Lab 03: BOE-Bots in Motion

When You Need to Get Where You Used to Be

INTRODUCTION

Flashing the LED is interesting, now that we know the how and the why. But it doesn't make the bot go, does it? Before we get too far ahead of ourselves, we need to pause for a moment and think a little about programming in general.

As exciting as it might be to just sit down and start frantically typing line after line of code (…seriously, even I can't believe I just wrote that…), we really need to develop an understanding of how to approach the problem-solving process.

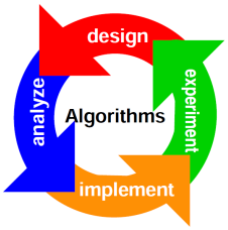
OBJECTIVES

The specific goals for this exercise are:

- Developing an algorithm for completing a specific task
- Learning simple P-Basic syntax and commands
- Becoming accustomed to the development environment
- Translating an algorithm into an executable program

CONSTRUCTING ALGORITHMS

An algorithm is your plan of attack for solving a problem or completing a task. There are multiple ways to construct an algorithm, but not a single one of them involves writing lines of syntactically correct code (in whatever language) on a piece of paper then re-typing it on the computer.



Whether you prefer outlining, or constructing a classical flowchart, the whole point is to create the map of the solution. What steps need to be taken? In what specific order? Are there multiple ways to achieve your goal? Your algorithm should help you figure out the most efficient way to solve your problem.

An algorithm should never be written in any specific coding language; it should be in plain English. You should be able to translate an algorithm into any programming language you choose, and anyone should be able to read your algorithm and understand it, even if they are not programmers.

CONTROL THE SERVOS, CONTROL THE WHEELS

Just like we sent a signal to the pin to light the LED, we can send a signal (actually, complete the circuit and send a pulse of energy from the battery) to the servo motors that drive the wheels.

Notice that your servos are connected to the circuit board at sockets 12 and 13. Which is left and which is right?

```
'Program: NBOTB
'{$STAMP BS2}           'Stamp directive
'{$PBASIC 2.5}          'Language directive
DEBUG "Forward pulse to left wheel"
PULSOUT 12, 650         'Left wheel forward
PAUSE 20                'Take a 20ms break
```

Did the left wheel roll forward (just a little bit)? If not, notice which wheel did move, and whether it ticked forwards or backwards. Go back and change the comments (not the actual program!) to reflect what really happened.

Once you figure out which wheel is connected to which socket, add code to move the other wheel:

```
DEBUG "Send forward pulse to right wheel"
PULSOUT 13, 650         'Right wheel forward
PAUSE 20.               'Take a 20ms break
```

That didn't spin the right way, did it? To go forward, the right wheel has to spin clockwise. But the left wheel has to spin counterclockwise! (Check it.)

This is easy enough to adjust in the program. The PULSOUT command needs two inputs: PIN, SPIN. You already know which pin (12 or 13) controls which wheel. The SPIN argument can rotate the servo in either direction, depending on the value you use.

Change the 650 (clockwise) values to 850 (counter-clockwise) values. What happens? Note which value ticks each wheel forward and which one ticks backwards.

Now that you know the combinations, we can roll forwards, backwards, and even turn corners! But one pulse to the servo doesn't really move you very far, does it?

CODE ONCE, RE-USE FOREVER

Before we go any further, let's re-write what we just wrote, but make it easier for us to modify in the future. Like we did last week, we're going to use PIN declarations for our wheel servos, and constants for the rotation direction and speed:

```
leftWheel   PIN 12  ' PIN 12 =  left wheel servo
rightWheel  PIN 13  ' PIN 13 = right wheel servo
CW  CON 650         ' Clockwise spin = 650
CCW CON 850         ' Counterclockwise spin = 850
```

Now re-write the PULSOUTs using the appropriate labels:

```
PULSOUT 12, 650         'Left wheel forward
```

becomes

```
PULSOUT leftWheel, CW       'Left wheel forward
```

It's a small thing, and might feel unnecessary—but when we have to switch the bots back and forth from front-wheel to rear-wheel drive, it's easier to change two numbers at the top of the program than hunt down every PULSOUT and edit it!

MAP THE SOLUTION

Let's practice constructing an algorithm while we think about continuous motion. How would we solve the problem of having the bot navigate a closed course? Think about it: you telling a person to "take a walk around the block" is probably a sufficient statement for them to understand what it is you expect them to do. But it's clearly nowhere near sufficient for the bot.

The program must be completely, perfectly literal. You would not need to tell a person: "Take 200 steps forward. Stop. Turn 90° left. Stop. Take 200 steps forward. Stop." And so on, until you had navigated them back to their starting position. But the bot needs every last detail made explicit.

- Write the algorithm: For the bot to traverse a completely closed path, ending back at the same location where they started. Remember that you really, really need to be explicit about every single action the bot must take.
- Measure the path: How far do you have to go on each leg of the path? Note that it doesn't matter if you measure the path in inches or cm.
- Correlate the PULSOUT tick: How far does the bot travel forward with one PULSOUT command? Hmmm….

PROGRAM YOUR ALGORITHM

- Roll forward: Use the same FOR-NEXT loop structure you wrote for your previous program to make your bot roll forward (don't forget to declare your variables!). This is also a great way to calibrate your PULSOUT! You may not be able to accurately measure how far one pulse takes the bot, but ten is a nice, round number. How far do ten PULSOUTs take the bot?
- Roll backward: Create a second FOR-NEXT loop to roll the bot backwards by the same distance you rolled forwards. Never forget that copy/paste is your best friend.
- Turn right: Copy/paste one more loop, and modify the PULSOUT commands so that your bot turns right. What count will give you a nice 90° turn?
- Turn left: Fourth loop, now turn to the left. Adjust to make it a 90° turn—it might not be the same as the right turn!

So, these four loops don't really solve the around-the-block problem at all, but they have just confirmed your ability to navigate the bot. Return to your algorithm and notice how easy it will be to modify your code to accomplish the task.

Modify your program so that your bot can drive itself around the closed course. Notice that it's a rectangle, so you will have to carefully adjust and test your count value for each segment of the track.

SAVE AND SUBMIT

Be sure to save your source code frequently! Always save to your own UCA Google drive. If you are using the UCA computers, save the program on the Desktop—but always save it in a second location. If it's on your Google drive, you won't need to be in the lab to access the files!

- Use the proper file name: Whatever you have named your program in the source code, you must use the correct filename for submission. Name your two-loop program lastnameLAB03, obviously using your own last name. It should already/automatically have the .bs2 file extension. Never submit word processor documents or .pdf files!
- Submit electronically: Your program is due at 6:00 PM on Tuesday, 13 February 2024. Submit via the Blackboard Assignment.
- Demonstrate your success: Once you have a program that successfully navigates the closed-loop course, you should be sure to demonstrate it to me! If you are able to come to class a little early, or stay a few minutes late, you can do a live demonstration of your bot's performance. Alternatively, you may use your phone to record a video of your bot in action. Your video should make it clear that it is you, using your assigned bot. Blackboard Assignments supports the submission of multiple video file formats, so you should be able to upload your video with your source code. Please note that demonstrated success here represents a full 50% of the assessment!

GRADING RUBRIC

Your submitted source code and video will be graded using the assessment rubric below.

ASSESSMENT	CRITERIA / VALUE		POINTS EARNED
BOE-Bot Walkabout Video	Navigation: BOE-Bot successfully navigates the closed-loop course; video submission (or live demo if possible)	15 points	
Lab 03 Program: Loop! DUE: Tue 13 Feb 24 (filename: lastnameLAB03.bs2)	Compilation: Program compiles cleanly	4 points	
	Execution: Program executes correctly (yields correct results)	6 points	
	Annotation: Program is sufficiently commented	5 points	