Boe-Bot Programming

Mostly copy/paste from BASIC_Stamp_Manual 3-Robotics-With-The-Boe-Bot-v3.0


Page 89: **FREQOUT** You will see this command at the start of many of the book’s examples.

The next example program tests the piezospeaker. It uses the FREQOUT command to send precisely timed high/low signals to a speaker. Here is the FREQOUT command’s syntax:

```
FREQOUT Pin, Duration, Freq1 {,Freq2}
```

Here’s an example of a FREQOUT command that’s used in the next example program.

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

Page 92: Just FYI
The Debug Terminal also has a Transmit windowpane, which allows you to send information to your BASIC Stamp while a program is running.

**RPM of the Boe-Bot wheels**

Here is a curve out of the manual on page 96. Figure 3-20 shows an example of a transfer curve for a continuous rotation servo. The horizontal axis shows the pulse width in ms, and the vertical axis shows the rotational velocity in RPM.
How do we connect this curve with what we know? The abscissa (pulse width) axis tells us the full width of the pulse. To relate these numbers to PULSOUT values, divide by 2. Thus the abscissa axis goes from 650 to 850.

I ran the Boe-Bot holding it in the air. Pin12 was 650 with a 2s pause. The average for each wheel for five rotations was 6.29 s. Now subtract the 2s delay and divide by 5. RPS=5/4.29=1.17 rev/s. Multiply by 60 s/min to get RPM. So the RPM data was 69.6 RPM.

Chapter 4 Activities

1) Program the Boe-Bot to perform the basic maneuvers: forward, backward, rotate left, rotate right, and pivoting turns.
2) Tune the maneuvers from Activity #1 so that they are more precise.
3) Use math to calculate the number of pulses to deliver to make the Boe-Bot travel a predetermined distance.
4) Instead of programming the Boe-Bot to make abrupt starts and stops, write programs that make the Boe-Bot gradually accelerate into and decelerate out of maneuvers.
5) Write subroutines to perform the basic maneuvers so that each subroutine can be used over and over again in a program.
6) Record complex maneuvers in the BASIC Stamp module's unused program memory and write programs that play back these maneuvers.

ACTIVITY #1: BASIC BOE-BOT MANEUVERS

Program the Boe-Bot to perform the basic maneuvers: forward, backward, rotate left, rotate right, and pivoting turns.

Do problem 1.
**ACTIVITY #2: TUNING THE BASIC MANEUVERS**

850 To test the Boe-Bot for direction and turning accuracy, first we run the Boe-Bot at full speed for about 10 seconds. Then fuss with the numbers to make it go in a straight line.

Then we make the Boe-Bot turn 90°.

<table>
<thead>
<tr>
<th>Page in Manual</th>
<th>Counter Max</th>
<th>Pin13 left wheel PULSOUT</th>
<th>Pin12 right wheel PULSOUT</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110 has</td>
<td>407</td>
<td>850</td>
<td>650</td>
<td>Full speed for about 10 seconds. Did it go in a straight line? Increase the 650. Why not the 850? Iterate until the BB goes in a straight line.</td>
</tr>
<tr>
<td>Rotation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>24</td>
<td>650</td>
<td>650</td>
<td>If not 90° change max counter to 23 or 25. If overshoots, reduce the speed a little by increasing PULSOUT values. Iterate until the BB turns 90°.</td>
</tr>
</tbody>
</table>

**ACTIVITY #3: CALCULATING DISTANCES**

The easiest way to do this is to set the Boe-Bot next to a ruler and make it travel forward for one second. The max counter for one second of operation is found using the document I emailed you, case 4.

One loop = 1.3 + 1.7 + 20 ms + 1.6 = 24.6 ms = 0.0246 s

Number of pulses = 1 s / 0.0246 s = 40.65 pulses in one second → max counter = 41

Do problem 2

Once you know the cm/s value, you can calculate the time it takes to go a particular distance.

**My data:**
I ran more tests taking data for five revolutions on a table. The average time was 6.193-2=4.19s for five revolutions. So revolutions per second is 6/4.19=1.43 rev/s. So the RPM data rolling on the table was 85.9 RPM.

The average distance was 107.7cm. Hence the average speed was 107.7cm/4.19s=25.7cm/s. Knowing the average speed you can predict the distance for a given time or the time to go a given distance.
I used max counter = 160 with Pin12=650, Pin13=850. The Boe-Bot went five revolutions plus about 20 degrees.

**ACTIVITY #4: MANEUVERS—RAMPING**

Ramping is a way to gradually increase or decrease the speed of the servos instead of abruptly starting or stopping. This technique can increase the life expectancy of both your Boe-Bot’s batteries and your servos.

To ramp, start with no motion then keep increasing one pin’s PULSOUT until you get to PULSOUT of 850. At the same time keep decreasing the other pin’s PULSOUT until you get to PULSOUT of 650.

Do problems 3 and 4

Following is a complete activity programmed.
Robotics with the Boe-Bot - StartAndStopWithRamping.bs2

'Ramp up, go forward, ramp down.
' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG "Program Running!"
pulseCount VAR Word

' FOR...NEXT loop counter.

'----[ Initialization ]-------------------------------------------------------------
FREQOUT 4, 2000, 3000 ' Signal program start/reset.
'----[ Main Routine ]-------------------------------------------------------------

'Ramp up forward.
FOR pulseCount = 1 TO 100 ' Loop ramps up for 100 pulses.
  PULSOUT 13, 750 + pulseCount ' Pulse = 1.5 ms + pulseCount.
  PULSOUT 12, 750 - pulseCount ' Pulse = 1.5 ms - pulseCount.
  PAUSE 20 ' Pause for 20 ms.
NEXT

' Continue forward for 75 pulses.
FOR pulseCount = 1 TO 75 ' Loop sends 75 forward pulses.
  PULSOUT 13, 850 ' 1.7 ms pulse to left servo.
  PULSOUT 12, 650 ' 1.3 ms pulse to right servo.
  PAUSE 20 ' Pause for 20 ms.
NEXT

'Ramp down from going forward to a full stop.
FOR pulseCount = 100 TO 1 ' Loop ramps down for 100 pulses.
  PULSOUT 13, 750 + pulseCount ' Pulse = 1.5 ms + pulseCount.
  PULSOUT 12, 750 - pulseCount ' Pulse = 1.5 ms - pulseCount.
  PAUSE 20 ' Pause for 20 ms.
NEXT

END ' Stop until reset.

Turning the Bo-Bot. The two PULSOUTs are the same value to turn about the centerline.

Do Problem 5 and 6.
ACTIVITY #5: SIMPLIFY NAVIGATION WITH SUBROUTINES

What is a subroutine? It is a separate piece of code you can call over and over again. Using subroutines (properly named) makes the program shorter and easier to follow. In order to separate the commands, we put in a carriage return. What happens is the same thing as hitting the Return button on a computer.

The following program sends messages back to the STAMP terminal,

```
' Robotics with the Boe-Bot - OneSubroutine.bs2
' This program demonstrates a simple subroutine call.
' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG "Before subroutine", CR       ' CR means carriage return, i.e. start a new line
PAUSE 1000

GOSUB My_Subroutine
DEBUG "After subroutine", CR
END

My_Subroutine:
DEBUG "Command in subroutine", CR
PAUSE 1000
RETURN
```

Do problem 7 Page 123
How can we make the program shorter and more flexible? We use variables. (page 124). Let’s analyze the following program.

' Robotics with the Boe-Bot - MovementWithVariablesAndOneSubroutine.bs2
' Make a navigation routine that accepts parameters.
' {STAMP BS2}
' {PBASIC 2.5}

DEBUG "Program Running!"

counter VAR Word
pulseLeft VAR Word
pulseRight VAR Word
pulseCount VAR Byte

FREQOUT 4, 2000, 3000 ' Signal program start/reset.

' Forward
pulseLeft = 850: pulseRight = 650: pulseCount = 64: GOSUB Navigate

' Left turn
pulseLeft = 650: pulseRight = 650: pulseCount = 24: GOSUB Navigate

' Right turn
pulseLeft = 850: pulseRight = 850: pulseCount = 24: GOSUB Navigate

' Backward
pulseLeft = 650: pulseRight = 850: pulseCount = 64: GOSUB Navigate

END

Navigate:
FOR counter = 1 TO pulseCount
    PULSOUT 13, pulseLeft
    PULSOUT 12, pulseRight
    PAUSE 20
NEXT

PAUSE 200
RETURN