# THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY ISDN 2602

**Laboratory 4: Source and Channel Coding (5%)**

**Answer Sheet**

Please write down your answer here and submit your answer on GitHub by Wednesday (Oct 29th) 23:59

***Part I: Source Coding***

# Task 1 – Length of the bit streams

In this task, we will compare the lengths of the bit streams for four source coding algorithms applied to a black-and-white image: "raw" image encoding, run-length encoding with lengths encoded as 8-bit binary numbers, and run-length encoding with lengths encoded by Huffman coding with one or two dictionaries.

# Check Point:

1. Write down the lengths of the bit streams using “raw” image encoding and the run-length encoding. Is the run-length code better than the raw encoding? **Explain why**.

Raw-length encoding is better than raw encoding as it is more efficient with data storage, consuming less space overall.

1. Type “help transpose” in the command window to learn how to perform matrix transpose operation on a matrix in MATLAB. Revise the MATLAB codes so that the image will be rotated along the diagonal. Then, write down and compare the lengths of the bitstreams for these four source coding algorithms before and after the rotation. **Explain why**.

Raw encoding: 250 000

Run-length: 301688

Huffman encoding: 117374 100981

Transposed raw encoding: 250 000

Transposed Run-length: 196680

Transposed Huffman encoding: 134892 120565

The transposed image means that the Huffman encoding is less efficient at reducing the occupied size as the patterns do not repeat as much as the original image (they have been tilted to the side), meaning the Huffman encoding technique does not perform as well.

***Fill in the answers to the blanks and Show your result to the TA.***

# Task 2 – Huffman code

In this task, you will generate the Huffman code for a set of run-lengths, and use it to encode the run- lengths of black or white pixels. You will find that Huffman coding enables us to encode the sequence of run lengths using fewer bits than the standard 8-bit encoding.

# Check point:

1. Find an optimal dictionary to represent these 11 symbols using the symbol probabilities and the Huffman coding algorithm. Once you have found it, replace the value of **dict** defined between the line:

*% % % % Revise the following code to generate a valid and efficient dictionary % % % %*

and

*% % % % Do not change the code below % % % %*

The remaining part of the code uses this dictionary to encode the run lengths, and to measure the length of the resulting bit stream. It also checks whether the dictionary is valid by reconstructing the image from the run lengths encoded by the dictionary using the function **huffman\_encode\_dict**. If your dictionary is correct, the original and reconstructed images should be the same and the **size\_huffman** should be equal to 117374.

# (Commit the revised codes to GitHub. Show your results to TAs.)

1. Attach the corresponding Huffman tree of the revised optimal dictionary.

A white board with writing on it

AI-generated content may be incorrect.

***Fill in the answers, commit the revised codes to GitHub***

***and Show your result to the TA.***

***Part II: Channel Coding***



# Task 3 – (n,k) block code decoder and Error Correction Capability

In this task, we will implement the (n,k) block code decoder and compare the error correction capability of the repetition code, hamming block code, and no error correction code.

# Check point:

1. Generate a figure with three curves representing the BER performance.

# A graph of a graph with different colored lines AI-generated content may be incorrect.

# （Show your results to the TA）

1. Write down/Insert a screenshot of the modified code in “**blk\_decoder.m**”.

A screen shot of a computer code

AI-generated content may be incorrect.

Graph after fixing the code:

A graph of a graph with a line

AI-generated content may be incorrect.

**(Commit the revised codes to GitHub. )**

1. Based on your observations, which coding scheme performs the best? **Explain why**.

Based on my observations, after fixing the error in the code it seems that the Hamming Block Code preforms the best, as it matches the no error code curve much more than the repetition code on the curve while being more efficient:

Repetition Code: 1 message bit / 3 total bits = 33% efficiency

Hamming Code: 4 message bits / 8 total bits = 50% efficiency

No Coding: 1 message bit / 1 total bit = 100% efficiency

***Fill in the answers, commit the revised codes to GitHub***

***and Show your result to the TA.***

**----------------------------------End-----------------------------------**