599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office: (916) 624-8333
Fax: (916) 624-8003

General: info@parallaxinc.com
Technical: stamptech@parallaxinc.com
Web Site: www.parallaxinc.com
Educational: www.stampsinclass.com

Parallax Line Follower Module (#29115)

Introduction

Welcome to one of the most exciting and important aspects of robotics: line following. Line following robots help automate thousands of factories around the world, making the delivery of mail, packages and materials fast and efficient. And line following is not just for small robots or those that work in factories. Scientists and engineers have been experimenting with snow plows and even passenger cars that can follow magnetic lines in "smart" highways. These robotic vehicles can sense the road, obstacles and each other, eliminating traffic snarls and making our highways safer and easier to travel. Someday we'll simply tell our cars where to take us and line following circuitry will help get us there safely and without effort. With the Parallax Line Follower module, you can experiment with this technology today using your BOE-Bot.

Features

The Parallax Line Follower module comes pre-assembled and ready to attach to your BOE-Bot. It uses a multi-sensor array over which you have complete programmatic control. This level of control allows you to develop line following algorithms that are very simple to those that are very advanced. Since the module uses reflective sensors, it contains a threshold adjustment that can tune the sensor array to the ambient light conditions. The Line Follower module connects to the BOE-Bot AppMod socket with a simple ribbon cable and male-male header.

Packing List

The Parallax Line Follower Module (#29115) package should include the following parts (the source code included in this documentation is only available for download from www.parallaxinc.com/linefollower):

- Documentation (these pages)
- Pre-assembled Line Follower module
- Ribbon cable assembly
- 2x10 dual-row header
- (2) 1" female-female hex standoff
- (4) 4/40 x 3/8" screw
- (2) Nylon washer
- Parallax screwdriver

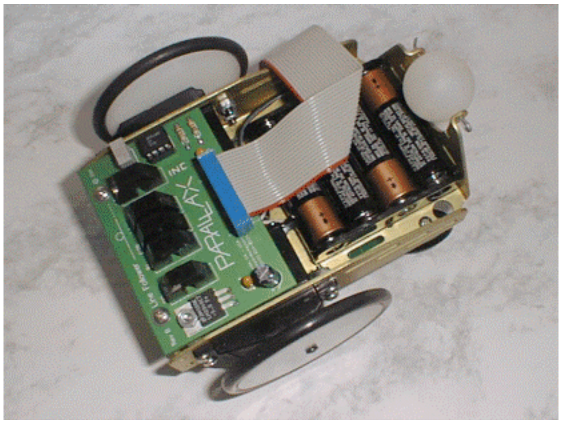
Setting Up

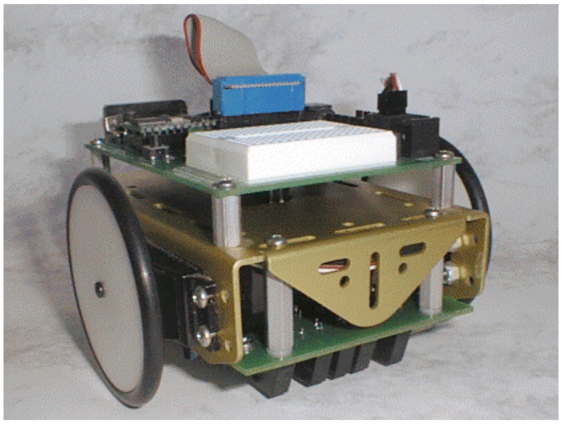
Follow these steps to attach the Parallax Line Follower module to the BOE-Bot (see Figure 1):

1. Clear any program currently in the BASIC Stamp of your BOE-Bot. All pins should be made inputs (Dirs = 0).
2. Disconnect the power source from your BOE-Bot.
3. With the Line Follower module sensors facing up, place a 4/40 screw into one of the mounting holes.
4. Holding the screw in place with a finger, turn the module over. Place a nylon washer over the screw, then thread a 1” hex stand-off onto the screw. Tighten finger-tight.
5. Repeat steps 3 and 4 for the second mounting hole.
6. Carefully place the BOE-Bot on a work surface with the bottom side up.
7. Place the Line Follower module onto the bottom of the BOE-Bot so that the mounting posts are facing forward as the BOE-Bot faces. This will allow the module to rest on the BOE-Bot servo motors and the mounting posts to mate with slots at the front of the BOE-Bot chassis.
8. Holding the Line Follower module in place with your hand, turn the BOE-Bot over. You should see the Line Follower stand-offs through the mounting slots in the BOE-Bot chassis. Use the other two 4/40 screws to secure the Line Follower Module to the BOE-Bot.
9. Holding the BOE-Bot in your hand, turn it over so the bottom side faces up. Hold the ribbon cable in your other hand so that the connector sockets are facing away from you and that the red stripe is on the left.
10. Carefully align the top ribbon connector with the header on the Line Follower module. Press the connector onto the Line Follower header.
11. Make a 90 degree bend to the right in the ribbon cable by folding it under itself. This will allow the ribbon cable to pass behind the BOE-Bot drive wheel.
12. Place the BOE-Bot on a work surface, top side up.
13. Insert the 2x10 dual-row male-male header into the BOE-Bot AppMod socket.
14. Make a 90 degree bend to the left in the ribbon cable by gently folding it under itself. Carefully align the ribbon cable connector with the header in the AppMod socket. Press the ribbon connector onto the header.
15. You may wish to secure the ribbon cable to the side of the robot with a piece of double-sided foam tape.
16. Reconnect power to your BOE-Bot. The Line Follower module is now ready for testing.

Example code with programming explanations are shown on the following pages. Source code for each of these projects is available for download from www.parallaxinc.com/linefollower.

Figure 1: The Mounted Line Follower Module

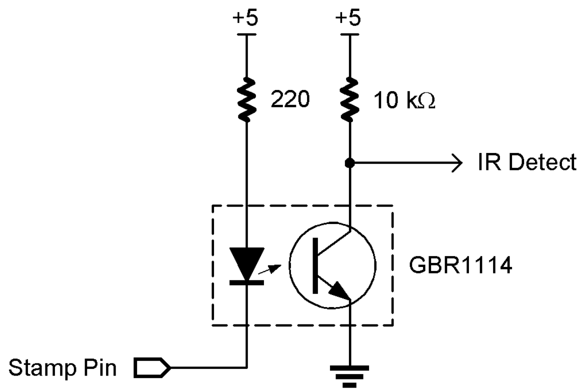




How It Works

The Parallax Line Follower module is composed of two distinct sections: an IR emitter/detector array and a threshold comparator circuit. A simplified diagram of each emitter/detector is shown in Figure 2.

Figure 2: IR Emitter/Detector Circuit



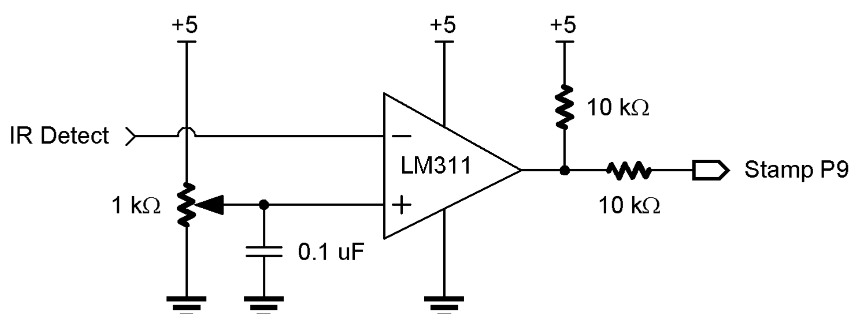
The IR LED is activated by making the associated BASIC Stamp output pin low (see Figure 4 for connections). The active-low configuration is used because the BASIC Stamp can sink more current per pin than it can source. When the LED is active, reflected IR light from the course surface will strike of the IR detector transistor, affecting the current flow through it. More reflected IR causes greater current to flow through the transistor.

Notice that the IR transistor is placed in series with a 10 kΩ fixed resistor and that the IR Detect output is taken at collector of the transistor. As the current flow through the transistor increases the voltage across the 10 kΩ resistor also increases, causing the voltage at the output to decrease. The greater the reflected IR, the lower the output voltage.

This circuit forms a voltage divider with the output dependant on current flow through the transistor. The greater the IR reflection from the course surface, the greater the current flow through the transistor. This causes the voltage at the output to go lower because it “looks” like a smaller resistance. When there is little or no reflection, the current flow through the transistor is reduced making it look like a very large resistance in the circuit, causing the output voltage to increase.

The second portion of the Line Follower circuit is the threshold comparator (see Figure 3). The purpose of this circuit is to compare the output from the IR detector with the level setting on the threshold potentiometer.

Figure 3: Threshold Comparator Circuit



The comparator will output will be high (1) or low (0), depending on which of the two input pins has the higher voltage. If the minus input (voltage from IR detector) is higher than the plus input, the comparator output will go low. If the plus input (threshold pot) is higher than the minus input, then the output will go high. The purpose of the threshold potentiometer is to allow you to adjust for course reflectivity and ambient lighting conditions.

When a detector “sees” a highly reflective surface, the current flow through the transistor is high so the voltage goes low. If it falls below the threshold pot setting, the comparator will output a low (0). If the detector is over a surface that absorbs the IR light from the LED, the current through the transistor will be low, causing detector output voltage to increase. If the output voltage goes higher than the threshold pot setting, the comparator will output a high (1).

In review, reflective surfaces will cause the Line Follower module to output a low (0); non-reflective surfaces will output a high (1). We can modify this behavior with PBASIC code to suit the type of “track”

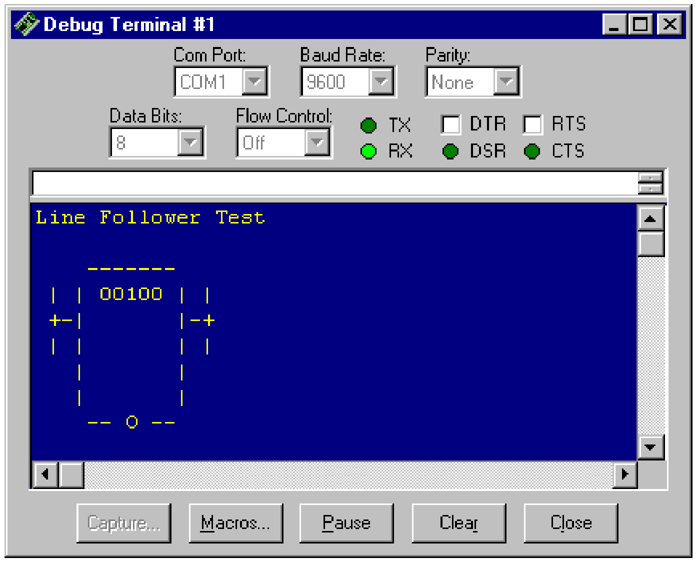
Important Note: The IR LEDs are under direct control of the BASIC Stamp. It is the programmer’s responsibility to ensure that only one LED is lit at a time. Allowing multiple LEDs to light will cause inaccurate and unpredictable results. Figure 4 shows the connections between the BASIC Stamp and the Line Follower module.

Figure 4: Line Follower Module Connections to the BASIC Stamp

LF Function	Stamp Pin	Direction
Outer Right LED	P2	Output
Inner Right LED	P3	Output
Center LED	P4	Output
Inner Left LED	P5	Output
Outer Left LED	P6	Output
Line Detect	P9	Input

Testing And Calibration

Load and run the program called **LF_TEST.BS2** (see Listing 1, starting on page 8). The purpose of this program is to read and display what the Line Follower module "sees" in a **DEBUG** window. Using this program we can test and calibrate the Line Follower module.



On screen will be a small representation of the BOE-Bot with the detector activity displayed in real time. A "1" indicates the presence of the line under the corresponding detector. Of course, you'll need a test track for the program (see to page 14 for instructions).

Or you can print this page and use the test strip below:

You should be able to move the BOE-Bot back and forth across this line and see the corresponding detector bit change on the **DEBUG** screen. Be careful not to lift the BOE-Bot off the paper. The emitter/detector pairs are focused devices and the mounting of the Line Follower module to the BOE-Bot places them at the correct height above the surface for optimal reflectivity.

If output does not change, follow these steps to calibrate the Line Follower module:

1. Using a small screwdriver, turn the threshold pot completely counter-clockwise.
2. Carefully center the middle detector over a solid black line on white paper.
   (the **DEBUG** screen should read: "11111")
3. Turn the threshold pot clockwise just past the point where the middle sensor detects the line
   (the **DEBUG** screen should read: "00100")

Repeat the test and recalibration as necessary until all sensors accurately detect the line with no false positives (detection when no line is present).

Test Program Analysis

Even though the program is simple in its design and purpose, it holds an important piece of code that will be used in all other projects: the reading of the Line Detector module bits.

The program starts by defining some useful constants. Using constant values is a very good habit as it increases code readability and allows a program to be modified easily and more reliably – especially if the same value is used in several places in the program. Constant definitions have been created for **LEDon** and **LEDoff** that will make the program easier to read. In this case **LEDon** has a value of zero since the circuit is configured to turn LEDs on by making the corresponding output pin low.

The next set of values have to do with the kind of course that the BOE-Bot is running. Most of the time the BOE-Bot will follow a black line drawn on a white course. The difficulty with this kind of course, however, is that the large white surface can reflect a lot of additional light – including IR – onto the detectors, causing them to be less sensitive than required to reliably detect the line. For this reason, many robot clubs create line following courses that use a white line on a black background. See page XX for tips on making your own course and dealing with excess light on the sensors.

For this program, the value of **LFmode** (line following mode) is se to **BLine** (black line) which gives it a value of one. The **LFmode** value will be used with the Line Detector output to return the correct value for the specified course type.

The final constant value is called **MoveTo** and has a value of 2. This is a little-used, yet very useful code to use with **DEBUG**. **MoveTo** allows the cursor to be moved to a specific x/y character coordinate in the **DEBUG** window. This technique is great for creating advanced displays.

The Line Follower is initialized by making sure that all of its LEDs are OFF. This is done by setting bits two through six in register **OutL** to one. As soon as the same bits are set in **DirL**, the pins will be made outputs and will go high, causing the LEDs to stay off. Notice that the output bits are set before setting the direction bits. This eliminates any glitches on the outputs when the Stamp is reset and runs the initialization sequence.

The next section of code creates a BOE-Bot diagram in the **DEBUG** window. This diagram will the Line Follower bits in real time and in relation to the correct orientation of the BOE-Bot.

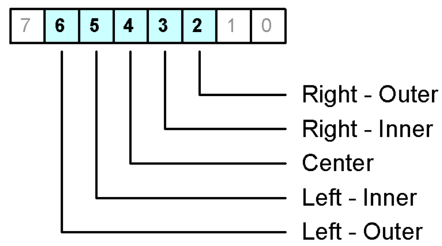
Once in the main body of code, the first thing the program does is call the subroutine **Read_Line_Follower**. This is the heart of the program and will be used by all other line following experiments.

```
Read_Line_Follower:
  lfBits = 0                                    ' clear last reading
  FOR ledPos = 2 TO 6
    OutL.LowBit(ledPos) = LEDon                 ' turn the LED on
    PAUSE 1                                     ' allow sensor to read
    lfBits.LowBit(ledPos) = In9 ^ LFmode        ' record the sensor reading
    OutL = OutL | %01111100                     ' turn LEDs off
  NEXT
  lfBits = lfBits >> 2                          ' shift bits to zero index
  RETURN
```

The subroutine starts by clearing the old reading. This is important because ***lfBits*** is eight bits wide but only five are used. The core of the routine uses a **FOR...NEXT** loop to cycle through all five sensors. The first line of the loop code turns on an LED by setting the corresponding output pin low.

This line of code demonstrates that any nibble, byte or word variable can be treated as an array of bits. Using **.LowBit(*index*)** modifier allows the access (read or write) of any bit in the variable. In this case, the variable is **OutL** since the LEDs are connected to Stamp pins P2 through P6. Figure 5 shows the relationship between the bits in **OutL** and the LEDs in the Line Follower module.

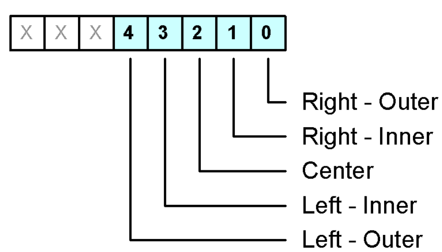
Figure 5: Line Follower LED Control Bits in OutL (P0...P7)



With the LED lit, the program will **PAUSE** for one millisecond to give the IR detector time to respond. The output of the Line Follower module (on P9) is placed into appropriate bit of the variable ***lfBits***. Note that the Exclusive OR (^) operator is used on the bit before it is saved. What this does is allow the program to see the line – be it black-on-white or white-on-black – as a "1." If no line is detected, the output bit will be zero.

After each bit is recorded, the LEDs are switched off. Note that the current value of **OutL** is OR'd (**|**) with %01111100 (all LEDs off). This technique allows the program to uses pins 0, 1 and 7 as outputs without being affected by this subroutine. Before returning to the main loop, the value of ***lfBits*** is shifted two bits to the right. While not required, this process does make display of ***lfBits*** and its use in some algorithms simpler. Figure 6 shows the bits in ***lfBits*** at the end of the **Read_Line_Follower** subroutine.

Figure 6: *lfBits* After Shifting



Back in the main code loop, the **MoveTo** character is used to position the cursor in the **DEBUG** window, then the ***lfBits*** value is displayed with the **BIN5** (binary output, five digits) modifier. With the current Line Follower bits displayed, **GOTO** returns the program to **Main** and process starts over.

Listing 1

```
' -----[ Title ]-----------------------------------------------------------
'
' File...... LF_TEST.BS2
' Purpose... Line Follower Test and Calibrate
' Author.... Parallax
' E-mail.... stamptech@parallaxinc.com

' { $STAMP BS2 }


' -----[ Program Description ]---------------------------------------------
'
' This program is used to test and calibrate the BOE-Bot Line Follower module.


' -----[ Revision History ]------------------------------------------------
'
' 01 DEC 2001 - Version 1.0


' -----[ I/O Definitions ]-------------------------------------------------
'


' -----[ Constants ]-------------------------------------------------------
'
LEDon           CON     0                       ' LF LEDs are active low
LEDoff          CON     1

WLine           CON     0                       ' white line on black field
BLine           CON     1                       ' black line on white field
LFmode          CON     BLine                   ' set pgm for black line

MoveTo          CON     2                       ' move to position character


' -----[ Variables ]-------------------------------------------------------
'
ledPos          VAR     Nib                     ' LED position in lfBits
lfBits          VAR     Byte                    ' line follower input bits


' -----[ EEPROM Data ]-----------------------------------------------------
'


' -----[ Initialization ]--------------------------------------------------
'
Initialize:
  OutL = %01111100                              ' all LF LEDs off
  DirL = %01111100                              ' make pins outputs

Draw_Output_Screen:
  PAUSE 200
  DEBUG "Line Follower Test", CR
  DEBUG CR
  DEBUG "    -------    ", CR
  DEBUG " | |       | | ", CR
  DEBUG " +-|       |-+ ", CR
  DEBUG " | |       | | ", CR
  DEBUG "   |       |   ", CR
```

```
  DEBUG "   |       |   ", CR
  DEBUG "    -- O --    ", CR


' -----[ Main Code ]------------------------------------------------------------
'
Main:
  GOSUB Read_Line_Follower                      ' read the Line Follower
  DEBUG MoveTo, 4, 3, BIN5 lfBits               ' display LF reading
  GOTO Main
  END


' -----[ Subroutines ]----------------------------------------------------------
'
Read_Line_Follower:
  lfBits = 0                                    ' clear last reading
  FOR ledPos = 2 TO 6
    OutL.LowBit(ledPos) = LEDon                 ' turn the LED on
    PAUSE 1                                     ' allow sensor to read
    lfBits.LowBit(ledPos) = In9 ^ LFmode        ' record the sensor reading
    OutL = OutL | %01111100                     ' turn LEDs off
  NEXT
  lfBits = lfBits >> 2                          ' shift bits to zero index
  RETURN
```

Notes:

Simple Line Follower Program

Now that a method has been developed to detect a line, it's a matter of processing and decoding this information in order to cause the BOE-Bot to follow the line. Listing 2 (starts on page 11) is a simple program that will cause the BOE-Bot to follow a ¼" line around a closed course.

The program logic is straightforward:

1. Read the Line Follower module
2. Adjust motor speed (steering) based on Line Follower input
3. Save last Line Follower reading
4. Go to Step 1

Line Following Program Analysis

This line following code uses great deal of the material developed in the test and calibration program. Pin definitions for the BOE-Bot servo motors have been added, as well as speed control values for the servos in the Constants section. These values were determined empirically by running a BOE-Bot in a straight line (both motors at the same speed) and timing a measured distance. By independently controlling motor speed, the BOE-Bot can be caused to move forward, backward or to turn[1].

A single line has been added to the Initialization code; a line that is very important. In the main body of program, the last sensor reading is saved. This will be used to guide the BOE-Bot in the event that an invalid value is returned from the Line Follower module and will keep the BOE-Bot moving in the last known direction. At the beginning of the program there have been no readings, so the value of ***lastBits*** is initialized to %00100. This will cause the BOE-Bot to go straight, even if it is started off the line.

The function of the main loop is to read the Line Follower module and decode its bits, determining the appropriate steering input to give the BOE-Bot. The decoding process is done with the **NCD** operator. **NCD** returns the highest set ("1") bit of a given number, or zero if no bits are set. A non-zero value will be the highest set bit position plus one. Using **NCD**, the value of ***steer*** will fall between zero and five. Then **BRANCH** is used to direct control to the motor control (steering) routines. Once the servos have been updated, the program saves the current reading and continues from the top.

If the BOE-Bot starts -- or somehow manages to roam – off the line (this can happen in very sharp corners), the value returned in ***lfBits*** will be zero. This will cause the **BRANCH** table to send it to the label at **Off_Line**. This section of code restores the last valid sensor reading, then jumps back to the steering control. This will keep the BOE-Bot moving and should allow it to find the line and resume on course.

Programmers with some BOE-Bot experience may notice a lack of any loop padding – time between servo motor updates (typically 20 to 30 ms). The reason is that the **Read_Line_Follower** subroutine takes about 17 milliseconds to execute[2]. This time, combined with other program overhead, provides sufficient delay between servo updates. Keep in mind that adding additional code to the main program loop will tend to slow the BOE-Bot.

---

[1] Refer to ***Robotics!***, Chapter 2, for a complete discussion on BOE-Bot motor control and steering.
[2] Refer to The ***Nuts & Volts of BASIC Stamps***, Column #44, for a routine for code timing.

Listing 2

```
' -----[ Title ]-----------------------------------------------------------------
'
' File...... LF_SIMPLE.BS2
' Purpose... Simple Line Follower
' Author.... Parallax
' E-mail.... stamptech@parallaxinc.com

' { $STAMP BS2 }


' -----[ Program Description ]---------------------------------------------------
'
' This program uses a very simple approach to follow a thin black line on
' a white field.  A test track can be created using a large sheet of white
' construction paper and a wide-tipped black marking pen.  The line width
' should be just as wide as one sensing element.


' -----[ Revision History ]------------------------------------------------------
'
' 01 DEC 2001 - Version 1.0


' -----[ I/O Definitions ]-------------------------------------------------------
'
LMotor          CON     15                      ' servo motor connections
RMotor          CON     14


' -----[ Constants ]-------------------------------------------------------------
'
LEDon           CON     0                       ' LF LEDs are active low
LEDoff          CON     1

WLine           CON     0                       ' white line on black field
BLine           CON     1                       ' black line on white field
LFmode          CON     BLine                   ' set for black line

MStop           CON     750                     ' motor stop
Speed100        CON     125                     ' full speed
Speed075        CON      50                     ' three-quarter speed
Speed050        CON      40                     ' half speed


' -----[ Variables ]-------------------------------------------------------------
'
ledPos          VAR     Nib                     ' LED position in lfBits
lfBits          VAR     Byte                    ' line follower reading
lastBits        VAR     Byte                    ' previous reading
steer           VAR     Nib                     ' steering control


' -----[ EEPROM Data ]-----------------------------------------------------------
'


' -----[ Initialization ]--------------------------------------------------------
'
Initialize:
  OutL = %01111100                              ' all LF LEDs off
  DirL = %01111100                              ' make pins outputs
```

```
  lastBits = %00100                             ' assume starting straight


' -----[ Main Code ]-------------------------------------------------------
'
Main:
  GOSUB Read_Line_Follower                      ' read the Line Follower

Steer_Robot:
  steer = NCD lfBits                            ' get highest "on" bit
  BRANCH steer,[Off_Line, Hard_Right, Right, Straight, Left, Hard_Left]

Save_Last:
  lastBits = lfBits                             ' save last reading
  GOTO Main


' -----[ Subroutines ]-----------------------------------------------------
'
Read_Line_Follower:
  lfBits = 0                                    ' clear last reading
  FOR ledPos = 2 TO 6
    OutL.LowBit(ledPos) = LEDon                 ' turn the LED on
    PAUSE 1                                     ' allow sensor to read
    lfBits.LowBit(ledPos) = In9 ^ LFmode        ' record the sensor reading
    OutL = OutL | %01111100                     ' turn LEDs off
  NEXT
  lfBits = lfBits >> 2                          ' shift bits to zero index
  RETURN


Off_Line:
  lfBits = lastBits                             ' get last known position
  GOTO Steer_Robot


Hard_Right:
  PULSOUT LMotor, MStop + Speed075              ' slow a bit on left
  PULSOUT RMotor, MStop                         ' stop right motor
  GOTO Save_Last


Right:
  PULSOUT LMotor, MStop + Speed100              ' full speed on left
  PULSOUT RMotor, MStop - Speed050              ' slow right motor
  GOTO Save_Last


Straight:
  PULSOUT LMotor, MStop + Speed100              ' both motors forward
  PULSOUT RMotor, MStop - Speed100
  GOTO Save_Last


Left:
  PULSOUT LMotor, MStop + Speed050              ' slow left motor
  PULSOUT RMotor, MStop - Speed100              ' full speed on right
  GOTO Save_Last
```

```
Hard_Left:
  PULSOUT LMotor, MStop                         ' stop left motor
  PULSOUT RMotor, MStop - Speed075
  GOTO Save_Last
```

Notes:

Contest Line Follower Program

Line Following competitions are very popular with robotics clubs. A typical contest course consists of a curvy line that moves across the playing surface, with start and end points are marked with a “T” (see the photo on page 18). Listing 3 (starts on page 15) is a contest-ready line following program.
Contest code logic:

1. Read the Line Follower module
2. If lfBits = %00100 then BOE-Bot is on course
3. If lfBits = %11111 and BOE-Bot has been on course, then stop
4. Adjust motor speed (steering) based on Line Follower input
5. Save Line Follower input
6. Go to Step 1

Contest Code Program Analysis

Line following contests are timed events, so this program allows for a controlled start and will stop when the end of the track is detected. These are the two aspects that differentiate this program from the simple line follower, so that is where the discussion will be focused.

To obtain a controlled start, the program uses a flag value stored in the Stamp’s EEPROM. After download, the value at EEPROM location 0 (**RstValue**) is $FF. Before initializing, the program reads this value, inverts all the bits, then writes it back. If the inverted value is greater than zero, the program runs, otherwise it jumps to the label **No_Run** and stops in a low-power state.

The next time the Stamp is reset (by pressing the Reset button), the reset flag value is read and inverted again. This time the inverted value will be $FF and the program will proceed to initialization and the BOE-Bot will start.

Detection of the end of the course is fairly easy, but not trivial. Simple logic would dictate that a current Line Follower value of %11111 with a previous reading of %00100 would indicate the end of the course. While it does, this method is not always successful in application. If the BOE-Bot hits the end marker just after a curve, the sensor array my not be centered on the course line and the end point will be missed.

This code, then, keeps a flag value called ***onCourse***. After each reading, the value is checked to see if it is %00100. If it is, the onCourse flag is set to **Yes** (true). Now, when the Line Follower returns %11111 the program checks to see if the BOE-Bot has been on the course. If yes, then the end has been found and the motors are stopped.

Before shutting down, zero is written to the **RstValue** EEPROM location. This will allow the BOE-Bot to start again when the Reset button is pressed.

This program uses the same steering control routines, but they have been made somewhat more aggressive than those used in the simple line follower. The turning radius has been shortened to keep the BOE-Bot closer to course center, reducing the time to move around the course. You may want to fine tune the steering values to match the code to the particulars of the course to be run.

Listing 3

```
' -----[ Title ]-----------------------------------------------------------
'
' File...... LF_CONTEST.BS2
' Purpose... Line Follower contest code
' Author.... Parallax
' E-mail.... stamptech@parallaxinc.com

' { $STAMP BS2 }


' -----[ Program Description ]---------------------------------------------
'
' This program is designed to run line follower contests where the BOE-Bot will
' start and stop on a "T" in the track. It implements a controlled start feature
' using the BOE Reset button. Pressing the Reset button starts the BOE-Bot.


' -----[ Revision History ]------------------------------------------------
'
' 01 DEC 2001 - Version 1.0


' -----[ I/O Definitions ]-------------------------------------------------
'
LMotor          CON     15                      ' servo motor connections
RMotor          CON     14


' -----[ Constants ]-------------------------------------------------------
'
LEDon           CON     0                       ' LF LEDs are active low
LEDoff          CON     1

WLine           CON     0                       ' white line on black field
BLine           CON     1                       ' black line on white field
LFmode          CON     BLine                   ' set for black line

MStop           CON     750                     ' motor stop
Speed100        CON     125                     ' full speed
Speed075        CON      50                     ' three-quarter speed
Speed050        CON      40                     ' half speed

Yes             CON     1
No              CON     0


' -----[ Variables ]-------------------------------------------------------
'
temp            VAR     Byte
ledPos          VAR     Nib                     ' LED position in lfBits
lfBits          VAR     Byte                    ' line follower input bits
lastBits        VAR     Byte                    ' last LF input
steer           VAR     Nib                     ' steering control
onCourse        VAR     Bit                     ' on course flag


' -----[ EEPROM Data ]-----------------------------------------------------
'
RstValue        DATA    $FF                     ' $FF = no run
```

```
' -----[ Initialization ]--------------------------------------------------------
'
Run_Check:
  READ RstValue, temp                           ' get reset value
  temp = ~temp                                  ' invert bits
  WRITE RstValue, temp                          ' write inverted bits back
  IF (temp) THEN Initialize                     ' run if inverted > 0

No_Run:
  END                                           ' low power mode

Initialize:
  PAUSE 500                                     ' allow hand to release
  OutL = %01111100                              ' all LF LEDs off
  DirL = %01111100                              ' make pins outputs

  lastBits = %00100                             ' assume straight
  onCourse = No                                 ' we haven't run course yet


' -----[ Main Code ]-------------------------------------------------------------
'
Main:
  GOSUB Read_Line_Follower                      ' read the Line Follower
  IF (lfBits <> %00100) THEN Check_End          ' check for middle sensor
  onCourse = Yes                                ' -- has been on the course

Check_End:
  IF (lfBits <> %11111) THEN Steer_Robot        ' keep steering if not at end
  IF (onCourse = No) THEN Straight              ' still at start
  IF (lastBits <> %00000 ) THEN At_End          ' verify end of course

Steer_Robot:
  steer = NCD lfBits                            ' get highest "on" bit
  BRANCH steer,[Off_Line, Hard_Right, Right, Straight, Left, Hard_Left]

Save_Last:
  lastBits = lfBits                             ' save last reading
  GOTO Main

At_End:
  PULSOUT LMotor, MStop                         ' stop motors
  PULSOUT RMotor, MStop
  Dirs = 0                                      ' turn off outputs
  WRITE RstValue, 0                             ' run again after Reset press
  END


' -----[ Subroutines ]-----------------------------------------------------------
'
Read_Line_Follower:
  lfBits = 0                                    ' clear last reading
  FOR ledPos = 2 TO 6
    OutL.LowBit(ledPos) = LEDon                 ' turn the LED on
    PAUSE 1                                     ' allow sensor to read
    lfBits.LowBit(ledPos) = In9 ^ LFmode        ' record the sensor reading
    OutL = OutL | %01111100                     ' turn LEDs off
  NEXT
  lfBits = lfBits >> 2                          ' shift bits to zero index
  RETURN
```

```
Off_Line:
  lfBits = lastBits                            ' get last known position
  GOTO Steer_Robot


Hard_Right:
  PULSOUT LMotor, MStop + Speed100             ' full speed on left
  PULSOUT RMotor, MStop + Speed100             ' reverse right motor
  GOTO Save_Last


Right:
  PULSOUT LMotor, MStop + Speed100             ' full speed on left
  PULSOUT RMotor, MStop
  GOTO Save_Last


Straight:
  PULSOUT LMotor, MStop + Speed100             ' both motors forward
  PULSOUT RMotor, MStop - Speed100
  GOTO Save_Last


Left:
  PULSOUT LMotor, MStop
  PULSOUT RMotor, MStop - Speed100             ' full speed on right
  GOTO Save_Last


Hard_Left:
  PULSOUT LMotor, MStop - Speed100             ' reverse left motor
  PULSOUT RMotor, MStop - Speed100             ' full speed on right
  GOTO Save_Last
```

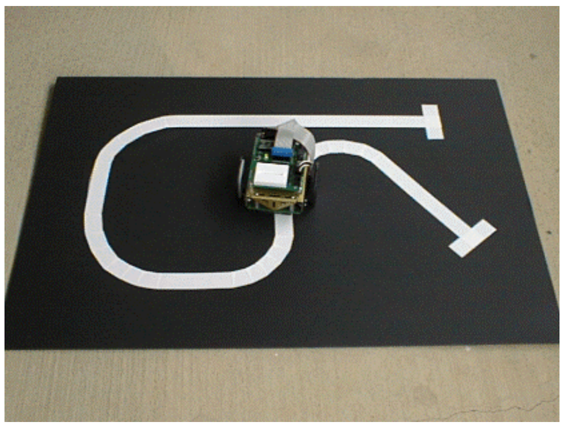
Notes:

Making Tracks

This simplest way to construct a track for your Line Follower is with white art paper and a black, ¼" (6 mm) wide felt tip marking pen. Select a heavy paper stock, one that is opaque, has a matte (non-glossy) finish and is designed to take paint or pens (papers intended for watercolors work well). Draw the line carefully, making sure that it is no wider than one IR sensor. If you prefer a white line on black paper, you can use ¼" (6 mm) model striping tape to make the line.

For very large tracks, some robotics clubs use 1" white athletic tape on black photographers backdrop paper. It may be necessary to make cuts in the tape on the insides of sharp corners. Be sure to burnish the tape down flush so that it doesn't interfere with the sensor array. Figure 7 shows a small practice track made with black foam core and 1" athletic tape.

Figure 7: Practice Track



Troubleshooting

You may find that some track and environment combinations cause trouble for the Line Follower module. Tracks that use a black line on white field that are lit with fluorescent lights or are affected by sunlight can be troublesome. If adjusting the threshold pot does not solve the problem, you can make a skirt from a $^{5}/_{8}$" (16 mm) wide strip of black construction paper. Wrap the strip around the body of the sensors and secure with cellophane tape.

Figure 8: Line Follower With Light Skirt