√ Try deleting (or commenting) this code block.

```plaintext
ELSE
    PULSOUT 13, 750
    PULSOUT 12, 750
√ Run the modified program.
√ Can you detect any difference in the Boe-Bot’s behavior?
```

ACTIVITY #4: GETTING MORE INFORMATION FROM YOUR PHOTORESISTORS

The only information the BASIC Stamp was able to gather from the photoresistor divider circuits was whether the light level was above or below a threshold. This activity introduces a different circuit that the BASIC Stamp can monitor, and actually gather enough information from it to determine relative light levels. The value the BASIC Stamp gets from the circuit will range from small numbers, indicating bright light, to large numbers, indicating low light. This means no more manually replacing series resistors based on light levels. Instead, you will be able to adjust your program to look for different ranges of values.

Introducing the Capacitor

A capacitor is a device that stores charge, and it is a fundamental building block of many circuits. How much charge the capacitor tends to store is measured in farads (F). A farad is a very large value that’s not practical for use with the Boe-Bot. The capacitors you will use in this activity store fractions of millionths of farads. A millionth of a farad is called a microfarad, and it’s abbreviated µF. The capacitor you will use in this exercise stores one one-hundredth of a millionth of a farad. That’s 0.01 µF.
Common capacitance measurements are:

- Microfarads: (millionths of a Farad), abbreviated µF. \( 1 \, \mu F = 1 \times 10^{-6} \, F \)
- Nanofarads: (billionths of a Farad), abbreviated nF. \( 1 \, nF = 1 \times 10^{-9} \, F \)
- Picofarads: (trillionths of a Farad), abbreviated pF. \( 1 \, pF = 1 \times 10^{-12} \, F \)

The 103 on the 0.01 µF capacitor’s case is a measurement picofarads or (pF). 103 is 10, with three zeros added, which is 10,000. Here is how to relate 103 to 0.01 µF.

\[
10,000 = 10 \times 10^3.
\]

\[
(10 \times 10^3) \times (1 \times 10^{-12}) \, F = 10 \times 10^{-9} \, F
\]

which is also \( 0.01 \times 10^{-9} \, F \)

which is 0.01 µF.

Figure 6-7 shows the schematic symbol for a 0.01 µF capacitor along with a drawing of the part in your Boe-Bot parts kit. The 103 marking on the capacitor indicates its value.

**Parts List:**

- (2) Photoresistors - CDS
- (2) Capacitors – 0.01 µF (103)
- (2) Resistors - 220 Ω (red-red-brown)
- (2) Jumper wires

**Figure 6-7**

Capacitor Schematic Symbol and Part Drawing

There may also be 0.1 µF capacitors marked 104 in your kit. Do not use them in these activities.

√ Make sure you have selected the 0.01 µF capacitors (marked 103) for this activity.

The 0.1 µF capacitors can be used in brightly lit areas, but they interfere with the Boe-Bot’s performance in indoor and low lighting activities.

**Rebuilding the Photosensitive Eyes**

The circuit the BASIC Stamp can use to determine light levels is called a resistor/capacitor (RC) circuit. Figure 6-8 shows schematics of the Boe-Bot’s RC light detection circuits and Figure 6-9 shows examples wiring diagrams for the Board of Education and the HomeWork Board.

√ Disconnect power from your board and servos.

√ Build the RC circuits shown in Figure 6-8 using Figure 6-9 as a reference.
Chapter 6: Light Sensitive Navigation with Photoresistors

Figure 6-8
Schematic - Two Photoresistor RC Circuits

For measurement of resistance that varies with light.

Figure 6-9
Wiring Diagrams for Photoresistor Circuits

Board of Education (left) and HomeWork Board (right).
About RC Decay Time and the Photoresistor Circuit

Think of a capacitor in the circuit shown in Figure 6-10 as a tiny rechargeable battery. When P6 sends the high signal, it essentially charges this capacitor-battery by applying 5 V to it. After a few ms, the capacitor charges up to almost 5 V. If the BASIC Stamp program then changes the I/O pin so that it just quietly listens, the capacitor loses its charge through the photoresistor. As the capacitor looses its charge through the photoresistor, its voltage decays, getting lower and lower as it looses charge. The amount of time it takes for the voltage that IN6 senses to drop below 1.4 V depends on how strongly the photoresistor “resists” the flow of electric current supplied by the capacitor. If the photoresistor has a large resistance value due to very dim lighting conditions, the capacitor takes longer to discharge. If the photoresistor has a small resistance value because the light incident on its surface is very bright, it will not resist current very strongly, and the capacitor will lose its charge very rapidly.

Measuring RC Decay Time with the BASIC Stamp

The BASIC Stamp can be programmed to charge the capacitor and then measure the time it takes the capacitor's voltage to decay to 1.4 V. This decay time measurement can be used to indicate the photoresistor's resistance. This resistance in turn indicates how bright the light detected by the photoresistor really is. This measurement requires a combination of the HIGH and PAUSE commands along with a new command called
RCTIME. The RCTIME command is designed to measure RC decay time on a circuit like the one in Figure 6-10. Here is the syntax for the RCTIME command:

```
RCTIME Pin, State, Duration
```

The Pin argument is the number of the I/O pin that you want to measure. For example, if you want to measure P6, the Pin argument should be 6. The State argument can either be 1 or 0. It should be 1 if the voltage across the capacitor starts above 1.4 V and decays downward. It should be 0 if the voltage across the capacitor starts below 1.4 V and grows upward. For the circuit in Figure 6-10, the voltage across the capacitor will start close to 5 V and decay to 1.4 V, so the State argument should be 1. The Duration argument has to be a variable that stores the time measurement, which is in 2 µs units. In this next example program, we’ll measure the RC decay time on the photoresistor circuit connected to P6, which is the photoresistor on the Boe-Bot’s left.

To measure RC decay, the first thing you have to do is make sure you have declared a variable that will store the time measurement:

```
timeLeft    VAR   Word
```

These next three lines of code charge the capacitor, measure the RC decay time and then store it in the timeLeft variable.

```
HIGH 6
PAUSE 3
RCTIME 6,1,timeLeft
```

To get the measurement, the code implements these three steps:

1. Start charging the capacitor by connecting the circuit to 5 V (using the HIGH command).
2. Use PAUSE to give the HIGH command enough time to charge the capacitor in the RC circuit.
3. Execute the RCTIME command, which sets the I/O pin to input, measures the decay time (from almost 5 V to 1.4 V), and stores it in the timeLeft variable.

**Example Program: TestP6Photoresistor.bs2**

√ Reconnect power to your board.
√ Enter, save, and run TestP6Photoresistor.bs2.
√ Cast a shadow over the photoresistor connected to P6 and verify that the time measurement gets larger as the environment gets darker.
√ Point the photoresistor’s light collecting surface directly at an overhead light, or shine flashlight directly at it. The time measurement should get very small. It should then get larger as you gradually direct the photoresistor further away from the light source. It should get even larger if you cast a shadow over it or turn out the lights.

` Robotics with the Boe-Bot - TestP6Photoresistor.bs2`  
` Test Boe-Bot photoresistor circuit connected to P6 and display the decay time.`

`{${STAMP BS2}}`  
` {${PBASIC 2.5}}`  
`timeLeft       VAR     Word`

`DO`  
`HIGH 6`  
`PAUSE 2`  
`RCTIME 6,1,timeLeft`  
`DEBUG HOME, "timeLeft = ", DECS timeLeft`  
`PAUSE 100`  
`LOOP`

**Your Turn**

√ Save TestP6Photoresistor.bs2 as TestP3Photoresistor.bs2.
√ Modify the program so that it performs the RC decay time measurement on the right photoresistor, the one connected to P3.
√ Repeat the shadow and bright light tests with the P3 RC circuit and verify that it works correctly. You will need to modify the Pin arguments for both the HIGH and RCTIME commands, changing them from 6 to 3.

**ACTIVITY #5: FLASHLIGHT BEAM FOLLOWING BOE-BOT**

In this activity, you will test and calibrate your Boe-Bot’s light sensors so that they recognize the difference between ambient light and a directed flashlight beam. You will then program the Boe-Bot to follow the flashlight beam that is pointed at the surface in front of the Boe-Bot.
Extra Equipment

(1) Flashlight

Adjust Sensors to Search for Flashlight Beam

This activity works best if the photoresistors’ light-collecting surfaces are pointing ahead at separate points on the ground about 2 in (5.1 cm) in front of the Boe-Bot.

√ Point the light collecting surfaces of your photoresistors at the surface in front of the Boe-Bot as shown in Figure 6-11.

Figure 6-11: Photoresistor Orientation
Testing Sensor Response to Flashlight Beam

Before you can program the Boe-Bot to turn towards a flashlight beam, you have to know the difference between light readings with and without the flashlight beam shining in the Boe-Bot’s path.

Example Program: TestBothPhotoresistors.bs2

√ Enter, save, and run TestBothPhotoresistors.bs2.
√ Place the Boe-Bot on the surface where it is to follow the flashlight beam. Make sure it is still connected to the serial cable and that the measurements are displaying in the Debug Terminal.
√ Record the values of both time measurements in the first row of Table 6-1.
√ Turn on your flashlight, and focus your beam in front of the Boe-Bot.
√ Your time measurements should now be significantly lower than the first set. Record these new values of both time measurements in the second row of Table 6-1.

| Table 6-1: RC-Time Measurements With and Without Flashlight Beam |
|-----------------|-----------------|----------------|
| **Duration Values** | **Description** |
| timeLeft | timeRight | Time measurements with no flashlight beam (ambient light). |
| timeLeft | timeRight | Time measurements with flashlight beam focused in front of the Boe-Bot. |

' Robotics with the Boe-Bot - TestBothPhotoresistors.bs2
' Test Boe-Bot RC photoresistor circuits.

' {$STAMP BS2}                               ' Stamp directive.
' {$PBASIC 2.5}                              ' PBASIC directive.
timeLeft      VAR     Word                   ' Variable declarations.
timeRight     VAR     Word
DEBUG "PHOTORESISTOR VALUES", CR,       ' Initialization.
"timeLeft  timeRight", CR,
"--------  --------"
DO               ' Main routine.
HIGH 6           ' Left RC time measurement.
PAUSE 3
RCTIME 6,1,timeLeft
HIGH 3                                     ' Right RC time measurement.
PAUSE 3
RCTIME 3,1,timeRight
DEBUG CRSRXY, 0, 3,                             ' Display measurements.
      "",
      "",
      ""
DEC5 timeLeft,
DEC5 timeRight
PAUSE 100

Your Turn
√ Try facing the Boe-Bot in different directions, and repeat your measurements.
√ For better results, you can average your measurements for "flashlight on" and "flashlight off" and replace the values in Table 6-1 with your average values.

Following the Flashlight Beam
You have been using variable declarations up to this point. For example, counter VAR Nib gives the name counter to a particular memory location in the BASIC Stamp’s RAM. After you have declared the variable, every time you use counter in a PBASIC program, it uses the value stored at that particular location in the BASIC Stamp’s RAM.

You can also declare constants. In other words, if you have a number you plan on using in your program, give it a useful name. Instead of the VAR directive, use the CON directive. Here are some CON directives from the next example program:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftAmbient</td>
<td>108</td>
</tr>
<tr>
<td>RightAmbient</td>
<td>114</td>
</tr>
<tr>
<td>LeftBright</td>
<td>20</td>
</tr>
<tr>
<td>RightBright</td>
<td>22</td>
</tr>
</tbody>
</table>

Now, everywhere in the program the name LeftAmbient is used, the BASIC Stamp will use the number 108. Everywhere RightAmbient is used, the BASIC Stamp will use the value 114. Likewise, everywhere LeftBright appears, it’s really the value 20, and RightBright is 22. You will substitute your values from Table 6-1 before running the program.
Constants can even be used to calculate other constants. Here is an example of two constants, named \texttt{LeftThreshold} and \texttt{RightThreshold} that are calculated using the four constants just discussed. The \texttt{LeftThreshold} and \texttt{RightThreshold} constants are used in the program to figure out whether or not the flashlight beam has been detected.

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
 & Average & Scale factor \\
\hline
\texttt{LeftThreshold} & \texttt{LeftBright} + \texttt{LeftAmbient} / 2 & \texttt{5} / \texttt{8} \\
\texttt{RightThreshold} & \texttt{RightBright} + \texttt{RightAmbient} / 2 & \texttt{5} / \texttt{8} \\
\hline
\end{tabular}
\end{center}

The math performed on these constants is an average, and then a scale. The average calculation for \texttt{LeftThreshold} is \texttt{LeftBright + LeftAmbient} / 2. That result is multiplied by 5 and divided by 8. This means that \texttt{LeftThreshold} is a constant whose value is the $\frac{5}{8}$ of the average of \texttt{LeftBright} and \texttt{LeftAmbient}.

\begin{itemize}
\item Math expressions in PBASIC are executed from left to right. First, \texttt{LeftBright} is added to \texttt{LeftAmbient}. This value is divided by 2. The result is then multiplied by 5 and divided by 8.
\item Let’s try this: \texttt{LeftBright + LeftAmbient = 20 + 108 = 128.}
\item \texttt{128 / 2 = 64.}
\item \texttt{64 * 5 = 320}
\item \texttt{320 / 8 = 40}
\end{itemize}

You can use parentheses to force a calculation that is further to the right in a line of PBASIC code to be completed first. For example, you can rewrite this line of PBASIC code:

\begin{verbatim}
pulseRight = 2 - distanceRight * 35 + 750
\end{verbatim}

like this:

\begin{verbatim}
pulseRight = 35 * (2 - distanceRight) + 750
\end{verbatim}

In this expression, 35 is multiplied by the result of (2 - \texttt{distanceRight}), then the product is added to 750.

Example Program: FlashlightControlledBoeBot.bs2

\begin{itemize}
\item Enter FlashlightControlledBoeBot.bs2 into the BASIC Stamp Editor.
\item Substitute your \texttt{timeLeft} measurement with no flashlight beam (from Table 6-1) in place of the value 108 in the \texttt{LeftAmbient CON} directive.
\end{itemize}
√ Substitute your `timeRight` measurement with no flashlight beam in place of the value 114 in the `RightAmbient CON` directive.
√ Substitute your `timeLeft` measurement with focused flashlight beam in place of the value 20 in the `LeftBright CON` directive.
√ Substitute your `timeRight` measurement with focused flashlight beam in place of the value 22 in the `RightBright CON` directive.
√ Reconnect power to your board and servos.
√ Save and then run FlashlightControlledBoeBot.bs2.
√ Experiment and figure out exactly where to focus the light to get the forward, left turn, and right turn maneuvers to execute.
√ Use the flashlight to guide your Boe-Bot through various obstacle courses and maneuvers.

```plaintext
' -----[ Title ]--------------------------------------------------------------
' Robotics with the Boe-Bot - FlashlightControlledBoeBot.bs2
' Boe-Bot follows flashlight beam focused in front of it.

' {$STAMP BS2}                               ' Stamp directive.
' {$PBASIC 2.5}                              ' PBASIC directive.

DEBUG "Program Running!"

' -----[ Constants ]----------------------------------------------------------

' REPLACE THESE VALUES WITH THE VALUES YOU DETERMINED AND ENTERED INTO
' TABLE 6.1.

LeftAmbient   CON     108
RightAmbient  CON     114
LeftBright    CON     20
RightBright   CON     22

'               Average                   Scale factor
LeftThreshold  CON     LeftBright + LeftAmbient / 2 * 5 / 8
RightThreshold CON     RightBright + RightAmbient / 2 * 5 / 8

' -----[ Variables ]----------------------------------------------------------

' Declare variables for storing measured RC times of the
' left & right photoresistors.

timeLeft       VAR     Word
timeRight      VAR     Word

' -----[ Initialization ]------------------------------------------------------
```
FREQOUT 4, 2000, 3000

' -----[ Main Routine ]-----------------------------------------------

DO
  GOSUB Test_Photoresistors
  GOSUB Navigate
LOOP

' -----[ Subroutine - Test_Photoresistors ]---------------------------

Test_Photoresistors:
  HIGH 6
  PAUSE 3
  RCTIME 6,1,timeLeft
  HIGH 3
  PAUSE 3
  RCTIME 3,1,timeRight
RETURN

' -----[ Subroutine - Navigate ]---------------------------------------

Navigate:
  IF (timeLeft < LeftThreshold) AND (timeRight < RightThreshold) THEN
    PULSOUT 13, 850
    PULSOUT 12, 650
  ELSEIF (timeLeft < LeftThreshold) THEN
    PULSOUT 13, 700
    PULSOUT 12, 700
  ELSEIF (timeRight < RightThreshold) THEN
    PULSOUT 13, 800
    PULSOUT 12, 800
  ELSE
    PULSOUT 13, 750
    PULSOUT 12, 750
  ENDIF
  PAUSE 20
RETURN
How FlashlightControlBoeBot.bs2 Works

These are the four constant declarations that you used with your own values from Table 6-1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftAmbient</td>
<td>CON</td>
<td>108</td>
</tr>
<tr>
<td>RightAmbient</td>
<td>CON</td>
<td>114</td>
</tr>
<tr>
<td>LeftBright</td>
<td>CON</td>
<td>20</td>
</tr>
<tr>
<td>RightBright</td>
<td>CON</td>
<td>22</td>
</tr>
</tbody>
</table>

Now that the four constants have been declared, the next two lines average and scale the values to come up with threshold values for the program. These threshold values can be compared with the current `timeLeft` and `timeRight` measurements to determine whether the photoresistors are sensing ambient light or a focused beam.

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftThreshold</td>
<td>LeftBright + LeftAmbient / 2</td>
<td>* 5 / 8</td>
</tr>
<tr>
<td>RightThreshold</td>
<td>RightBright + RightAmbient / 2</td>
<td>* 5 / 8</td>
</tr>
</tbody>
</table>

These variables are used to store the `RCTIME` measurements.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeLeft</td>
<td>VAR</td>
<td>Word</td>
</tr>
<tr>
<td>timeRight</td>
<td>VAR</td>
<td>Word</td>
</tr>
</tbody>
</table>

This is the reset indicator that has been used in most of the programs in this text.

```
FREQOUT 4, 2000, 3000
```

The Main Routine section contains just two subroutine calls. All the actual work in the program occurs in the two subroutines. `Test_Photoresistors` takes the `RCTIME` measurements for both RC photoresistor circuits, and the `Navigate` subroutine makes the decisions and delivers the servo pulses.

```
DO
    GOSUB Test_Photoresistors
    GOSUB Navigate
LOOP
```
This is the subroutine that performs the `RCTIME` measurements on both photoresistor RC circuits. The measurement for the left circuit is stored in the `timeLeft` variable, and the measurement for the right circuit is stored in the `timeRight` variable.

```
Test_Photoresistors:
  HIGH 6
  PAUSE 3
  RCTIME 6,1,timeLeft

  HIGH 3
  PAUSE 3
  RCTIME 3,1,timeRight

RETURN
```

The Navigate subroutine uses an `IF...THEN` statement to compare the `timeLeft` variable against the `LeftThreshold` constant and the `timeRight` variable against the `RightThreshold` constant. Remember, when the `RCTIME` measurement is small, it means bright light is detected, and when it’s large, it means the light is not as bright. So, when one of the variables that stores an `RCTIME` measurement is smaller than the threshold constant, it means the flashlight beam has been detected; otherwise, the flashlight beam has not been detected. Depending on which condition this subroutine detects (both, left, right or neither), the correct navigation pulses is applied, followed by a `PAUSE` before the `RETURN` command exits the subroutine.

```
Navigate:
  IF(timeLeft<LeftThreshold)AND(timeRight<RightThreshold) THEN
    PULSOUT 13, 850
    PULSOUT 12, 650
  ELSEIF (timeLeft < LeftThreshold) THEN
    PULSOUT 13, 700
    PULSOUT 12, 700
  ELSEIF (timeRight < RightThreshold) THEN
    PULSOUT 13, 800
    PULSOUT 12, 800
  ELSE
    PULSOUT 13, 750
    PULSOUT 12, 750
  ENDIF
```
Your Turn – Adjusting the Performance and Changing the Behavior

You can adjust the program’s performance by adjusting the scale factor term in this constant declaration:

\[
\begin{array}{l}
\text{Average} & \text{Scale factor} \\
\text{LeftThreshold} & \text{CON} & \frac{\text{LeftBright} + \text{LeftAmbient}}{2} \times \frac{5}{8} \\
\text{RightThreshold} & \text{CON} & \frac{\text{RightBright} + \text{RightAmbient}}{2} \times \frac{5}{8}
\end{array}
\]

If you change the scale factor from \(\frac{5}{8}\) to \(\frac{1}{2}\), it will make the Boe-Bot less sensitive to the flashlight, which may (or may not) lead to improved flashlight control.

Try different scale factors, such as \(\frac{1}{4}\), \(\frac{1}{2}\), \(\frac{1}{3}\), \(\frac{2}{3}\), and \(\frac{3}{4}\) and make notes about any differences in the way the Boe-Bot responded to the flashlight beam.

By modifying the \texttt{IF...THEN} statement in the example program, you can change the Boe-Bot’s behavior so that it tries to get the light out of its eyes.

Modify the \texttt{IF...THEN} statement so that the Boe-Bot backs up when it detects the flashlight beam with both photoresistor circuits and turns away if it detects the flashlight beam with only one of its photoresistor circuits.

ACTIVITY #6: ROAMING TOWARD THE LIGHT

The example program in this activity can be used to guide the Boe-Bot through exiting a fairly dark room toward a doorway that’s letting in brighter light. It also allows for much better control over the Boe-Bot’s roaming by casting shadows over the photoresistors with your hand.

Readjusting the Photoresistors

This activity works best if the photoresistors’ light collecting surfaces are pointing upwards and outwards.
Point the light collecting surfaces of your photoresistors upward and outward shown in Figure 6-12.

**Figure 6-12: Photoresistor Orientation**

---

**Programming the Roaming Toward the Light Behavior**

The key to roaming toward brighter light sources is going straight ahead when the differences between the photoresistor measurements are small, and turning toward the smaller photoresistor measurement when there is a large difference between the two measurements. In effect, this means the Boe-Bot will turn toward bright light.

Initially this seems like a simple enough programming task; \textbf{IF...THEN} reasoning like this example below should work. The problem is, it doesn’t because the Boe-Bot gets stuck turning left and then right again because the change in \texttt{timeLeft} and \texttt{timeRight} is too large. Each time the Boe-Bot turns a little, the \texttt{timeRight} and \texttt{timeLeft} variables change so much that the Boe-Bot tries to correct and turn back. It never manages to get any forward pulses in.

\begin{verbatim}
IF (timeLeft > timeRight) THEN ' Turn right.
PULSOUT 13, 850
PULSOUT 12, 850
ELSEIF (timeRight > timeLeft) THEN ' Turn left.
\end{verbatim}
Here is another code block that works a little better. This code block fixes the turning back and forth problem under certain conditions. The `timeLeft` variable now has to be larger than `timeRight` by a margin of 15 before the Boe-Bot will apply a left pulse. Likewise, `timeRight` has to be larger than `timeLeft` by 15 before the Boe-Bot adjusts to the left. This gives the Boe-Bot the opportunity to apply enough forward pulses before it has to correct with a turn, but only at certain light levels.

```plaintext
IF (timeLeft > timeRight + 15) THEN ' Turn right.
PULSOUT 13, 850
PULSOUT 12, 850
ELSEIF (timeRight > timeLeft + 15) THEN ' Turn left.
PULSOUT 13, 650
PULSOUT 12, 650
ELSE ' Go forward.
PULSOUT 13, 850
PULSOUT 12, 650
ENDIF
```

The problem with the code block above is that it works under medium dark conditions only. If you take it into a much darker area, the Boe-Bot starts turning back and forth again, and it never applies any forward pulses. If you take it into a brighter area, the Boe-Bot just goes forward, and never makes any adjustments to the left or right.

Why does that happen?

Here is the answer: When the Boe-Bot is in a dark part of a room, the measurement for each photoresistor will be large. For the Boe-Bot to decide to turn toward a light source, the difference between these two measurements has to be large. When the Boe-Bot is in a more brightly lit area, the measurement for each photoresistor will be smaller. For the Boe-Bot to decide to make a turn, the difference between photoresistor measurements also has to be much smaller than it was in the darker part of the room. The way to make this difference respond to the lighting conditions is to make it a variable that is a fraction of the average of `timeRight` and `timeLeft`. That way, it will always be the right value, regardless whether the lighting is bright or dim.
average = timeRight + timeLeft / 2
difference = average / 6

Now, the difference variable can be used in this IF...THEN statement, and it will be a large value when the lighting is low, and a small value when the lighting is bright.

IF (timeLeft > timeRight + difference) THEN  ' Turn right.
PULSOUT 13, 850
PULSOUT 12, 850
ELSEIF (timeRight > timeLeft + difference) THEN ' Turn left.
PULSOUT 13, 650
PULSOUT 12, 650
ELSE                                         ' Go forward.
PULSOUT 13, 850
PULSOUT 12, 650
ENDIF

Example Program – RoamingTowardTheLight.bs2

Unlike RoamingWithPhotoresistorDividers.bs2 on page 201, this program will be very responsive to your hand casting a shadow over the photoresistor, regardless of whether the light is bright or dim. This program does not need to change resistors depending on the lighting conditions. Instead, it takes into account the lighting conditions and the sensitivity adjustment is made in software using the average and difference variables.

For this program to work well, your photoresistors should respond similarly to similar light levels. If the RC circuits are severely mismatched, your measurements from Table 6-1 will be very different under the same lighting conditions. You can correct these mismatched measurements using techniques discussed in Appendix F: Balancing Photoresistors.

This program measures the overall average of timeLeft and timeRight and uses it to set the difference between the timeLeft and timeRight measurements that’s needed to justify delivering a turning pulse.

√ Enter, save, and run RoamingTowardTheLight.bs2
√ Take it to various areas, and let it roam, and verify that you can change its course by casting a shadow over one of the photoresistor RC circuits, regardless of the lighting conditions.
√ Also try placing your Boe-Bot in a room that is poorly lit, but that has light streaming in through a doorway from an adjacent brightly lit room or hallway. See if the Boe-Bot can successfully find its way out the door.

```plaintext
\[
\begin{align*}
\text{Robotics with the Boe-Bot - RoamingTowardTheLight.bs2} \\
\text{Boe-Bot roams, and turns away from dark areas in favor of brighter areas.}
\end{align*}
\]

\[
\begin{align*}
\text{DEBUG "Program Running!"}
\end{align*}
\]

\[
\begin{align*}
\text{Declare variables for storing measured RC times of the left & right photoresistors.}
\end{align*}
\]

\[
\begin{align*}
timeLeft \quad \text{VAR} \quad \text{Word} \\
timeRight \quad \text{VAR} \quad \text{Word} \\
average \quad \text{VAR} \quad \text{Word} \\
difference \quad \text{VAR} \quad \text{Word}
\end{align*}
\]

\[
\begin{align*}
\text{FREQOUT 4, 2000, 3000}
\end{align*}
\]

\[
\begin{align*}
\text{DO}
\end{align*}
\]

\[
\begin{align*}
\text{GOSUB Test_Photoresistors}
\end{align*}
\]

\[
\begin{align*}
\text{For mismatched photoresistors, use Appendix F, uncomment and use next line.}
\end{align*}
\]

\[
\begin{align*}
timeLeft = (timeLeft / 351) + 7 \quad \text{Replace 351 and 7 with your own values.}
\end{align*}
\]

\[
\begin{align*}
\text{GOSUB Average_And_Difference} \\
\text{GOSUB Navigate}
\end{align*}
\]

\[
\begin{align*}
\text{LOOP}
\end{align*}
\]

\[
\begin{align*}
\text{LOOP}
\end{align*}
\]

\[
\begin{align*}
\text{Test_Photoresistors:}
\end{align*}
\]

\[
\begin{align*}
\text{HIGH 6} \quad \text{Left RC time measurement.}
\end{align*}
\]

\[
\begin{align*}
\text{PAUSE 3}
\end{align*}
\]

\[
\begin{align*}
\text{RCTIME 6, 1, timeLeft}
\end{align*}
\]

\[
\begin{align*}
\text{HIGH 3} \quad \text{Right RC time measurement.}
\end{align*}
\]

\[
\begin{align*}
\text{PAUSE 3}
\end{align*}
\]

\[
\begin{align*}
\text{RCTIME 3, 1, timeRight}
\end{align*}
\]

\[
\begin{align*}
\text{RETURN}
\end{align*}
\]
' -----[ Subroutine - Average_And_Difference ]---------------------------------

Average_And_Difference:

average = timeRight + timeLeft / 2
difference = average / 6
RETURN

' -----[ Subroutine - Navigate ]-----------------------------------------------

Navigate:

' Shadow significantly stronger on left detector, turn right.
IF (timeLeft > timeRight + difference) THEN
  PULSOUT 13, 850
  PULSOUT 12, 850
' Shadow significantly stronger on right detector, turn left.
ELSEIF (timeRight > timeLeft + difference) THEN
  PULSOUT 13, 650
  PULSOUT 12, 650
' Shadows in same neighborhood of intensity on both detectors.
ELSE
  PULSOUT 13, 850
  PULSOUT 12, 650
ENDIF
PAUSE 10
RETURN

Why PAUSE 10 instead of PAUSE 20? Because the Test_Photoresistors subroutine has two PAUSE commands adding up to 6 ms plus some extra time to execute the RCTIME commands. Both these factors add to the amount of time between servo pulses, so the PAUSE in the Navigate subroutine has to be reduced. After some trial and error experiments, PAUSE 10 appeared to give the servos the most reliable performance over the widest range of light levels.

Your Turn – Adjusting the Sensitivity to Differences in Light

Right now, the difference variable is the average divided by 6. You can divide average by a smaller value if you want to make the Boe-Bot less sensitive to differences in light or divide it by a larger value if you want to make the Boe-Bot more sensitive to differences in light level.
Instead of the value 6, try dividing the `average` variable by the values 3, 4, 5, 7, and 9.

Run the program and test the Boe-Bot’s ability to exit a dark room with each denominator value.

Decide what the optimum denominator value is.

```
Average_And_Difference:
    average = timeRight + timeLeft / 2
    difference = average / 6

RETURN
```

You can also change the denominator into a constant like this:

```
Denominator CON 6
```

Then, in your `Average_And_Difference` subroutine, you can replace 6 (or the optimum value that you determined) with the `Denominator` constant, like this:

```
Average_And_Difference:
    average = timeRight + timeLeft / 2
    difference = average / Denominator

RETURN
```

Make the changes just discussed, and verify that the program still works correctly.

You can also use one less variable in this program. Notice that the only time the `average` variable is used is to temporarily hold the average value, then it gets divided by `Denominator` and stored in the `difference` variable. The `difference` variable is needed later, but the `average` variable is not. One way to fix this problem would be to simply use the `difference` variable in place of the `average` variable. It will work fine, and you would no longer need the `average` variable. Here is how the subroutine would look:

```
Average_And_Difference:
    difference = timeRight + timeLeft / 2
```
difference = difference / Denominator
RETURN

There is a better way though.

√ Leave the Average_And_Difference routine like this:

Average_And_Difference:

    average = timeRight + timeLeft / 2
    difference = average / Denominator

RETURN

√ Next, make this change in the variable declarations:

Figure 6-13: Modify RoamingTowardTheLight.bs2 to Save a Word of RAM

<table>
<thead>
<tr>
<th>' Unchanged code</th>
<th>' Changed to save Word of RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>average       VAR Word</td>
<td>average       VAR Word</td>
</tr>
<tr>
<td>difference     VAR Word</td>
<td>difference     VAR average</td>
</tr>
</tbody>
</table>

We don’t really need the average variable, but the program will make more sense to someone trying to understand it if we use the word average in the first line and the word difference in the second line. Here is how to create an alias name difference for the average variable.

    difference VAR average

Now, both average and difference refer to the same word of RAM.

√ Test your modified program and make sure it still works properly.
SUMMARY

This chapter focused on measuring differences in light intensity and programming the Boe-Bot to act on these differences. A pair of cadmium sulfide (CdS) photoresistors were used to measure differences in visible light. The CdS photoresistors were first connected to resistors to form voltage dividers, and the BASIC Stamp monitored the voltage at the connection between the photoresistor and the fixed resistor. When this voltage dropped below or rose above 1.4 V the input register for the I/O pin connected to the circuit stored either a 0 or 1. The Boe-Bot was programmed to make decisions using these binary values in a manner similar to the whiskers.

The photoresistor divider technique works so long as the right resistors are chosen and the lighting doesn’t change. However, a much more versatile way of detecting light levels with the BASIC Stamp is to use the CdS photoresistor in an RC circuit, charge the capacitor, and then measure the decay time. RC stands for resistor capacitor, and the capacitor was introduced in this chapter along with a circuit that makes it possible for the BASIC Stamp to measure RC decay time. This is easily done with the BASIC Stamp using the RCTIME command, which is specifically designed for measuring RC decay and growth times.

Constants were introduced as a way to substitute meaningful names for numbers that are used in a PBASIC program. Scaling and averaging were also introduced. Scaling was used to set a threshold value to indicate whether or not a flashlight beam was detected. It was also used to determine the average value of the light levels in an area based on the two photoresistor RC time measurements. This was used to create a threshold that automatically self-adjusted to the overall lighting conditions, eliminating the need to change resistors when the light levels change.

Questions

1. How does the resistance of a photoresistor respond to bright and dim light? What happens if the light levels are between bright and dim?
2. Does an I/O pin have any effect on the circuit when it’s set to input? What causes the input register for an I/O pin to hold a 1 or 0 when it’s set to input?
3. What does threshold voltage mean? What’s the threshold voltage of a BASIC Stamp I/O pin?
4. Referring to Figure 6-4 on page 197, what causes $V_o$ to rise above or fall below a BASIC Stamp I/O pin’s threshold voltage? What is it about the circuit that causes $V_o$ to change value?
5. How does the program ShadowGuidedBoeBot.bs2 differ from the program RoamingWithPhotoresistorDividers.bs2? What does this change in the Boe-Bot’s performance?
6. What is a constant declaration? What does it do? How can you use one in a program?
7. How are math expressions evaluated in PBASIC?
8. What are the two examples in this chapter where PBASIC was used to calculate an average? How are they different? How are they the same?

**Exercises**

1. Calculate $V_o$ for Figure 6-4 on page 197 if $R = 10 \, k\Omega$. Repeat for $R = 30 \, k\Omega$.
2. If $V_o$ in Figure 6-4 on page 197 is 1.4 V, what’s the value of $R$? Repeat for $V_o = 1 \, V$ and $V_o = 3 \, V$.
3. Assume you have three variable values: `firstValue`, `secondValue`, and `thirdValue`. Write a command that takes the average of these three values in a variable named `myAverage`. Write a command that stores 7/8 of the average value in a variable named `myScaledAverage`. Write the variable declarations needed to make your command able to run in a program, first with `myAverage` and `myScaledAverage` as separate variables, then with one of these variable names aliased as the other.

**Projects**

1. With your Boe-Bot’s photoresistors looking down in front of it, develop a program that makes your Boe-Bot recognize the difference between black and white. Find a large white surface and place dark-black sheets of paper on it. Develop a program that makes the Boe-Bot avoid the black sheets of paper. Hints: Make sure to test and understand what the Boe-Bot sees when it is focused on a black sheet of paper and what it sees when it is focused on a white background. Use example programs from the last three activities in this chapter. The RC decay time circuit and programs will be much more helpful for making
the program work than the photoresistor divider techniques. Also, make sure this obstacle course is in a uniformly lit area. Bright sunlight from windows, and shadows cast by onlookers can make the demonstration fail.

2. If you succeeded with project 1, experiment with confining the Boe-Bot so that it can only roam in a space that is enclosed by black sheets of paper.