Lab 07: Real-Time Navigation
When You Gain Keyboard Control

INTRODUCTION

The last time we saw our little friend BOE, he was still lost in a maze, and our ambition got ahead of our programming skills. We discovered that it was easy to imagine what we wanted to do, but complicated to program it. So now what?

Sometimes you need to pause. Do we really need the autonomous course correction *right now*? Or can we work on building more programming skills now and revisit that problem later when we're better equipped to solve it?

The simplest way to control your BOE-Bot will be with keyboard commands. Back in the olden days, keystrokes were everything... because there was no such thing as a mouse or a joystick. The wheel had barely been invented, fire was just becoming A Thing, and MTV actually played music videos all day long...

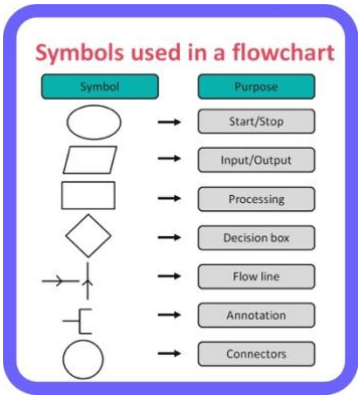
OBJECTIVES

The specific goals for this exercise are:

- Develop an algorithm for keyboard-controlled navigation
- Implement fundamental branching and looping structures
- Program your BOE-Bot to move using keyboard commands
- Install and test a servo-driven putter on your BOE-Bot
- Write a subroutine to enable your bot to play golf

CONSTRUCT THE ALGORITHM

We want to program the BOE-Bots to respond to keyboard commands; for example, keystroke "w" (or maybe "i") results in the bot moving forward, and "s" (or perhaps "k") moves the bot backwards. We also want keystrokes for left and right turns. Pick your own favorite keystroke combinations!



Without writing code, articulate exactly how the program should function. For example:

User enters keystroke
Keystroke calls specific subroutine
Bot executes the subroutine
Repeat until user quits

The loop structure is obvious, but how does a keystroke call a subroutine? Clearly it's possible, because vending machines! It's exactly how a vending machine works: you press a button to select a specific snack or drink. If you choose B17, you get Cheetos, but choosing B18 gets you Doritos. For example:

If keystroke is w, then call forward
If keystroke is s, then call backward

Now what does that look like in our programming language?

WRITE THE CODE

1. **Add to the existing.** Continue to add to and modify the dead-reckoning program you already have. Make a copy and rename to preserve your original motion program.
2. **Check your subroutines.** These should already be written and debugged, but if you have not done so already, you should construct the simple subroutines for forward, backward, left and right. The code fragment below should remind you of the subroutine syntax. Notice that the FOR-NEXT loop is no longer required and must be removed!

```
'      --- --- Forward Motion --- ---
Forward:
  PULSOUT 12,650         'Advance right wheel
  PULSOUT 13,850         'Advance  left wheel
  PAUSE 20               'Wait 20ms
RETURN
```

3. **Call them with a keystroke.** Decide which keystroke to use for forward motion. In this example, I am using "w" to move the bot forward and an "s" to move in reverse. Use the following syntax to call the proper subroutine at the user's request:

```
'Program: Click and Go
'{$STAMP BS2}          'Stamp directive
'{$PBASIC 2.5}         'Language directive


X VAR WORD        'Allocate memory for keystroke
DO UNTIL (X = "q" OR X = "Q")
  DEBUGIN X           'Accept input keystroke
  DEBUG X             'Display to the user
  IF X = "w" OR X = "W" THEN
    GOSUB Forward
  ELSEIF X = "s" OR X = "S" THEN
    GOSUB Backward
  'Keep elseif-ing until you have all the
  'keystrokes to fully maneuver!
  ENDIF               'Close the conditional
LOOP                  'Close the loop
END                   'End the program
```

Notice that the user can type either lower- or uppercase letters, and the program will respond correctly! Also notice the loop structure: this will permit the user to keep driving the bot until selecting to quit the program.

4. **Complete the conditional.** The IF-THEN-ELSEIF structure is a conditional: nothing happens unless the user input matches the set of keystrokes established as the conditions for motion! Within the loop, add the conditions for the right and left turns (for example, "a" to turn left and "d" to call the subroutine for right turns).
5. **Compile and test.** When you have completed the code, you should have keyboard control of your bot. It should move in the proper direction with the proper keystroke. Please note that if you press and hold a key (sending many keystrokes to the bot very quickly), you might notice some response rate hiccups. You can adjust your laptop's keyboard response rate, which can help. This is also partly why we are echoing the keystroke back to the debug terminal.

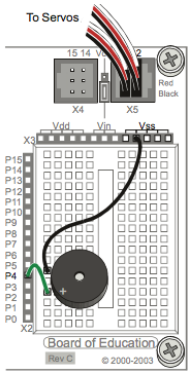
PROGRAM THE PUTTER

Unfortunately, we do not have enough attachments to give every student a putter. Extra robots are provided with putters already attached. These are located on the shelves along the south wall of CCCS 112. Develop a subroutine to operate the attachment, and modify your program to allow the user to play a round of golf!

6. Create a new program. You should now have a working navigation program. Do not overwrite or modify it! Use a copy of the nav program and alter it as necessary.
7. Which way is forward? When you examine the putter, you should notice that your bot, which has been programmed for front-wheel drive, is now a rear-wheel drive vehicle. Unless you *want* to putt backwards. You should be able to quickly modify your program for rear-wheel drive (literally, forwards is backwards, left is right, so swap the subroutine names).
8. Control the putter. Add two more subroutines: one to raise and the other to lower, the putter. The putter servo is the same as the servos which drive the wheels, so using the PULSOUT pin,spin is the proper command. To which pin did you attach the putter servo? 14? 15?
9. Complete the conditional. Within the main program loop, add the conditions for raising and lowering the putter. You might, for example, use keystroke "u" to raise the putter.

HONK IF YOU LOVE ROBOTS!

Blinking our LEDs was fun and easy, and it's pretty obvious that having visual signals is important. But what if you can't see the bot? An audio ping might be helpful. And back-up beepers exist for an obvious reason.



10. Wire the speaker. Use the diagram on the right to add a piezospeaker to your breadboard. In this example, the diagram shows the speaker wired to PIN 4. Go ahead and wire it to any unused pin, it doesn't have to be PIN 4, but avoid using PINs 0–2.
11. Create the Subroutine. Start by creating and naming a new subroutine and adding a keystroke subroutine call to the main body of your program.
12. Program the speaker. This is very straightforward. In fact, a single command is all you need to emit a sound:
    FREQOUT PIN, TIME, FREQUENCY.
    The PIN is whichever one you wired the speaker to, the TIME is in milliseconds, and the FREQUENCY is measured in Hertz. For example:

```
'A 2 second tone at 3000Hz = 3kHz:
FREQOUT 4, 2000, 3000.
```

Human hearing is sensitive from about 20Hz to 20,000Hz. Choose your favorite frequency from the table below.

Note Frequency Chart

	Octave 0	Octave 1	Octave 2	Octave 3	Octave 4	Octave 5	Octave 6	Octave 7	Octave 8
C	16.35	32.70	65.41	130.81	261.63	523.25	1046.50	2093.00	4186.01
C#	17.32	34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	4434.92
D	18.35	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	4698.64
D#	19.45	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	4978.03
E	20.60	41.20	82.41	164.81	329.63	659.26	1318.51	2637.02	5274.04
F	21.83	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	5587.65
F#	23.12	46.25	92.50	185.00	369.99	739.99	1479.98	2959.96	5919.91
G	24.50	49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	6271.93
G#	25.96	51.91	103.83	207.65	415.30	830.61	1661.22	3322.44	6644.88
A	27.50	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	7040.00
A#	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	7458.62
B	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	7902.13

13. Play a Round of Golf. The putting green has replaced the maze, and you should demonstrate the success of your program in the usual way: either film a video clip of your bot putting a hole-in-one, or a live demonstration of the same.

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!

- Demonstrate your success: Practice navigating and putting, then use your phone to document your skills with a short video. You should be sure that your video shows clearly who is controlling the Bot, and that the Bot operates properly (forward, backward, left and right, putting). It does not need to be long or fancy, the video simply needs to demonstrate that you have achieved the objectives of the exercise. Blackboard Assignments supports the submission of multiple video file formats, so you should be able to upload your video with your source code.
- 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 putter program lastnameLAB07, 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: All programs are due no later than 6:00 PM on Tuesday, 12 March 2024. You must submit via Blackboard.

GRADING RUBRIC

Your performance will be evaluated using the rubric below:

ASSESSMENT	CRITERIA / VALUE		POINTS EARNED
Lab 07 Program: Putt! DUE: Tue 12 Mar 24 (filename: lastnameLAB07.bs2)	Compilation: Program compiles cleanly	5 points	
	Annotation: Program is sufficiently commented	5 points	
	Execution: Program executes correctly (yields correct results)	6 points	
	Beeper: Piezospeaker is attached / controlled with a keystroke	7 points	
	Video: Proper operation is completely documented via video	7 points	