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

General: info@parallax.com
Technical: support@parallax.com
Web Site: www.parallax.com
Educational: www.stampsinclass.com

Boe-dar: A Simulated Radar Display for Boe-Bots with Digital Encoders and RF Transmitters

By Philip C. Pilgrim

Introduction

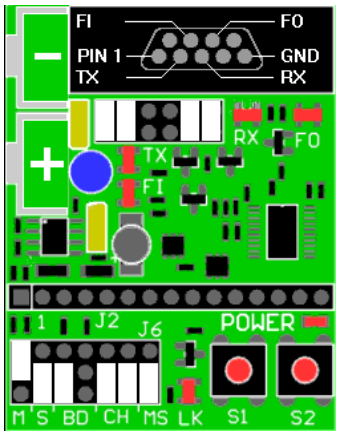
The Boe-dar software presented here is designed to be used with a PC and Parallax Boe-Bot equipped with the Boe-Bot Digital Encoder Kit (#28107), the SureLink RF Module set (2 each of #30065), and the QuickLink Demo Board (#30066). The encoders allow the Boe-Bot to keep track of its position. The RF module allows the Boe-Bot to transmit that position, along with other data, wirelessly to a PC host program, which displays the data in a simulated radar screen. Two pieces of software are included:

1. **Boe-dar.bs2**: The Boe-Bot-resident BASIC Stamp program the tracks the bot's position and sends the data via the RF link.
2. **Boe-dar.exe**: The PC-resident host program that receives data from the Boe-Bot and displays it on the screen.

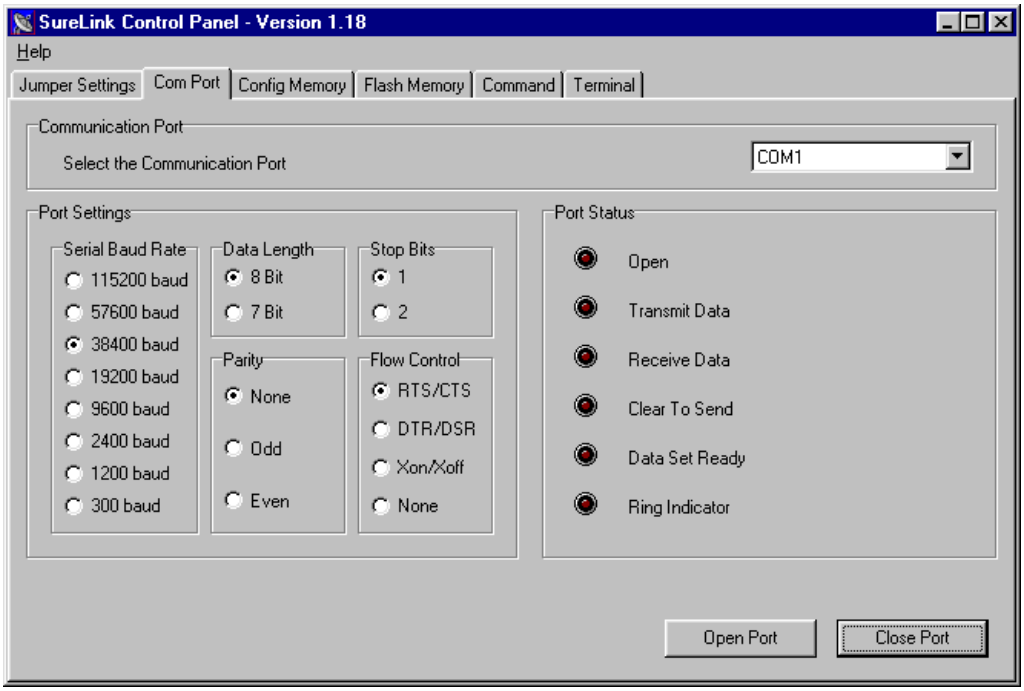
The encoders are fully explained in the document *Applying the Boe-Bot Digital Encoder Kit*, which can be downloaded from the Parallax website, as well as in the Digital Encoder Kit datasheet. The BASIC Stamp program included here includes software discussed in these documents, along with additional routines for transmitting the data.

Setting Up the Hardware

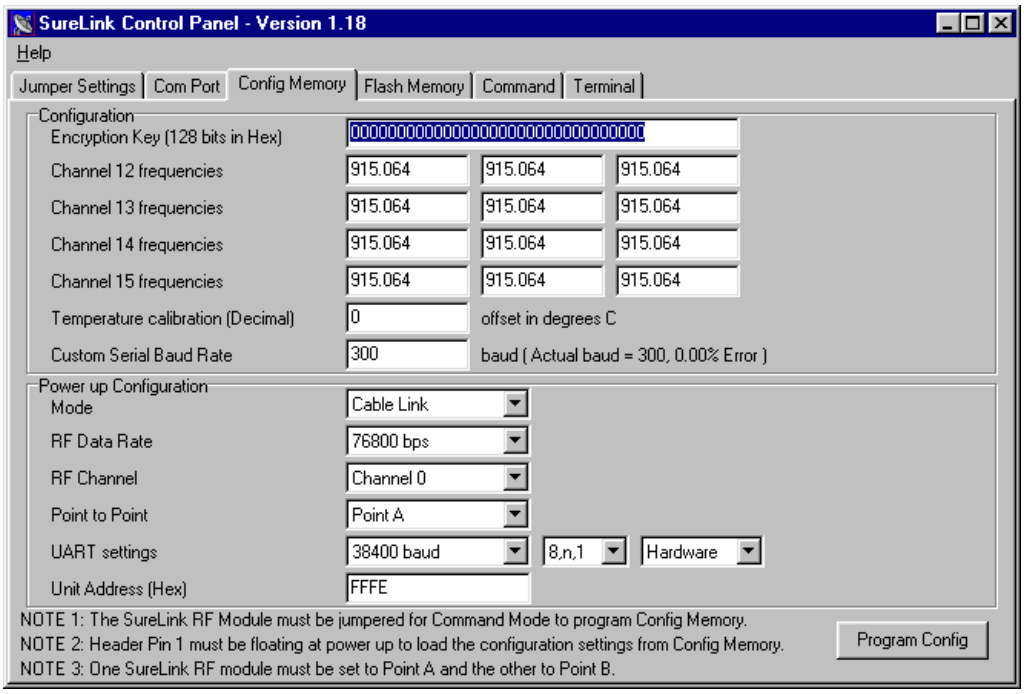
The SureLink transceivers will be operated in Cable Link mode. (See the SureLink datasheet.) To save space and wiring complexity on the BOE's prototype area, we will be doing a software setup and operating the SureLink modules with Pin 1 floating, rather than using jumper settings. So the first order of business is to do the software setup. Jumper your QuickLink Demo Board for Command Mode as shown below:



Plug a SureLink module into the QuickLink board, connect it to an available serial port on your PC, and apply power from a 9V battery. Start the SureLink Control Panel program, and click on the *Com Port* tab. Select the com port you're currently using, along with the other settings shown below. Then click "Open Port".



Next, click on the *Config Memory* tab. Select the options shown below and click "Program Config".

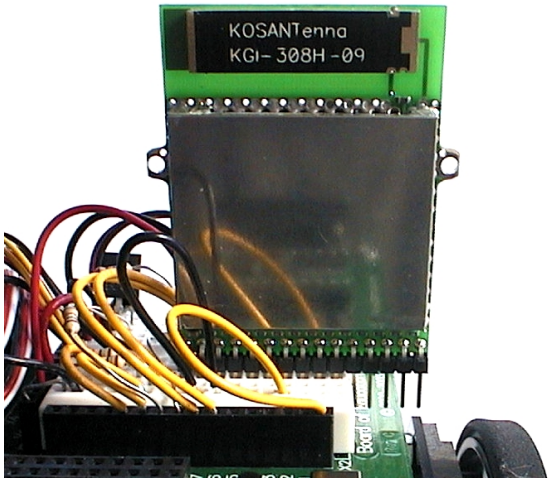


NOTE: If you are using Version 1.17 or earlier of the Control Panel program, select 57600 baud for the UART settings instead of 38400 baud. This is due to a bug in the earlier versions wherein the actual baudrate assigned is different from the one selected. While this will certainly work, it is nonetheless recommended that you download the latest version from **www.needhams.com**.

Go back to the *Com Port* tab and click "Close Port". Remove power from the Quick Link Demo board. Remove the QuickLink module and install the other one. Repeat the above procedure for the second module, *except choose **Point B** for the "Point to Point" setting*. Again close the serial port and remove power.

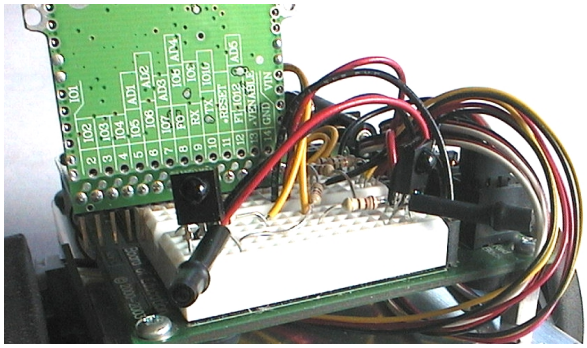
Now remove the "M" jumper from the demo board entirely, or place it so that it hangs on a single pin.

The first module you programmed may now be installed on the Boe-Bot. To make room for the other circuitry necessary you may wish to hang the module partly over the end of the prototype area as shown in the photo:

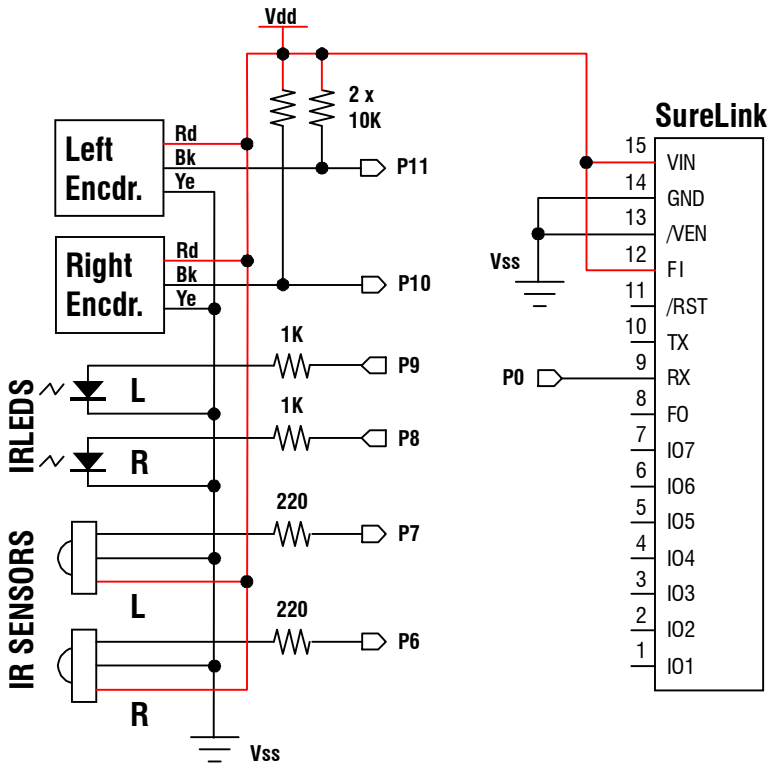


One pin will bend slightly, but not enough to cause any trouble.

This demo also uses the IR navigation circuitry described in Andy Lindsay's *Robotics with the Boe-Bot Version 2.0*, p. 220. The IRLEDs and sensors are installed pointing diagonally, as shown below:



A complete schematic, including the encoders, the SureLink module, and the IRLEDs and sensors is shown below:



BASIC Stamp Software

Before installing the BASIC Stamp software, make sure the encoders are well-calibrated, as described in the paper *Applying the Boe-bot Digital Encoders*. The program described here uses the constant **SDIRINC**, which needs to be obtained by and transferred from the calibration procedure. Also make sure to run on a flat, smooth surface. Poorly-calibrated encoders or rough terrain will cause rapid accumulation of position and direction errors.

The stamp-resident program, **Boe-dar.BS2**, uses this circuit to roam about, backing up and turning when its sensors encounter an obstacle. All the while, it monitors the encoders to keep track of position and orientation. It sends this data periodically to the host PC via the SureLink module, along with an additional character, the "event" byte. This byte may be one of the following:

! Indicates a restart. Tells the host PC program to reset all position and orientation information.
0 - 9 A digit (0 – 9) serves only as an identifier for the Boe-Bot sending the data.
R, **G**, **B**, **C**, **M**, **Y**, **W** Color indicators for red, green, blue, cyan, magenta, yellow, and white, respectively. These can be enabled by the host program to leave like-colored markers on the screen, indicating events detected by the Boe-Bot while it's roaming about.

The format used for each data transmission is:

`[HEX2 X, HEX2 Y, HEX2 Xd, HEX2 Yd, Event, CR]` , where

X is the most significant byte of the X position coordinate, **Y** is the most significant byte of the Y position coordinate, **Xd** is the most significant byte of the X orientation coefficient, **Yd** is the most significant byte of the Y orientation coefficient, and **Event** is the event byte described above. **CR** is the carriage return character, $0D.

In operation, this program has two modes, depending on how many times the reset button was pressed in rapid succession to start it. In each case, it reinitializes its position and transmits that position with an event code of "!". This is to make sure the host program resets to the "home" quadrant, in case it's keeping track of odometer "rollovers" from $FF to $00 or back. If the reset button is pressed once, the Boe-bot won't move anywhere, but will continuously send out its ID. If two presses, the bot will begin roaming, backing up and turning whenever it encounters an obstacle, all the while keeping track of its position and transmitting it periodically to the host PC. Each obstacle it detects registers as an event. A left-side obstacle is given the event code "R" (red for port side); a right-side obstacle, "G" (green for starboard).

Following is a program listing for this program, Boe-dar.BS2:

Sample Program (Boe-dar.BS2)

```
'{$STAMP BS2}
'{$PBASIC 2.5}

'-----------------------------------------------------------------------------
' Boe-dar.BS2: Program to roam about using IR sensors for obstacle avoidance.
' Wheel encoders keep track of position and orientation. These data are sent
' periodically to a host PC via an RF transmitter.
'-----------------------------------------------------------------------------
'
'Written by Philip C. Pilgrim      29 April 2004
'
'---------[Calibration Constants]-------------------------------------------

SDIRINC   CON 33101     'Sin(delta) factor by which to modify the angle vectors for each
                        'encoder pulse. Increase this number if Boe-Bot turns
                        'more than odometer says it does. Decrease, if less.
                        'Sin(delta) = SDIRINC / 262144.)

SPOSINC   CON 264       'Sin(delta) factor to compute distance for each encoder pulse.
                        '256 corresponds to 0.5" travel per two-wheel pulse.
                        'For centimeters, start with 650 to get 1.27cm of travel per
                        'two-wheel pulse. (SPOSINC = W * SDIRINC / 512, where W
                        'is the effective distance between wheels in the desired units.)

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

XINIT     CON 128       'Initial X position (byte) in whole units.
YINIT     CON 128       'Initial Y position (byte) in whole units.
                        'Initial direction is always assumed to be towards +Y (north).

POSINT    CON 16        'Position transmit interval.
IDINT     CON 128       'ID transmit interval.

ID        CON "1"       'Transmitted ID.

BAUD      CON $4006     'BS2 baudrate constant for 38400 baud, inverted polarity.
RSTBTN    CON 64        'EEPROM location used on startup to count reset button pushes.

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

RIGHT     CON 0         'Constants used as subscripts into bit arrays.
LEFT      CON 1
FWD       CON 0
BAK       CON 1

VMIN      CON 0         'Minimum adder/subtractor to null velocity.
VMAX      CON 50        'Maximum adder/subtractor to null velocity.
VINC      CON 2         'Velocity ramp rate.

RFXmt     PIN 0         'To RX pin on RF transmitter.
IRInpR    PIN 6         'Right IR sensor.
IRInpL    PIN 7         'Left IR sensor. (MUST be IRInpR + 1.)
IROutR    PIN 8         'Right IRLED.
IROutL    PIN 9         'Left IRLED. (MUST be IROutR + 1.)

SenseR    PIN 10        'Righthand encoder input.
SenseL    PIN 11        'Lefthand encoder input. (MUST be SenseR + 1.)

MotorR    PIN 12        'Righthand motor output.
MotorL    PIN 13        'Lefthand motor output. (MUST be MotorR + 1.)
```

```
IRInp     CON IRInpR    'Base address for IR sensors.
IROut     CON IROutR    'Base address for IRLEDs.
Sense     CON SenseR    'Base address for encoders.
Motor     CON MotorR    'Base address for motors.

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

Prev      VAR Bit(2)    'Previous readings from encoders.
Dir       VAR Bit(2)    'Wheel directions (FWD or BAK).
LastSide  VAR Bit       'Last side to get a pulse (LEFT or RIGHT).
NewSeq    VAR Bit       'Set if last movement started a new LR or RL sequence.
Side      VAR Bit       'Side index (LEFT or RIGHT).
Moved     VAR Bit       'Indicates motion since last checking.

NewDir    VAR Bit(2)
Event     VAR Byte

Xpos      VAR Word      'X location of the Boe-Bot's center.
X         VAR Xpos.HIGHBYTE 'Integer portion of Xpos. Low byte is fraction.
Ypos      VAR Word      'Y location of the Boe-Bot's center.
Y         VAR Ypos.HIGHBYTE 'Integer portion of Ypos. Low byte is fraction.
Xdir      VAR Word      'X component of the Boe-Bot's direction.
Xd        VAR Xdir.HIGHBYTE
Ydir      VAR Word      'Y component of the Boe-Bot's direction.
Yd        VAR Ydir.HIGHBYTE
PosCtr    VAR Byte      'Countdown for position interval.
IDCtr     VAR Byte      'Countdown for ID interval.

Veloc     VAR Byte
i         VAR Byte      'Scratch variables...
n         VAR Byte

          DATA          @RSTBTN, 0

'---------[Program begins here.]-------------------------------------------

READ RSTBTN, i            'Determine how may times reset button was pressed.
WRITE RSTBTN, i + 1
PAUSE 1000
WRITE RSTBTN, 0
SELECT i

  CASE 0:                 'One press: Just send out ID string.
    GOSUB Initialize
    Event = ID
    DO
      GOSUB Transmit
      PAUSE 500           'Do it every half second.
    LOOP

  CASE 1:                 'Two presses: Start roaming...
    GOTO MainProg

  CASE ELSE:              'Otherwise, just die.
    END

ENDSELECT

'---------[Main routine begin here.]----------------------------------------

MainProg:                                           'Begin roaming...

  GOSUB Initialize                                  'Do encoder initialization.

  DO
    FOR Side = RIGHT TO LEFT                        'Read IR obstacle sensors.
      FREQOUT IROut + (Side ^ 1), 1, 38500          'When obstacle detected,
      NewDir(Side) = ~ INS.LOWBIT(IRInp + (Side ^ 1)) 'opposite wheel goes backward.
    NEXT
    NewDir(LEFT) = NewDir(LEFT) & (NewDir(RIGHT) ^ 1) 'Still need to turn if both sides detect.
    IF (NewDir(LEFT) | NewDir(RIGHT)) THEN          'If obstacle...
      GOSUB WaitEnc                                 'Wait for motion to cease.
      IF NewDir(LEFT) THEN Event = "G" ELSE Event = "R" 'Event is red for port obstacle, green for stbd.
      GOSUB Transmit                                'Transmit event.
      Veloc = VMIN                                  'Throttle down.
      Dir(RIGHT) = BAK
      Dir(LEFT) = BAK
      n = 25
      GOSUB Move                                    'Back up 25 motor pulses..
      GOSUB WaitEnc                                 'Wait for motion to cease.
      Veloc = VMIN
```

```
      Dir(RIGHT) = NewDir(RIGHT)                      'Now turn in place.
      Dir(LEFT) = NewDir(LEFT)
      n = 5 * Dir(RIGHT) + 25                         'Different amounts for L & R, so don't get stuck in
      GOSUB Move                                      '  corners.
      GOSUB WaitEnc
      Veloc = VMIN
    ELSE                                              'If no obstacles...
      Dir(LEFT) = FWD
      Dir(RIGHT) = FWD
      n = 1
      GOSUB Move                                      'Move forward one motor pulse.
    ENDIF
  LOOP                                                'Ad infinitum...

  END

Move:

  FOR i = 1 TO n                                      'n motor pulses.
    PULSOUT MotorR, 750 + ((Dir(RIGHT) << 1 - 1) * Veloc)
    PULSOUT MotorL, 750 - ((Dir(LEFT) << 1 - 1) * Veloc)
    GOSUB ChkEnc                                      'Always check the encoders.
    PAUSE 10
    Veloc = Veloc + VINC MAX VMAX                     'Ramp up the velocity.
  NEXT
  RETURN

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

Initialize:                         'Do encoder initialization.

  Xdir = 0                          'Set direction to +Y.
  Ydir = $4000
  Xpos = XINIT << 8                 'Set positions from constants.
  Ypos = YINIT << 8
  FOR Side = RIGHT TO LEFT          'Read initial encoder values.
    Prev(Side) = INS.LOWBIT(Sense + Side)
  NEXT
  Event = "!"
  GOTO Transmit

'-------------------------------------------------------------------------------

WaitEnc:                            'Track encoders until motion ceases.

  FOR i = 1 TO 30                   'Need 30 motionless intervals to be sure.
    GOSUB ChkEnc                    'Check the encoder.
    IF (Moved) THEN i = 0           'If the bot moved, reset counter and start over.
  NEXT
  RETURN                            'No motion for 30 steps. It's stopped.

'-------------------------------------------------------------------------------

ChkEnc:                             'Update and record position from encoders.
                                    'Just call it often enough to catch all the
                                    'changes.
  Moved = 0                         'Initialize to no detected movement.
  FOR Side = RIGHT TO LEFT                'For both encoders...
    IF (INS.LOWBIT(Sense + Side) ^ Prev(Side)) THEN  'Encoder different from prior value?

        Prev(Side) = ~ Prev(Side)         '  Yes: Update with new value.
        Moved = 1                         '       Indicate that we've moved.
        NewSeq = ~ (Side ^ Dir(Side) ^ LastSide & NewSeq)   'Start a new LR or RL sequence unless
                                          '       different side moved last and that was
                                          '       the start of a new sequence.
        LastSide = Side ^ Dir(Side)               '       Update last side to move.
        IF (NewSeq) THEN DoPos            '       New sequence starts with position.
                                          '       2nd half of same sequence undoes angle first.

      DoAng:                              '       Angle change.

        Ydir = Ydir - ((Dir(Side) ^ Side ^ Xdir.BIT15 << 1 - 1) * (ABS(Xdir) ** SDIRINC >> 3))
        Xdir = Xdir + ((Dir(Side) ^ Side ^ Ydir.BIT15 << 1 - 1) * (ABS(Ydir) ** SDIRINC >> 2))
        Ydir = Ydir - ((Dir(Side) ^ Side ^ Xdir.BIT15 << 1 - 1) * (ABS(Xdir) ** SDIRINC >> 3))
        IF (NewSeq) THEN DoNext

      DoPos:                              '       Position change.

        Xpos = Xpos - ((Dir(Side) ^ Xdir.BIT15 << 1 - 1) * (ABS(Xdir) ** SPOSINC))
        Ypos = Ypos - ((Dir(Side) ^ Ydir.BIT15 << 1 - 1) * (ABS(Ydir) ** SPOSINC))
        IF (NewSeq) THEN DoAng
```

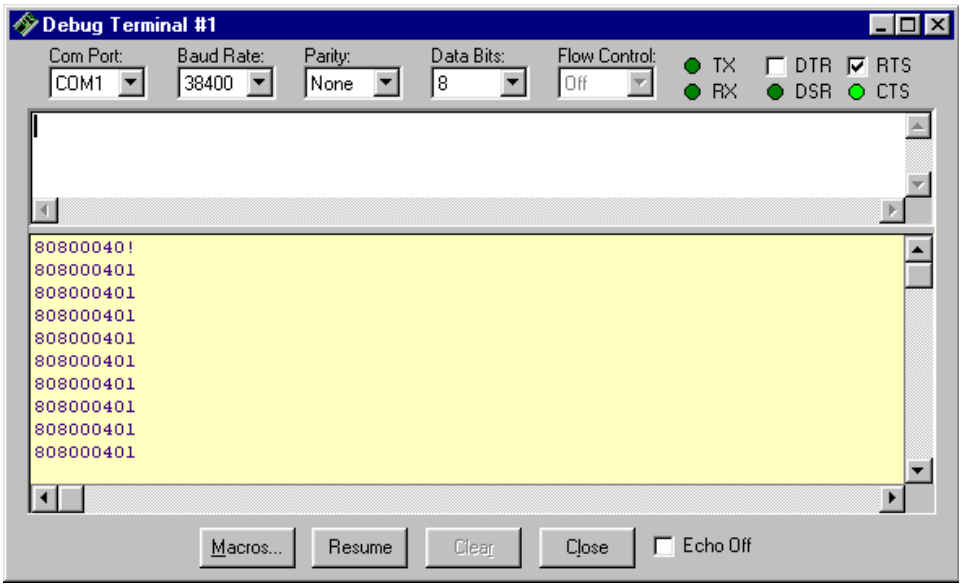
```
     DoNext:

    ENDIF
  NEXT
  IF (IDCtr) THEN IDCtr = IDCtr - 1      'ID counter decrements whether moved or not.
  IF (Moved AND PosCtr) THEN PosCtr = PosCtr - 1  'Position counter decrements only if moved.
  IF (PosCtr AND IDCtr) THEN RETURN      'Return if neither has reached zero.
  Event = ID                             'Default event is ID. Fall thru to Transmit.

'-----------------------------------------------------------------------------

Transmit:
  PosCtr = POSINT                        'Reset both counters.
  IDCtr = IDInt
  SEROUT RFXmt, BAUD, [HEX2 X, HEX2 Y, HEX2 Xd, HEX2 Yd, Event, CR, LF]     'Transmit data.
  Event = ID                             'Reset event to default.
  RETURN                                 'Over and out.
```

Load this program into the Boe-bot and press the reset button once. Now power up the Quicklink module. Its yellow “Link” LED should come on, indicating that the Boe-bot and PC are communicating. Open a debug screen from the BASIC Stamp editor, and set it up for the com port the Quicklink module uses and 38400 baud. Make sure to check the RTS box. You should see a display like the one shown below:



If so, you are ready to continue. If not, recheck all your connections. You may need to double check that the SureLink modules are programmed correctly.

Now, place the Boe-bot on the floor and press the reset button twice in rapid succession. It should begin to roam about, avoiding obstacles in its path. You will now see the numbers in the debug screen changing. When the Boe-bot encounters an obstacle, the last character on the line will be either an “R” (port side obstacle) or a “G” (starboard side obstacle). Otherwise, the bot’s ID, “1”, will be displayed.

Now close the debug screen. Setup is complete.

PC Host Software

While the Boe-bot is roaming about, the PC host program, **Boe-dar.exe**, can monitor its every move and track it on a simulated radar screen. To use this program, the Boe-bot should be powered on, and

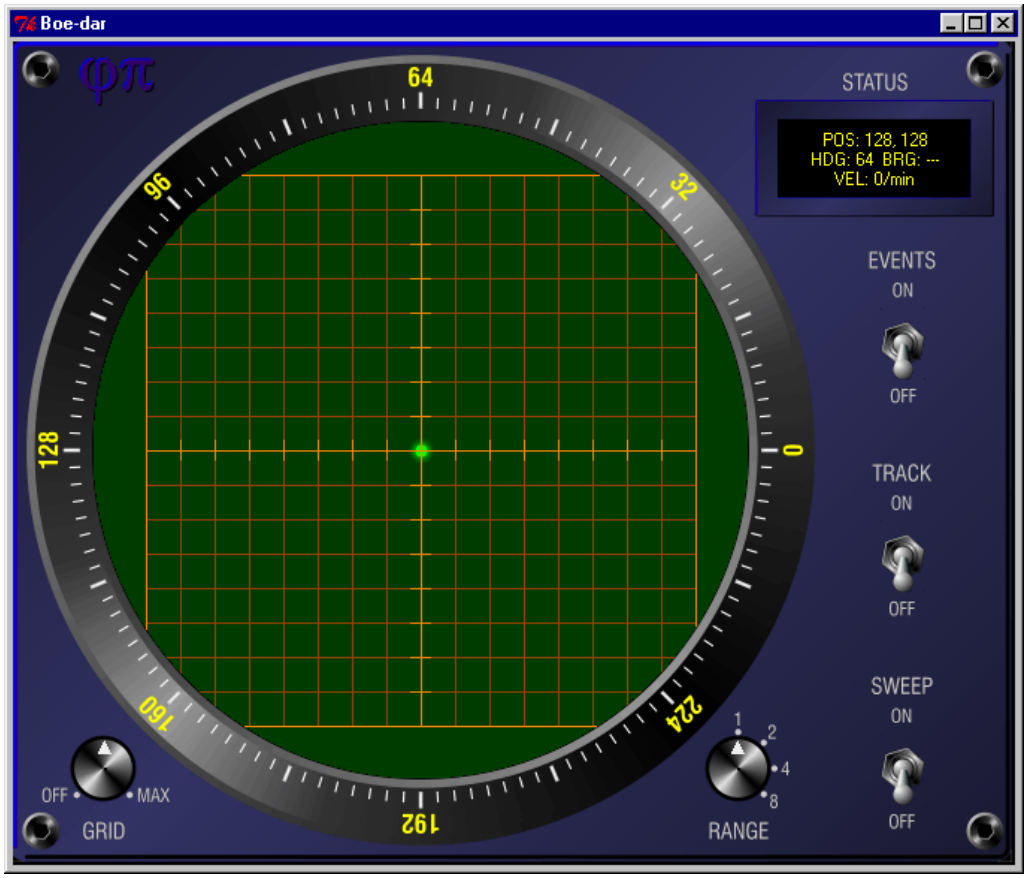
the QuickLink module should be powered up and connected to a PC serial port. When the program starts, it will scan its serial ports for a signal from the Boe-bot. Once it finds one, it will create a file named **Boe-dar.ini** in the same directory where **Boe-dar.exe** resides. The file will look like this:

```
# This file can be edited to enable any serial device/baud rate combination.
# Each line represents one device of the form: devicename: baudratelist

 COM1: 38400
 COM2: 38400
 COM3: 38400
 COM4: 38400
 COM5: 38400
 COM6: 38400
 COM7: 38400
 COM8: 38400
```

You can edit this file yourself to eliminate com ports you don't plan to use. This can speed up the startup process in future runs. You can even add or change baud rates, if you like, to use with other serial inputs than the one discussed here. To add 9600 baud to COM1, for example, that line would read: "**COM1: 9600, 38400**", quotes excluded.

Once the program links to a Boe-bot, its display should look like the following:



The green dot a the center of the screen is the Boe-bot's current position. The center of the screen is *always* location (128, 128). The highlighted square (here shown at the farthest outside extreme of the display grid) represents a boundary encompassing locations (0, 0) to (255, 255). You can zoom out from here by changing the RANGE switch setting. To do so, click on the range switch and drag the

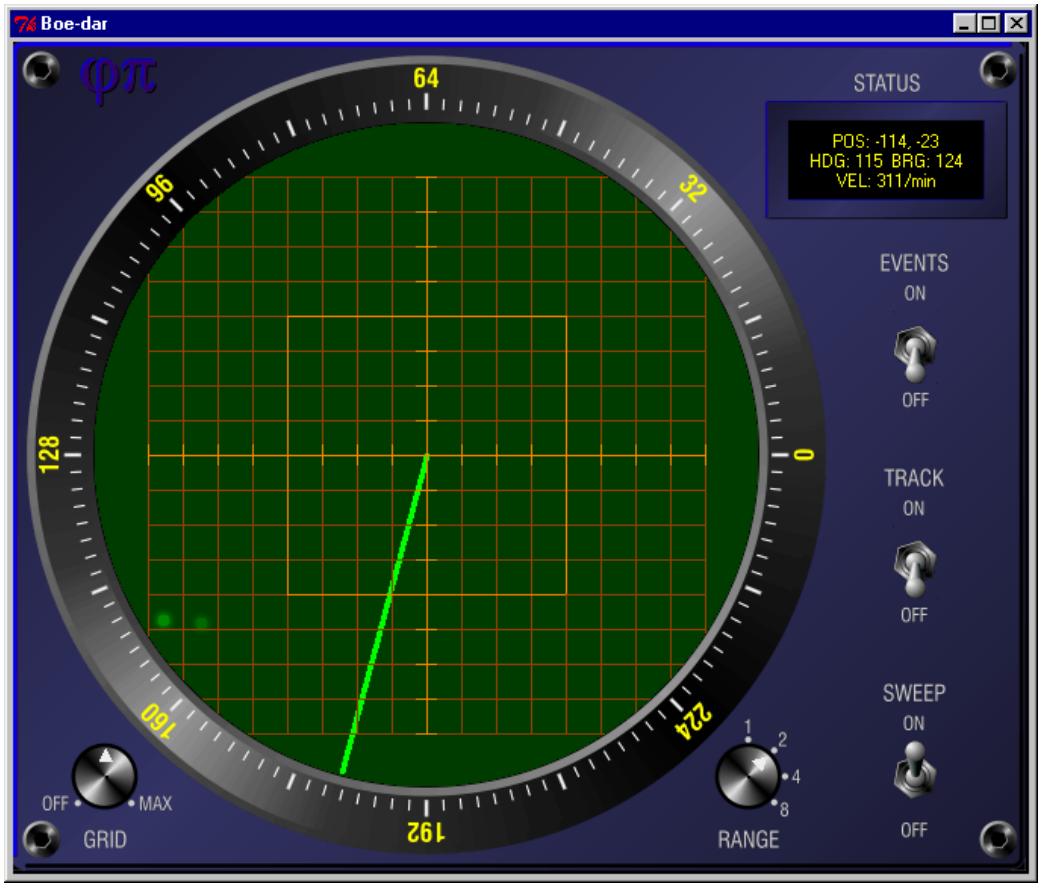
mouse north or east to increase the range; south or west to decrease it. As you increase the range, the highlighted square will shrink. At the maximum range of 8, the screen will encompass and area 2048 units on a side. You can brighten or darken the grid (reticle) by adjusting the GRID control. It works the same way as the RANGE switch.

The STATUS display shows the Boe-bot's current position, heading (the direction the Boe-bot is *pointed*), bearing (the direction the Boe-bot is actually *moving*), and velocity in units per minute. The heading, bearing, and azimuth ring around the scope are all calibrated in brads (binary radians).

You are now ready to roam. Press the Boe-bot's reset button twice to make it start moving. When you do so, you should see the green spot begin to move on the screen, tracking the bot's every movement. To lay down an actual track on the screen, click on the TRACK switch to raise it. You will then see yellow lines connecting each reported position. To remove any tracks from the screen, just flip the switch down.

Now click the EVENTS switch to raise it. With this switch on, whenever the Boe-bot encounters an obstacle, a colored square corresponding to the event will be deposited at that location. By leaving this switch on, you will eventually begin to see a "map" of the Boe-bot's environs. To remove any events from the screen, flip the EVENT switch down. A screen including tracks and events is shown below:

The SWEEP switch is used to turn on a more realistic, radar-like mode of operation. A green radial line will rotate about the screen. The Boe-bot's position will update *only* when the line encounters its "echo". During sweep mode, track lines can still be laid down on the screen, but event recording is disabled. (It would miss most events anyway.) A typical view in sweep mode is shown below:



Notice that there are two echoes, a bright one and a dimmer one. The bright one is the most recent and indicates the bot's latest known position. The dimmer one represents the prior position.

If the program should lose track of the Boe-bot (*e.g.* if it goes out of range of the Surelink modules), the STATUS box will display the message, "Lost signal." Once the signal is restored, the program will continue tracking.

NOTE: If, while the signal was lost, the Boe-bot were to cross from one unit sector (*i.e.* $xx00, $yy00 – $xxFF - $yyFF) to another, the program would not be able to track it accurately, since it needs to catch that event when it happens. Therefore, on signal recovery, the program will assume that locations begin from that point on in the home sector, $0000, $0000 – $00FF, $00FF.

To exit the program, just click the "X" box in the upper right-hand corner.