UNIVERSITY OF UTAH
ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

ECE1020 **COMPUTING ASSIGNMENT 5**
N. E. COTTER **MATLAB® CHARACTER STRINGS: ASCII MESSAGES**

READING

Matlab® Student Version: learning Matlab 6, Ch 6-11, 6-22, and A-16
Mastering Matlab® 7, Ch 9.1-9.4

TOPICS

String manipulations

OVERVIEW

In our design of a communication link for an imagined rover on a distant world, we will want to specify what distance and in which direction we want the rover to move. It is convenient to describe these moves in English, such as '1 left' or '2 right' where the units are understood to be meters.

To use these commands in our communication system, we enter them as character strings and then map them to numbers that represent the commands. We map these numbers, in turn, to code words that we ultimately transmit.

PROCEDURE

In this assignment, you will use string commands to create a list of movement commands and corresponding code words for our rover.

Script File for calculations

- Using a text editor program on your PC, create a script file called **trip.m** containing matlab commands to perform the calculations in this assignment.
- Do **not** use semicolons at the ends of commands in your script files.

+2 pts Create a horizontal numerical array called `dist` containing the following four distance values: 1, 2, 2, and 1.

+3 pts The sprintf function is very much like the sprintf command in C. One difference is that Matlab allows the use of an array in place of the list of variables to be printed. Use this feature of fprintf() to write a one-line command that prints the values in dist in a string called `dist_str`. Print the values as one-digit integers with no spaces in between.

+5 pts Use the transpose operator ' to create a variable called vert_dist_str from dist_str. The variable vert_dist_str consists of rows, each of which is one *character* representing a distance for the rover to move. The first row of vert_dist_str, for example, will be the character '1'.

+5 pts Because of issues relating to how Matlab handles spaces when concatenating strings, we will make use later on of a vertical array containing 4 rows with one period on each row. Use the strvcat function to create a vertical array called vert_periods whose four rows are each equal to the following string: '.'

+5 pts Now create a string variable called `direct_str` containing the following four direction command words separated by spaces: 'backward', 'forward', 'left', and 'right'.

+5 pts The strtok function, like the strtok function in C, finds tokens in a character string. A token may be thought of as a word. You may choose the delimiter separating the words to be something other than the default white space, however. You might wish to use tabs, or even commas, for example.

In its simplest form, strtok returns the first word, (i.e., group of letters surrounded by white space), in a string. If we wish to extract the second word and so on, we use the following form of strtok:

[R,T] = strtok(S)

where S is the original string, R is the first token or word found in S, and T is what remains of S after the first token is removed.

Use strtok four times to extract the four direction words from direct_str. Save the four words in variables called d1, ..., d4.

+5 pts Use strvcat() to create a vertical array called vert_direct_str whose rows are d1 through d4. Now Create an array called using vert_changed_str whose rows are d2, d1, d4, d3. This will provide an array that has all the 'left' strings swapped with the 'right' strings and 'forward' with 'backward'.

+5 pts We now wish to put the distance and direction information side by side in strings such as '2 left'. We can use the strcat function to place vert_dist_str and vert_direct_str side-by-side. Unfortunately, the strcat function eliminates spaces. To work around this problem we first put periods between the distance values and the direction words. Our first row will be '1.forward', for example.

Use strcat to put vert_dist_str, vert_periods, and vert_direct_str side-by-side. The result will be an array, (call it vert_commands), with four rows. These rows are our commands, except for the unwanted periods in them. *If we had used cell arrays to hold the strings white space eliminations would not be a problem.*

+5 pts Use array concatenation (ie… a = [a;b]) to add the result of strcat(vert_dist_str,vert_period_,vert_changed_str) to the bottom of vert_commands. The resulting string array will have 8 rows. All possible command combinations will be represented.

+5 pts Use the strrep function on the first row of vert_commands to replace the period with a space. Leave the result in vert_commands. In other words, overwrite the existing vert_commands array location. To process the entire array would require this step to be performed eight times.

Matlab is optimized for array and matrix manipulation so understanding ways to process entire arrays with one command provides the best computational method. Unfortunately using arrays of strings doesn't provide a method to do this but cell arrays can be manipulate in this way.

+5 pts Create an cell array of spaces called cell_spaces by typing repmat({' '},8,1). Create a 8x1 cell array of periods called cell_periods. Finally, use cellstr( ) to convert vert_commands to a cell array called cell_commands. Now you have all the cell arrays needed to perform batch removal of the periods.

+5 pts Use the name of each of the cell arrays and strrep() to remove all to period from cell_command and place the processed values back into cell_command. Convert cell_commands back to an array of strings using called vert_commands using char().

Now that we have created a series of commands for our rover, we may translate the commands into code words for reliable communication with the rover.

+5 pts Our first step is to create a codebook. We could use the methods developed in the previous lab, but we'll simplify further by just entering the following matrix of code words:

```
codewords = [ '111010'; '101010'; '010101'; '111000';
              '011111'; '111111'; '011001'; '101101'   ]
```

The rows of this character array are the bits we would transmit to indicate a single and then double move in all directions respectively (ie… 1 forward, 1 backward and so on). For practice, <u>use the eval function to enter the above command</u>. In other words, turn the command into a string and let eval interpret it as a Matlab command. (hint: using a ' in a string is a problem)

+5 pts Create a list of all the possible command in the same order as code words in a cell array called poss_commands use semi-colon separators to create multiple rows.

Now we want to find a method of indexing that allows us to extract the appropriate rows of the *codewords* array as we scan through our list of commands for the rover. One approach is to use the strmatch function to find the indices of the rows in vert_commands where each of the commands appear.

+5 pts Use the strmatch command on vert_commands to find the indices of the rows where '1 right' appears by indexing poss_commands and storing it in a variable called ind.

+5 pts Now that you understand the syntax of strmatch lets search all of the commands. To do this we will use a standard programming concept to reuse a generic command multiple time without the requirement for editing the command to change the index.

Define an index called 'i=1' then perform to following command

```
>> ind(i) = strmatch(poss_commands{i},cell_commands), i=i+1
```

the first time we run this command i=1 and strmatch is used to search for the first command, '1 forward' and return its index in cell_commands as ind(1). The value of 'i' is increased by one and the command is completed. On the next run of the command the index is already set to search for the second command and provide its value to ind(2). By using this command eight times all of the possible commands are located and placed in the 'ind' array.

+5 pts Using comment lines in your script file, explain in detail what the following command line does. Include this command in your file.

```
>> transmit_data(ind,:) = codewords(1:8,:)
```

ROVER IMPLEMENTATION

+20 pts Now that we have a set of data to transmit we can send the commands to the rover. The code book and the mapping of responses is already provided. Using the program *commands.rxe* the rover can receive the eight commands by sending the codes to the mailboxes 2-9. The rover will run the commands in ascending order from the mailboxes once the execute command, '101010' is sent to mailbox 1. Write code to do the following:

- Connect the the NXT.
- Start the Program *commands.rxe*
- Send all of the commands and verify that the rover responds as expected
- Create a pause to provide enough time to complete the commands
- Stop the program
- Disconnect from the nxt

Run your script file by typing the name of the file without the .m

```
>> trip
```

If you make any changes in your **trip.m** file, be sure to run the following Matlab command to insure that Matlab reads your file again the next time you run it:

```
>> clear all
```

End of Diary

```
>> diary off % Close the diary file.  Look for the diary in e.g., c:\matlab\work directory.
```

E-mail your script file (trip.m) and your diary file to your TA, (as two separate e-mails). In the Subject line of your e-mail, be sure to put Your Name, "ECE1020 Comp5," and the file name, (e.g. trip.m). Also, print out the files and hand them in to the TA or to the ECE1020 locker.