**Project 2 - Morse Code Decoder**

This is a team or individual assignment. You may work with one other person. Both of you

should submit your functions in the quiz server to receive full credit.

**Background**

Morse Code is a way to communicate a message with a series of dots and dashes, and is still used

today in avionics radio communication. The dots and dashes uniquely represent characters such as alphabet letters and numeric digits. Decoding the audio signal needs to be robust against noise to effectively communicate the message. Signal processing is a foundational technology in our digital world and an important application of electrical engineering.

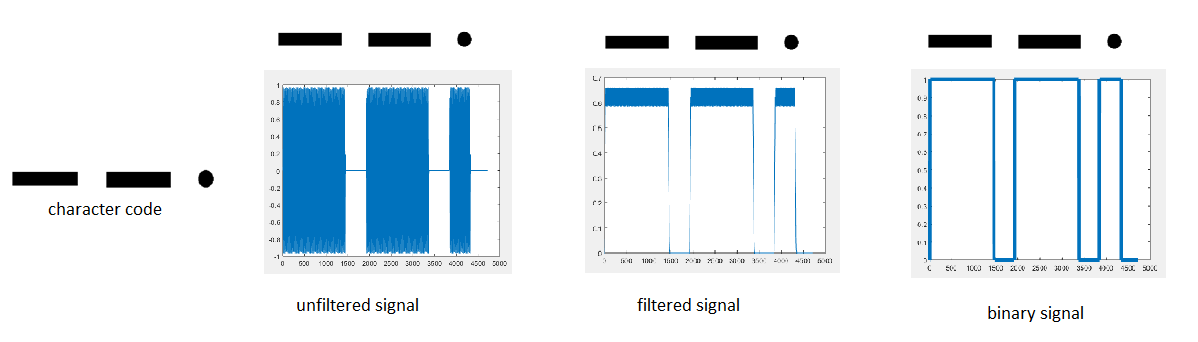
Figure 1 – Morse Code Table



In this lab, you will be building a morse code decoder which takes an audio file containing morse code as input and outputs the message.

Here is how it works: The audio wav file of the morse code message is converted to a column vector of floats representing an unfiltered signal. Then the vector is filtered using a half-wave rectifier and a slow-wave filter. Next the signal is converted to a binary signal (1s and 0s).

Figure 2 Original signal, filtered signal, and binarized signal



The binarized signal is much easier to work with. Where the signal is 1 (high), there is an encoded dot or dash. Where the signal is 0 (low) there is a space between the dots/dashes, a space between encoded characters, or a space between words. We will call these regions of 1s or 0s tokens. To extract the tokens, we look where the signal changes between 1 and 0. The distance between these changes are the duration of time of that token. Once the tokens are extracted, we can look up what character is represented by the tokens.

**Getting Started**

You will be working on this project in MATLAB. When you are finished with each part, submit your function in the quiz server for grading.

Up to 20 points will be deducted for bad style! Include comments under the function definition statement using the format described in MATLAB Function ‘docstring’ Format Rules. Use meaningful variable names and a consistent variable naming convention. Use indentation and comments to make it readable.

|  |  |  |
| --- | --- | --- |
| **Glossary** | | |
| **Term** | **Description** | **Examples** |
| dot | . |  |
| dash | - |  |
| tokens | region of 1’s and 0’s that represent dots and dashes | 111111  000  111 |
| character code | sequence of dots and dashes that represent one alphanumeric character in the Morse Code table | …  -.- |
| character | alphanumeric character | A  B  2 |
| string code | a string containing one or more character codes | -.- --- -..- |
| message | a sequence of words | HI THERE |

**Problem 1**

Download ***code\_to\_char.m*.** This functionaccepts a string of dots and dashes representing a single character code and returns the alphanumeric character that it represents in Figure 1.

|  |  |
| --- | --- |
| **Test** | **Result** |
| disp(code\_to\_char('--.')) | G |
| disp(code\_to\_char('---')) | O |
| disp(code\_to\_char('..---')) | 2 |

Define a function named ***str\_code\_to\_message*** that accepts a string of dots and dashes that represents one or more character codes and returns the corresponding message. This function should call ***code\_to\_char*** for each single character code. Each character code is separated by one space and each word is separated by a ***/***

|  |  |
| --- | --- |
| **Test** | **Result** |
| disp(str\_code\_to\_message('… --- …')) | SOS |
| disp(str\_code\_to\_message('.... ../- .... . .-. .')) | HI THERE |
| disp(str\_code\_to\_message('.----/..---/...--/....-/.....')) | 1 2 3 4 5 |

**Problem 2**

Define a function named ***tokens\_to\_str\_code*** that accepts a matrix and a scalar value representing a time\_unit. The token matrix has 2 columns and multiple rows. Each row represents either a dot, dash, a short space, a long space or a long-long space. This function uses the rules in the table below to map the token information for one row to either a dot, a dash, a forward slash or a blank space.

|  |  |  |
| --- | --- | --- |
| **Column One (Token Value)** | **Column Two**  **(Token Duration)** | **Converts to:** |
| 1 | greater than 4 \* time\_unit | * (dash) |
| 1 | greater than the time\_unit but less than 4 \* time\_unit | . (dot) |
| 0 | greater than 8 \* time\_unit | / (space between a word) |
| 0 | greater than 4 \* time\_unit but less than 8\* time\_unit | space (space between alphanumeric characters) |
| everything else can be ignored – doesn’t represent anything | | |

|  |  |
| --- | --- |
| **Test** | **Result** |
| token\_mat =  [1 5; 1 6; 1 5; 1 7; 0 20; 1 5; 1 5; 0 40;  1 5; 1 20; 1 6; 1 5; 0 21; 1 19; 1 20; 1 20; ];  time\_unit = 4;  disp (tokens\_to\_str\_code(token\_mat, time\_unit)) | .... ../.-.. --- |

**Problem 3**

Define a function named ***binary\_to\_tokens*** which accepts a column vector of binary values (0’s and 1’s) and counts how many 0's and 1's appear in sequence. The function should return a matrix where column 1 contains the token value (0 or 1) and column 2 contains how many appear in sequence.

|  |  |
| --- | --- |
| **Test** | **Result** |
| binary\_signal = [ 1; 1; 1; 1; 0; 0; 1; 1; 1; 0; 0; 0; 0];  disp(*binary\_to\_tokens* (binary\_signal) ) | [1 4; 0 2; 1 3; 0 4] |
| binary\_signal = [ 0; 0; 1; 1; 1; 0; 1; 1; 1; 0; 0; 0; 0;0];  disp(*binary\_to\_tokens* (binary\_signal) ) | [ 0 2; 1 3; 0 1; 1 3; 0 5] |

**Problem 4**

Define a function named ***signal\_to\_binary*** which accepts a column vector of positive and negative float values (representing an unfiltered signal) and a scalar value representing a threshold value. This function should return a binary signal column vector (0’s and 1’s) representing the unfiltered signal. Use the following algorithm below to create the binary signal:

1. Create a column vector containing the absolute value of the unfiltered signal.
2. Filter the absolute value from step 1 using a moving average\* with a window size of 20 to create a column vector of signal averages.
3. Apply > threshold to the averages to return a signal that contains only 1 or 0. The output should be 1 where the signal is above the threshold and 0 otherwise. (Don’t use a loop!)

\*To compute a moving average, you can use the filter function as follows:

filtered\_array = filter(coeff, a, x);

x – the array containing the data you want filtered

a – set it to 1

coeff – array containing the filter coefficients

set this array based on the window size you want to use:

for example: a window size of 4, coeff should contain 4 elements all equal to 1/4

a window size of 8, coeff should contain 8 elements all equal to

**Problem 5 – Put it all Together and Test your Program**

Download ***morse\_decoder.m*** and all of the \*.wav files from Canvas and place them in your code folder.

NOTE: If you are using octave-online.net you need to instead download all of the \*.mat files and upload them to your online folder because it does not support audioread.

Open ***morse\_decoder.m*** and take a look at the code. You do not need to add or change anything to this script. The first section processes the audio file and creates a column vector representing the unfiltered signal. The rest of the Sections use your functions to decript the message.

To test your code – in the MATLAB Command Window – call morse\_decoder and pass it one of the .wav files you downloaded from Canvas.

For example:

|  |  |
| --- | --- |
| **Test** | **Result** |
| disp(morse\_decoder('SOS\_morse.wav')) | SOS |
| disp(morse\_decoder('alphabet\_morse.wav')) | ABCDEFGHIJKLMNOPQRSTUVWXYZ |

If you want to generate your own Morse code messages to test in your program, go to <http://morsecode.scphillips.com/translator.html>and type a message and Download as a .wav file (make sure the settings are 550hz and 20wpm).

**Completion / Submission**

Submit all your functions in the last question to be graded for style. Make sure you included

your ‘docstring’ comments for each function. Use meaningful variable names and a consistent variable naming convention. Use indentation and comments. Your code should be readable.