TCS230 Color Sensor "Tuna Can" Compass

by

Philip C. Pilgrim, Bueno Systems, Inc.

Introduction

This paper describes how to build an electronic compass for a Parallax Boe-Bot using the TAOS/Parallax TCS230 Color Sensor Module. With it, the Boe-Bot is able to know which direction it's pointing at all times, and it can use this information as an aid to navigation. To build the compass, you will need the following materials:

1. Parallax Boe-Bot, Part #28132.
2. TAOS/Parallax TCS230 Color Sensor Module, Part # 30054.
3. Parallax TCS230 Boe-Bot Mounting Kit, Part # 28100.
4. Empty 6 oz. steel tuna can.
5. Two permanent magnets.
6. Thumbtack.
7. Packaging tape.
8. White, sticky-backed label stock or double-stick tape.
9. Light (3-in-1®) lubricating oil.
10. A handheld compass and declination data for your locale, or some other way of determining true north.

You will also need the following tools:

1. Center-finder or dividers and a marker or scribe.
2. Hammer and center punch.
3. Drill with 1/16" bit.
4. PC with a color printer and available serial port.
5. Paper punch.
6. (Optional) Hot-melt glue gun and glue.

Building the Compass Card

The rotating disc inside a navigation compass is called a compass "card". Our compass card is going to be a tuna can. To begin, obtain an empty 6 oz. steel tuna can, from which the lid and label have been removed. The examples shown here use a Star-Kist® can. You're going to drill a small hole in the bottom of the can, in the center; but first, you have to find the center. The illustrations at the right show one such method, using a center-finder. Because the bottom is recessed, it's difficult to scribe exactly along the straightedge. But by holding the marker at a consistent angle and making marks from each side of the can, you will form a small square whose center is easy to eyeball. You can also use a pair of dividers, as follows: First measure the inside diameter of the can. Set the dividers to half the amount. In the inside of the can now, with one point of the dividers positioned as far to the edge as you can, use the other point to scribe a mark across the center. Repeat this three more times, rotating the can 90 degrees each time. You will end up with a small square in the middle whose center you can then eyeball.

With your marks on top, put a block of wood under the can for support. Using the hammer and center punch, make a small

indentation. This will make drilling a hole more accurate. Finally, with the wood block still in place, drill a 1/16" hole at the punched location.

Next, take the thumbtack and insert it from the top through the hole you just drilled. Secure it in place with a square piece of packaging tape. Now, invert the can so you can see the point of the thumbtack protruding into the can.

Print the included file **ColorWheel.jpg** with a color printer on white label stock or plain paper. You can do this from most photo, drawing, or paint programs. Print it so it's about 1.75" in diameter (150 dpi). If you used plain paper, now would be a good time to adhere the double-stick tape. Without peeling the backing off, use a paper punch to punch a hole in the center. Now peel of the backing and adhere the color wheel to the inside of the can so the thumbtack protrudes through the center of the hole you just punched.


Finally, obtain two small identical permanent magnets. Long, skinny ones like the ones shown at the right are best, but round ones will also work. Just be sure the poles are oriented perpendicular to the largest surface and not parallel to it. Attach the magnets to opposite sides of the can as shown to the right. Make sure the poles are identically aligned and not opposing each other. The easiest way to do this is, with the magnets stuck together, separate them using both hands and, without rotating them, move them to opposite sides of the can and attach them. If the poles oppose each other, the compass just won't work. Also, with the can inverted as shown, make sure the magnets are near the top. This helps reduce the influence of ferrous materials present on the Boe-Bot, principally the DB9 serial connector.


Putting the pieces together

Assemble the TCS230 Color Sensor Module boards to the Boe-Bot Mounting Kit parts as shown. Plug the assembled unit into the AppMod socket. Apply a *very small* amount of light lubricating oil to the socket of the socket-head cap screw, being careful to avoid the lens and the electronics beneath. Invert the Boe-Bot and shake off the excess. Now, position the tuna can so the point of the thumbtack mates with the socket-head cap screw. The can should balance and rotate freely. You can trim the balance by shifting the magnets a little along the circumference of the can -- just as you would with tire weights when balancing a wheel on your car. Once balanced, you can tack the magnets in place, if you want, with hot-melt glue. When replaced on the Boe-Bot, the can should always return to the same position when displaced, with one of the magnets pointing to magnetic north. Notice that it may oscillate a number of times before settling down. This is because, unlike most navigation compasses that are liquid-filled for damping, this one is undamped. Nonetheless, it will still be useful for navigating your Boe-Bot!

Calibrating the Compass

Calibration is done using a program which runs on your PC and is connected to your Boe-Bot via a serial cable. First, however, you
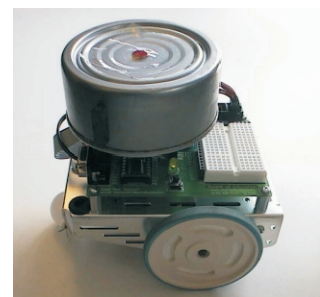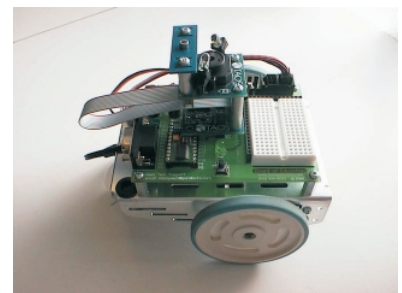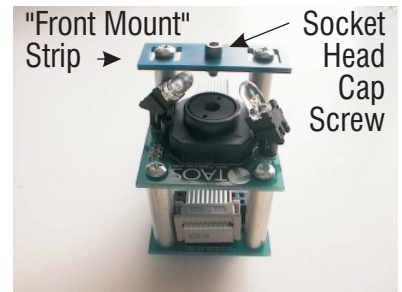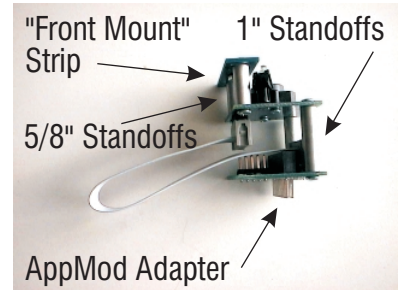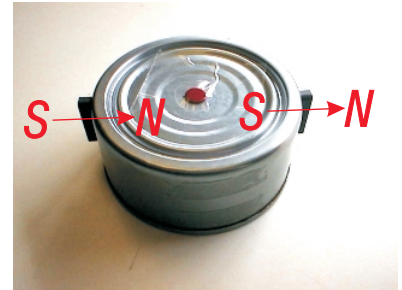
need to load the Boe-Bot-resident BASIC Stamp code. Open your Stamp editor and load the included program **TCS230Monitor.bs2**. When running, you should see the TCS230's LEDs come on and data being displayed on you debug window. Now close your debug window and start the PC host program, **TCS230Compass.exe** (**TCS230Compass.pl** for you Linux users). You should see a screen similar to the one below:
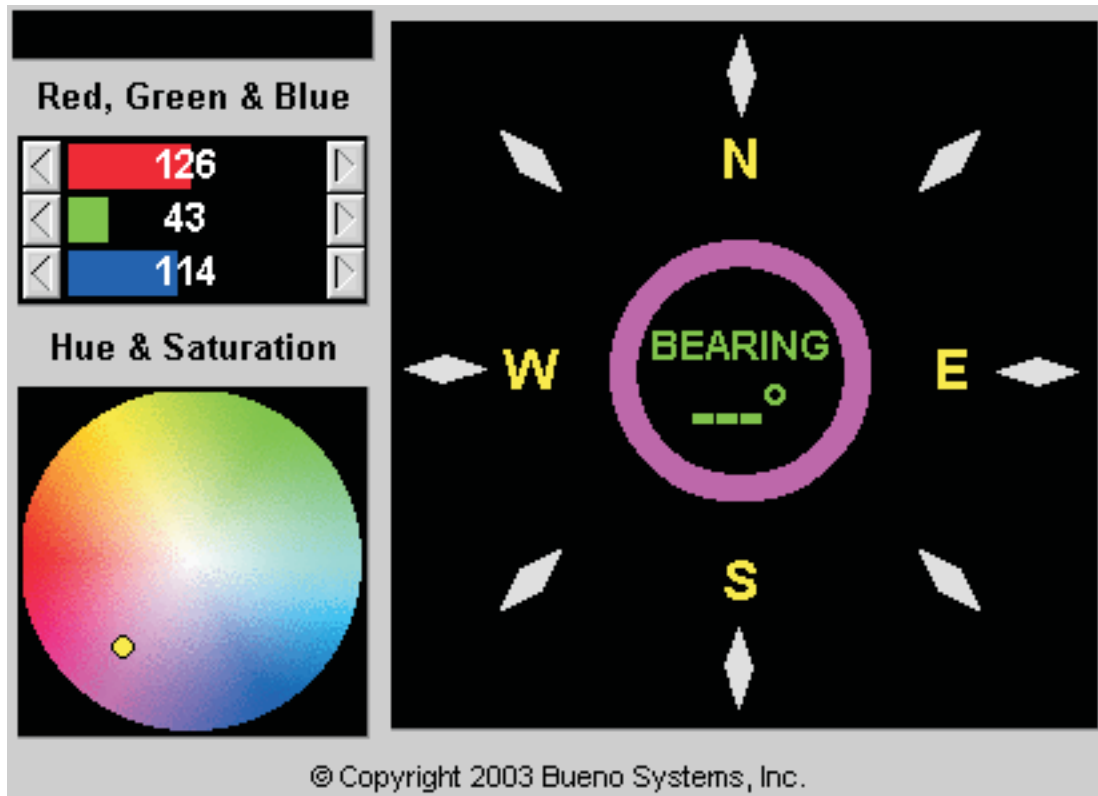
If you don't get a display like this, make sure of these three things: 1) that the program **TCS230Monitor.bs2** is running on the Boe-Bot, 2) that the debug window is closed, and 3) that the serial cable from your PC is connected to the Boe-Bot.

You will notice that as you spin the tuna can the displays in the left-hand column change. These displays reflect the different colors the TCS230 sees as the color wheel spins above it. The red, green, and blue bars show the color breakdown in the usual RGB color space. Take some time now to rotate the can slowly to see where each of these components peaks. You want each one to peak just below 255 and not to saturate. If it does saturate, the message box at the top will tell you so. You can adjust the response of each color using the arrow buttons on either side: left for lower, right for higher. Adjust each color now for the best peak response.

Another thing you will notice is that as you rotate the can, the little yellow dot in the hue and saturation display will rotate around the circle. This circle, in fact, is similar to the color wheel in the tuna can, so that should come as no surprise. Both the color wheel and this circle have colors arranged around the circumference by *hue*, which is just one component in the *hue/saturation/value* color space. *Hue* represents the

primary spectral component of the color, i.e. the location in the spectrum where the color has its peak. Of course, in reality, there is no continuum between red and blue, their being at opposite ends of the visible spectrum, but we perceive such a continuum, nonetheless, and representing *hue* in a circle like this is a way to represent that perception. *Saturation* is a measure of how "pure" the color is. What we usually call "bright red" or "scarlet" is 100% saturated. "Pink" is just red with a lower *saturation*, i.e. it has components of blue and green which make the red look "whiter". You will notice in the *hue* and *saturation* display that the center of the circle is white. This represents a *saturation* of zero, increasing to 100% as you approach the circumference. The third component, *value*, is a measure of the color's brightness. "Maroon", for example, is just red with a lower *value*.

The reasons for choosing the HSV (*hue/saturation/value*) color space for the compass are:

> 1. *Hue*, lying in a circular continuum dovetails nicely with direction which also lies in a circular continuum.
> 2. By isolating *value* and ignoring it, we can filter out the brightness variations that the TCS230 sees when the can teeter-totters on its axis rather than rotating.

Ideally, the colors that the TCS230 sees when looking at the color wheel would be 100% saturated. If that were the case, the yellow dot would always be on the edge of the circle. But it's not. In fact, it seldom, if ever, reaches the edge, oscillating in and out within the circle's interior. This is because a color printer cannot print a "pure" color. So, when the TCS230 looks at something that's been printed, it will never see a pure, 100% saturated color.

Another thing you will notice is that as you spin the can, even though the can is rotating at a uniform rate, the yellow dot seems to speed up and slow down, almost getting stuck at times in certain parts of the circle. This, again, has to do with the printer not being able to produce perfect representations of the colors it's asked to print. When what are supposed to be different hues look alike to the TCS230 because the printer printed them so similarly, they are these regions where the yellow dot will stick. Now this can be a problem for the compass, because we don't want several points on the color wheel looking like the same direction! What if it were possible to make another color wheel which minimized these "regions of confusion"? Well, it is possible, and that's what you're going to do next.

Print out the next page in this document and place it on the floor, well away from computer monitors and other sources of magnetic interference. You may want to tape it down to hold it in place. Don't worry about which direction north is; it's not important yet. Set the Boe-Bot on top of the paper so it points to the top of the page, aligned with the vertical-most rectangle -- but keep it connected to your PC. Wait for the can to settle, then click the diamond in the compass display corresponding to "north". It will change from gray to a fully-saturated version of the color the TCS230 is seeing. Notice

North
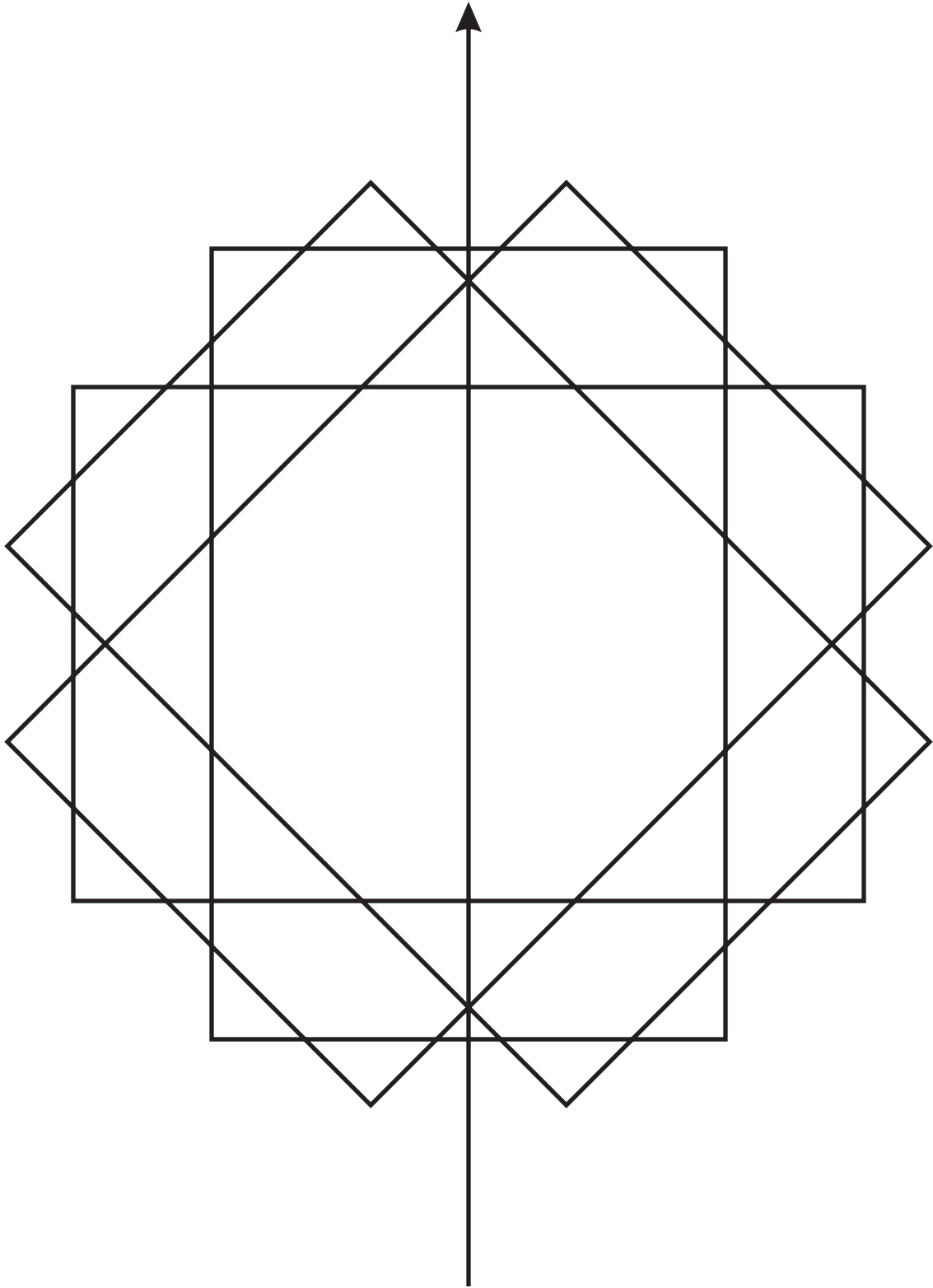
also that the ring surrounding the "Bearing" label changes color to match what the sensor is seeing. Now rotate the Boe-Bot right by 45 degrees to align with the next rectangle on the paper. Click on the diamond corresponding to "northeast". Continue this all the way around the circle. When you've clicked the last diamond, a large red needle will appear to point in the direction the Boe-Bot is headed, and the bearing indicator will change to an actual number, representing a direction. Ignore this stuff for now, though. It's not going to be accurate.

You can redefine any direction color you like by clicking on its corresponding diamond, or undefine it by right-clicking on it. When all eight directions are defined, you can save the definitions to a file for later recovery, if you like, by selecting **File -> Save Compass Colors**. Once you've done that, select **File -> Create New Color Disc**. You will be asked for a file name for the new jpeg file. Then the program will begin the process of creating a new color disc, keeping you abreast of its progress in the message. When completed, the new file will be written.

Open the file you just created and look at it. Do you see the difference in color distribution? If, for example, you were getting "stuck" in blue before, you will notice that blue now takes up much less of the overall circumference. Now print this new color wheel, as you did before, at about 1.75" (150 dpi) in diameter. Replace the original color wheel in the can with this new one (or just stick the new one over the original) and replace the can on the Boe-Bot. This time when you spin the can, the yellow dot should rotate about the center of the *hue/saturation* circle much more smoothly than before. You are now ready for a final compass calibration.

Place the calibration sheet on the floor again, this time making sure that "North" points to true north. You may need to use a real compass for this, remembering to correct for the declination in your area. ("Declination" is the difference between true north and magnetic north and is given as degrees east of north or degrees west of north.) Make sure all the diamonds are undefined by right-clicking on those that show a color other than gray. Now go through the entire calibration process again, clicking on each diamond in turn until you are done. This time, when you're finished, the red needle and the bearing number should correspond to reality. If they do, you've successfully calibrated your compass!

A compass isn't much good if you can't use it on an untethered robot, though. What's needed is a BASIC Stamp program that uses the calibration information to calculate direction on its own. Well, you're in luck! By selecting **File -> Create BASIC Stamp Code**, you can generate just such a program and save it for later upload to the Boe-Bot.

Boe-Bot Navigation

Close the calibration program, start the BASIC Stamp editor, and open the program you just created. It should look something like the one beginning on the following page. There's a subroutine in this program called **GetDir** which returns a number ranging from 0

Listing 1. Sample BASIC Stamp program output from calibration program.

```
'{$PBASIC 2.5}
'Define port pins

EN       con    1
A0       con    2
S0       con    3
S1       con    4
S2       con    5
S3       con    6
nLED     con    7
OUT      con    8


'Define variables for color results.

pRED     var    byte
pGREEN   var    byte
pBLUE    var    byte

RED      var    word
GREEN    var    word
BLUE     var    word
HUE      var    byte
cmax     var    byte
cmin     var    byte
DIR      var    word

'Program starts here.

Start: low     A0                      'For Unit 0; use high for Unit 1.
       high    S0                      'Maximum output rate.
       high    S1                      '          "
       low     nLED                    'Turn on LED.
       gosub   GetPeriods              'Get count periods for RGB.


'-------------------YOUR CODE GOES BELOW THIS LINE---------------------

MainLp: gosub   GetDir                 'Main loop
       debug   dec3 DIR, cr            'SAMPLE CODE: Read & echo direction.
       goto    MainLp                  'Back for another.

-------------------YOUR CODE GOES ABOVE THIS LINE---------------------


'GetDir: Read the current direction of travel.

GetDir: gosub   Color                  'Read the compass color.
       gosub   RGBtoHue                'Convert RGB to hue.
       read    HUE + 3, DIR            'Use hue to look up direction.
       return                          '  and return.

'Color: Read all three color components.

Color: high    EN                      'Enable output/turn on LEDs.
       low     S2                      'Address the red output.
       low     S3
       count   OUT, pRED, RED          'Read the red component.
       high    S3                      'Address the blue output.
       count   OUT, pBLUE, BLUE        'Read the blue component.
       high    S2                      'Address the green output.
       count   OUT, pGREEN, GREEN      'Read the green component.
       low     EN                      'Disable output/turn off LEDs.
       return                          'Return.

'RGBToHue: Extract Hue from RGB components.

RGBToHue:
       RED = RED max 255
       GREEN = GREEN max 255
       BLUE = BLUE max 255
       cmax = RED min GREEN min BLUE   'Find maximum RGB component.
       cmin = RED max GREEN max BLUE   'Find minimum RGB component.
       if cmin = cmax then             'Hue is undefined if SAT is zero.
         HUE = 0
         return
```

```
        endif
        if (cmax = RED) then            'Find hue in color circle.
          HUE = abs(GREEN - BLUE) * 43 / (cmax - cmin)
          if BLUE > GREEN then HUE = 256 - HUE
        else      if (cmax = GREEN) then
            HUE = abs(BLUE - RED) * 43 / (cmax - cmin)
            if RED > BLUE then HUE = 256 - HUE
            HUE = HUE + 85
          else
            HUE = abs(RED - GREEN) * 43 / (cmax - cmin)
            if GREEN > RED then HUE = 256 - HUE
            HUE = HUE + 170
          endif
        endif
        return                          'Return.

'GetPeriods: Read count periods for RGB from EEPROM.

GetPeriods:
        read    0, pRED
        read    1, pGREEN
        read    2, pBLUE
        return

'Data: Count periods for RGB components.

Periods:
        data 10, 8, 7

'Data: Lookup table for converting Hue to Direction.

Directions:
        data    28, 26, 24, 23, 21, 19, 17, 16
        data    14, 12, 10,  8,  7,  5,  3,  1
        data     0, 255, 254, 253, 252, 251, 250, 249
        data    248, 247, 246, 245, 245, 244, 243, 242
        data    241, 240, 239, 238, 237, 236, 235, 234
        data    234, 233, 232, 231, 230, 229, 228, 227
        data    226, 225, 224, 224, 223, 222, 222, 221
        data    221, 220, 220, 219, 219, 218, 218, 217
        data    217, 216, 216, 215, 215, 214, 214, 213
        data    212, 212, 211, 211, 210, 210, 209, 209
        data    208, 208, 207, 207, 206, 206, 205, 205
        data    204, 204, 203, 203, 202, 201, 201, 200
        data    200, 199, 199, 198, 198, 197, 197, 196
        data    196, 195, 195, 194, 194, 193, 193, 192
        data    192, 190, 188, 186, 184, 183, 181, 179
        data    177, 176, 174, 172, 170, 168, 167, 165
        data    163, 161, 160, 159, 158, 157, 156, 155
        data    154, 153, 152, 151, 150, 149, 149, 148
        data    147, 146, 145, 144, 143, 142, 141, 140
        data    139, 138, 138, 137, 136, 135, 134, 133
        data    132, 131, 130, 129, 128, 128, 126, 125
        data    123, 122, 121, 119, 118, 116, 115, 114
        data    112, 111, 109, 108, 107, 105, 104, 102
        data    101, 100,  98,  97,  96,  95,  94,  94
        data     93,  92,  92,  91,  90,  90,  89,  88
        data     88,  87,  87,  86,  85,  85,  84,  83
        data     83,  82,  81,  81,  80,  80,  79,  78
        data     78,  77,  76,  76,  75,  74,  74,  73
        data     72,  72,  71,  71,  70,  69,  69,  68
        data     67,  67,  66,  65,  65,  64,  64,  62
        data     60,  58,  56,  54,  52,  50,  48,  46
        data     44,  42,  40,  38,  36,  34,  32,  30
        end
```

(north), through 64 (east), 128 (south), 192 (west), to 255, representing the direction the Boe-Bot is pointed. This program just reads the direction and echoes it to the debug window. You may want to upload the program to the Boe-Bot now to verify that the directions are correct.

Finally, let's use the compass to write a simple Boe-Bot roaming program. This program will cause the Boe-Bot to travel north for a count of 200, then west for 200, south for 200, then east for 200, then stop -- hopefully where it started. Try it and see how closely the finish point matches the start point. Once you've generated the base code for your compass, paste the following code between the lines where user code is allowed. This code is also found in the included file **CompassDemo.bs2**.

Listing 2. Sample Boe-Bot roaming code using the compass.

```
'This is NOT a complete program. Paste the code between lines between the same lines
'in the code generated by the compass calibration program.

'-----------------------YOUR CODE GOES BELOW THIS LINE-----------------------

right      con     12                    'Servo controller: Left wheel.
left       con     13                    'Servo controller: Right wheel.

i          var     byte                  'Counter.
n          var     byte                  'Count (distance) to move.
pref       var     byte                  'Preferred direction.
dif        var     byte                  'Difference betw/ preferred & actual directions.
pDIR       var     byte                  'Previous measured direciton.
lpulse     var     word                  'Pulse duration for left wheel.
rpulse     var     word                  'Pulse duration for right wheel.

           n = 200                       'Set count (distance) to 200.
           pref = 0                      'Set preferred direction to 0 (north).
           gosub Turn                    'Turn to match preferred direction.
           gosub Travel                  'Travel for count n.
           pref = 192                    'Turn west.
           gosub Turn
           gosub Travel
           pref = 128                    'Turn south.
           gosub Turn
           gosub Travel
           pref = 64                     'Turn east.
           gosub Turn
           gosub Travel
           end

'Turn: Turn slowly (to keep compass from swinging) to direction pref.

Turn:      do                            'Do this until we're pointed the right way.
             gosub GetDir                '  Where are we pointed now?
             DIR = DIR + 128             '  This will be pDIR, so make it the worst.
             do                          '  Do this until we get two readings alike.
               pDIR = DIR                '    Set prev DIR to DIR.
               pause 20                  '    Pause to give compass time to swing.
               gosub GetDir              '    Get new DIR.
             loop until (DIR = pDIR)     '    Do it again if compass moved.
             dif = DIR - pref            '  How much (and which way) are we off?
             if (dif < 128) then         '  Too far right?
               gosub HardLeft            '    Yes: Rotate left.
             else
               gosub HardRight           '    No:  Rotate right.
             endif
           loop until (abs(DIR - pref) < 2)'  Go back if we weren't close.
           return                                 'Pointed okay now, so return.

'Travel: Move in direction pref for count n.

Travel:    for i = 1 to n                'Do it n times.
             gosub GetDir                '  Where are we pointed?
             gosub Move                  '  Make the appropriate move.
           next                          '  Back for the rest.
           return                        'Over and out.

'Move: Take a step in the right (or left) direction.

Move:      dif = DIR - pref             'How much (and which way) are we off?
           if (dif < 3 or dif > 253) then Straight 'If not much then go straight.
           if (dif < 30) then GoLeft     'If less than 42 degrees right, then correct left.
           if (dif < 128) then HardLeft  'If 43-180 degrees right, then hard left.
           if (dif < 226) then HardRight 'If 43-180 degrees left, then hard right.
                                         '  else...
GoRight:                                 'Correct right: left forward, right stopped.
           lpulse = 1000
         rpulse = 750
           goto DoPulse

HardRight:                               'Hard right: left forward, right backward.
           lpulse = 1000
```

```
            rpulse = 1000
            goto DoPulse

HardLeft:                               'Hard left: left backward, right forward.
            lpulse = 500
            rpulse = 500
            goto DoPulse

GoLeft:                                 'Correct left: left stopped, right forward.
            lpulse = 750
            rpulse = 500
            goto DoPulse

Straight:                               'Straight: both forward.
            lpulse = 1000
            rpulse = 500

DoPulse:                                'Perform the programmed servo pulses...
            pulsout left, lpulse
            pulsout right, rpulse
            return                      '  ...and return

'----------------------YOUR CODE GOES ABOVE THIS LINE----------------------
```