# 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 encoding data = 500 x 500 = 250,000 bits

Size run length = 301,688 bits

Run-length encoding is not always better than raw encoding. It depends on image characteristics. For images with long runs of identical pixels, run-length encoding reduces size significantly. However, for images with frequent pixel changes, the overhead of storing run lengths makes the encoded stream longer than raw encoding.

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 image encoding length = 250,000 bits

Run length encoding length = 196680 bits

After transposing, the run-length encoded bitstream becomes shorter because the image structure changes. Run-length encoding efficiency depends on the length of consecutive identical pixels. In the original orientation, runs were short, leading to high overhead. After rotation, runs became longer, reducing the number of run-length pairs and improving compression. However, raw encoding remains constant because pixel count does not change.

***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 graph paper with text and numbers

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.

# 

# （Show your results to the TA）

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

if isequal(S, [1 0 1 0])

msgblk(1) = ~msgblk(1); % D1

elseif isequal(S, [1 0 0 1])

msgblk(2) = ~msgblk(2); % D2

elseif isequal(S, [0 1 1 0])

msgblk(3) = ~msgblk(3); % D3

elseif isequal(S, [0 1 0 1])

msgblk(4) = ~msgblk(4); % D4

end

end

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

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

The worst scheme is no ECC as there is no error correction, every bit flip during transmission leads to a permanent error.

(8,4) block code is the best as it corrects 1-bit errors using parity bits with less redundancy than repetition code. This scheme offers better efficiency and protection over noisy channels.

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

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

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