Week 4

Experiment # 12

Activity #3

Aim: Navigating by the Numbers in Real-Time

In Program Listing 4.2, the Boe-Bot checked between each **forward** pulse to see if it was still okay to move forward. When the Boe-Bot performed maneuvers, they were essentially pre-recorded motions. Another approach to IR navigation is to check the sensors, apply a single pulse based on the sensor input, then check the sensors again. The Boe-Bot behaves very differently using this technique.

Real-Time IR Navigation

Program Listing 5.3 checks the IR pairs and delivers one of four different pulses based on the sensors. Each of the navigational routines is just a single pulse in either the **forward**, **left_turn**, **right_turn** or **backward** directions. After the pulse is applied, the sensors are checked again, then another pulse is applied, etc. This program also makes use of some programming techniques you will find very useful.

```
' Robotics! v1.5, Program Listing 4.3: IR Roaming by Numbers in Real Time

' {$Stamp bs2}                    ' Stamp Directive.

'----- Declarations ---------------
                                  ' The lower 2 bits of the
sensors var nib                   ' sensors variable are used to store
                                  ' IR detector values.
'----- Initialization -------------

output 2                          ' Set all I/O lines sending freqout
output 7                          ' signals to function as outputs
output 1
freqout 2, 2000, 3000             ' Program start/restart signal.
low 12                            ' Set P12 and 13 to output-low.
low 13
'----- Main Routine ---------------

main:
                                  ' Detect object on the left.
freqout 7,1,38500                 ' Send freqout signal - left IRLED.
sensors.bit0 = in8                ' Store IR detector output in RAM.
                                  ' Detect object on the right.
freqout 1,1,38500                 ' Repeat for the right IR pair.
```

```
sensors.bit1 = in0
pause 18                        ' 18 ms pause(2 ms lost on freqout).

' By loading the IR detector output values into the lower 2 bits of the sensors
' variable, a number btwn 0 and 3 that the branch command can use is generated.
branch sensors,[backward,left_turn,right_turn,forward]

'----- Navigation Routines -------

forward: pulsout 13,1000: pulsout 12,500: goto main
left_turn: pulsout 13,500: pulsout 12,500: goto main
right_turn: pulsout 13,1000: pulsout 12,1000: goto main
backward: pulsout 13,500: pulsout 12,1000: goto main
```

How IR Roaming by Numbers in Real-Time Works

Look up the **branch** command in Appendix C: PBASIC Quick Reference or in the BASIC Stamp Manual. This Program listing declares the **sensors** variable, which is one nibble of RAM. Of the four bits in the sensors variable, only the lowest two bits are used. Bit-0 is used to store the left detector's output, and bit-1 is used to store the right detector's output.

```
declarations:
sensors var nib
```

I/O pins P7, P1, and P2 are declared outputs. P2 is declared an output so that freqout can send signals to the speaker. P7 and P1 are declared outputs because these lines drive the left and right IR LED circuits.

```
initialization:
output 7
output 1
output 2
freqout 2,2000,3000
```

The main routine starts with the **freqout** commands used to send the IR signals, but the commands following each freqout command are slightly different from those used in the previous program. Instead of saving the bit value at the input pin to a bit variable, each bit value is stored as a bit in the **sensors** variable. Bit-0 of **sensors** is set to the binary value of **in8**, and bit-1 of the sensors variable is set to the binary value of **in0**. After setting the values of the lower two bits of the sensors variable, it will have a decimal

value between “0” and “3.” The **branch** command uses these numbers to determine to which label it sends the program.

```
main:
freqout 7,1,38500
sensors.bit0 = in8
freqout 1,1,38500
sensors.bit1 = in0
pause 18
branch sensors,[backward,left_turn,right_turn,forward]
```

The four possible binary numbers that result are shown in Table 5.1. Also shown is the branch action that occurs based on the value of the state argument.

Table 5.1: IR Detector States as Binary Numbers

Binary Value of state	Decimal Value of State	What the Value Indicates, Branch Action Based on State
0000	0	`in8 = 0` and `in0 = 0`, Both IR detectors detect object, pulse servos `backward`.
0001	1	`in8 = 0` and `in0 = 1`, Left IR detector detects object, pulse `right_turn`
0010	2	`in8 = 1` and `in0 = 0`, Right IR detector detects object, pulse for `left_turn`
0011	3	`in8 = 1` and `in0 = 1`, Neither IR detector detects object, pulse `forward`.

Depending on the value of the **sensors** variable, the **branch** command sends the program to one of four routines: **forward**, **left_turn**, **right_turn**, **or backward**. Whichever routine the program ends up in gives the servos a single pulse in the appropriate direction, after which, the routine sends the program back to the **main** routine for another check of the sensors.

```
routines:

forward:        pulsout 13,1000: pulsout 12,500: goto main
left_turn:      pulsout 13,500: pulsout 12,500: goto main
right_turn:     pulsout 13,1000: pulsout 12,1000: goto main
backward:       pulsout 13,500: pulsout 12,1000: goto main
```

Task

You can rearrange the address labels in the branch command so that the Boe-Bot does different things in response to obstacles. One interesting activity is to try replacing the **backward** address with the **forward** address. There will be two instances of **forward** in

the **branch** address list, but this is not a problem. Also, swap the **left_turn** and **right_turn** addresses.

1. Try making the changes just discussed.
2. Run the modified version of Program Listing 5.3, and have the Boe-Bot follow your hand as you lead it places.

If you stop your hand, the Boe-Bot will run into it. Because of this, one Boe-Bot cannot be programmed to follow another without some way of distance detection. If the one in front stops, the one in back will crash into it. This problem will be fixed as an example in the next chapter.