UNIVERSITY OF UTAH
ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

ECE1020 **COMPUTING ASSIGNMENT 7**
N. E. COTTER **MATLAB® CONTROL FLOW: FILTERING**

READING
Matlab® Student Version: learning Matlab 6, Ch 6-2 to 6-6
Mastering Matlab® 7, Ch 11

TOPICS
Control flow: For, while, if, switch

OVERVIEW
In previous assignments, we have considered all of the parts from which we could construct a communication link between a base transmitter and a receiver:

- translating commands from a control language, (expressed in strings such as 'right' '1'), into a sequence of binary numbers,
- creating a codebook for translating binary information into longer binary strings (allowing us to perform error correction later on),
- transmitting sinusoids of two different frequencies representing 0's and 1's in the codewords,
- using the radiation pattern for an antenna to determine the power of the signal at the receiver (as a function of distance and direction),
- using correlation and summation (i.e., a matched filter) to determine whether each received bit signal (plus noise) represents a 0 or 1,
- calculating the distance between the received binary bits and each codeword to find the best match,
- translating the index of the codeword into its corresponding bit pattern (i.e., the binary information we had at the very beginning of this process), and
- interpreting the bit pattern as a command to be carried out (such "right 2").

We are now ready to simulate an entire system (with some simplification) in a fashion that allows us to process an arbitrarily long sequence of input commands.

PROCEDURE
In this assignment, you will use for, while, if and swtch statements to write a complete communication system simulator. Then you will edit your code to send commands to the rover. The command received by the rover may be corrupted so the commands received by the rover will be sent back for verification by your code.

Script File for calculations

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

+5 pts Create a string variable (having eight rows) called command_str containing the commands 'right', 'backward', 'right', ' forward ', 'left', ' forward ', 'right', and ' backward '.

+10 pts Use a for loop to interpret the command strings:

- processing the strings (i.e., 'right', 'left', 'backward', or 'forward') in order, and (continued in next item)
- translating the strings (using a switch statement) into numbers (as opposed to binary codes used earlier):
  - ✓ backward = 0
  - ✓ forward = 1
  - ✓ left = 2
  - ✓ right = 3
- Concatenate the eight command numbers to create one horizontal array (of 8 numbers) called command_num. Then end the loop.

+5 pts Create a 4x6 array called codewords that consists of four 6-bit codewords. You may create the codewords in any way you desire, including code you wrote from earlier assignments.

+10 pts Use a for-loop to translate the numbers from command_num into the corresponding codewords. In other words, if the number is 0, use the first codeword; if the number is 1, use the second codeword, and etc. (Remember that Matlab indexing is one-based.) Concatenate the codewords to create a binary array (with only one row) called command_code (containing 48 bits for the 8 codewords for the 8 command numbers for the original 8 commands). These are the encoded bits to be transmitted to the rover.

+10 pts Rather than translating the bits in command_code into cosine waveforms and using the antenna radiation calculations from earlier assignments, we will simply create an array indicating which bits are received incorrectly. Using the randn function and a logical operator, create an array called "noise" consisting of 48 zero and one values. The value should be zero when the entry in the array returned by randn is less than 1.5. The 1's in "noise" will correspond the erroneous bits in the received code words.

+5 pts To flip the erroneous bits to the wrong value, compute the exclusive-or of the command_code and noise arrays and place the result in an array called command_recv. (On average the noise will change only a few bits.)

+5 pts In conjunction with the following steps, use an outer "for" loop to process the command_recv array one codeword at a time until the end of the array is reached. The goal is to decode the incoming commands.

+10 pts The first operation you will perform in the loop is to extract the next received codeword from command_recv and place it in an array called codeword_recv. In other words, extract 6 bits from the front end of command_recv and place them in codeword_recv. Don't forget to remove these six bits from command_recv for the next

time through the for-loop, but check to see if the array is already empty first so you don't generate an error.

+10 pts Using Hamming distance, (the number of bits that are not equal for two binary numbers), determine which codeword is closest to codeword_recv. Append the corresponding command number, (e.g., 2 if the codeword corresponds to the command 'left'), to an array called command_dec. When the loop is finished, command_dec will contain our sequence of decoded commands.

+10 pts Using a "switch" command, translate the command number in command_dec into the corresponding command string, (e.g., 'left'), and add that string as another row to a string array called command_str_recv. If we are lucky, command_str_recv will be the same as the command_str that we started. (With the level of noise we are using, however, you will get one error approximately half the time.) Where we have decoding errors, our rover will turn the wrong direction or go the wrong distance.

Rover implementation

Now that you have a simulated system the next step would be to send the command codewords to the rover via the mailbox system and see the resulting movements that the rover decodes. The commands that are sent to the rover will be corrupted by noise generated on the rover much like the simulated noise that you generate. The rover will communicate the codewords back for you to decode and see what it received. The “try” and “catch” control flow will be used from codeword reception.

+20 pts Create a copy of your simulated code called command_respond.m
In the section that creates codewords add the following:

- Store your codewords as an array of cell strings (ie.. codewords={‘100101’; … }). If you generated them as an array of logicals or doubles you can convert them to cell strings by the following method

  ```
  temp = num2str(codewords')'
  codewords = cellstr(temp(1:3:end,:))
  ```

- Connect to the NXT
- Run the program codewords.rxe and use pause(1)
- Use a for-loop to send codewords to mailboxes 1,2,3,4 and execute the change by sending “101010” to mailbox 0 and use pause(15)
- Stop the program and use pause(1)

The next section should be added after *command_code* is defined to send the command codes to the rover.

- Run the program *command_respond.rxe* and pause(1)
- Use a for-loop to send each command (realizing each command is a cell string)
- Execute the commands using mailbox 0 and ‘101010’

- Note the display on the rover: if a star appears in the lower left-hand corner bit errors have occurred and the rover has pick the closest codeword

As each of the commands sent by the last step are decoded the codeword has the possibility of been corrupted so the value that the rover decodes has been place a mailbox to be retrieved as part of you script. The main problem is that if a there is not a codeword available when *readmailbox()* is used an error is returned and matlab is halted. To fix this problem matlab provides "try" and "catch" to deal with functions that return error that might halt execution.

Delete the lines that generates *noise* and performs the *xor()* of *command_code* and *noise* and instead do the following:

- Set command_recv equal to an empty array.
- Create a variable call *len_command* and set it equal to the length of *command_code*
- Create a variable called *count* and initialize it to the value of *len_command*.
- Add a while loop that continues as long as *count*>0
- Inside the loop add the following code:

```
try [message inbox] = readmailbox(nxt,10,0,1);
        command_recv = [command_recv str2num(message)'];
        count = count-1;
catch
        pause(0.5)
End
```

- Add a comment to each line indicating its function
- After the while-loop pause(1) and stop the program and delete the nxt
- Lastly, you may need to fix the line that computes the hamming distance if you only created *codewords* as a cell array. To convert back from cell string array to an array of doubles do the following:

```
codewords = double(cell2mat(codewords(:))-48)
```

Assignment Wrap-up

Run each of your scripts files
Use a diary file to capture the output of *control_flow* and *command_respond*.

If you make any changes in either 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

E-mail your script files (control_flow.m and command_respond.m) and your diary file to your TA, (as separate e-mails). In the Subject line of your e-mail, be sure to put Your Name, "ECE1020 Comp7," and the file name, (e.g. control_flow.m). Also, print out the files and hand them in to the TA or to the ECE1020 locker.