Mars RoversLab 04: Navigating  
By Dead Reckoning

When Correcting Your Course Isn’t an Option

# Cat in a shark costume riding a RoombaIntroduction

Why would you need a robot capable of self-navigation? I think a cat in a shark suit riding a Roomba is all the answer you need.

However, sometimes—as in the case of the Mars probes shown above—it’s pretty obvious: You can’t drive that thing in real time. Mars is (on average) about 12.5 light-minutes away from Earth; that means if you send a “go forward 10 meters” signal, it’ll be at least 25 minutes before you know whether you’ve driven over a cliff…

# Objectives

The specific goals for this exercise are:

* **Construct** an algorithm for completing a specific task
* **Translate** an algorithm into an executable program
* **Modify** an existing program to complete a more sophisticated task
* **Develop** flexible subroutines which can be reused easily
* **Program** your BOE-Bot to navigate a maze unassisted

# Re-Use! Recycle!

Recycle symbolWhenever possible, we will try to use the programs we have written previously to build the programs we want now.

Let’s build some subroutines that we will be able to re-use and modify to perform a variety of different tasks. A subroutine is a named piece of code that can be called upon multiple times within a program.

Notice that your previous program repeats the same tasks: move forward, turn, forward, turn, etc.. But the forward distance wasn’t the same each time.

A subroutine is a convenient construct that allows you to perform repetitive tasks without re-coding (or even copy/pasting). Wouldn’t it be nice to just be able to tell the bot “forward, 15” and have it know exactly what you meant? Well, it’s not quite that simple, but it’s sure not that hard.

The syntax for any subroutine is the same, and all subroutines are placed after the end of the main program. Like this:

'Program: Shark Cat Rides A Roomba

'{$STAMP BS2} 'Stamp directive

'{$PBASIC 2.5} 'Language directive

index VAR Byte 'Store current count

length VAR Byte 'Store max count

'Move forward 100 ticks (or however many)

length = 100 : GOSUB Forward

'Turn left 40 ticks (or however many)

length = 40 : GOSUB Left

'Continue to set the length and call the

'subroutine you need to set the path

END 'End of program

'\*\*\*\*\*\*\*\*\* --- Subroutine Library --- \*\*\*\*\*\*\*\*\*

' --- --- Forward Motion --- ---

' Bot moves forward a specific number of pulses

Forward:

FOR index=1 to length 'Set travel distance

PULSOUT leftWheel, CW 'Left wheel forward

PULSOUT rightWheel, CCW 'Right wheel forward

PAUSE 20 'Wait 20ms

NEXT

RETURN 'Go back to program

# Maze Challenge!

Go ahead and complete subroutines for backward motion, right turn, and left turn. Test them with different values for Schematic of Lego mazethe length. Always make sure to save your code!

Can you program your BOE-bot to navigate a maze?

Of course you can! Using the subroutines you have written, you can direct your bot to move any specific distance in any given direction! Once you choose a path, the rest is just counting…

There are three distinct engineering challenges here:

* How quickly and efficiently can you modify your existing code to run the maze successfully?
* How quickly your bot can complete the maze?
* How consistently can your bot run the maze successfully?

Strategize! Will the shortest path necessarily be the fastest path? You want to program a reliable solution; if your shortest-path solution only works once every five tries, is it the best solution? Maybe a longer path that your Bot can navigate more reliably is a better bet.

# Turn Signals!

Here’s a one more programming challenge for you to figure out: Can you modify your left and right subroutines to include a turn signal? You already know how to trigger the LEDs using the HIGH and LOW commands. How can you incorporate that functionality into this project?

Think about two different ways to achieve the goal: writing separate subroutines (for example, when you’re driving a car, you can turn without using your blinkers—they are independent of the turn) or by adding lines to the existing Left and Right subroutines (automatically blink whenever bot turns). You can choose to implement either option.

# Practice and Perfect Your Performance

The maze will be available for practice when CCCS 112 is not in use by other classes. Your Bear Card will unlock the room. It’s going to take a few trials to get your bot to navigate the entire maze successfully and consistently.

A successful trial will consist of the bot **starting on the** **green square**. At least two of the three contact points must be in the green. The bot should **stop on the red square**, again with two of the three contact points in the designated area.

You can demonstrate your success either live or via video.

# 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 maze navigation program lastnameLAB04, 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, 20 February 2024. You must submit via Blackboard.
* **Demonstrate your success:** Once you have a program that successfully navigates the maze, 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.

# Grading Rubric

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

| Assessment | Criteria / Value | | Points Earned |
| --- | --- | --- | --- |
| Lab 04 Program: Maze!  DUE: Tue 20 Feb 24 (filename: lastnameLAB04.bs2) | **Compilation:** Program compiles cleanly | 5 points |  |
| **Annotation:** Program is sufficiently commented | 5 points |  |
| **Subroutines:** Program properly defines and uses subroutines | 7 points |  |
| **Turn Signals:** Program implements use of LED blinkers | 6 points |  |
| **Success:** Bot completes maze, documented live or via video | 7 points |  |