# 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 : 250,000

Runlength : 301,688

The run-length code is worse than the raw encoding because there is a large amount of variation in the image. As a result, the sum of the 8-bit values needed to encode each row of consecutive identical pixels uses more data than simply representing each pixel as a bit, as was done in the raw image 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**.

|  |  |  |
| --- | --- | --- |
| Source Coding Algorithm | Before Rotation | After Rotation |
| Raw Encoding | 250, 000 | 250,000 |
| Run-length Encoding | 301, 686 | 196,680 |
| Huffman Encoding (1 Dictionary) | 117,374 | 134,892 |
| Huffman Encoding (2 Dictionaries) | 100,981 | 120,565 |

The raw encoded bitstream has the exact same size after rotation because the number of black and white pixels does not change, only their positions. The run length encoded bitstream decreases in size after rotation. When transposed, the image has fewer unique sequences of consecutive black or white pixels, so the run-length encoder uses less data than counting pixels individually. The Huffman Encoding algorithms use less data than both the raw-data and run-length algorithms before and after the rotation due to the efficiency of storing probabilities for every unique row of consecutive identical pixels, instead of storing the length of every row of consecutive identical pixels. However, the Huffman Encoding algorithm increases in size after the rotation. When the image is rotated, the array of run lengths decreases in size and the probability of each given length appearing in the array increases. As a result, the amount of data needed to represent each run length increases.

***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 screenshot of a cellphone

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 curve

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 white paper with black and white text

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**.

The repetition code scheme performs the best. This is due to the fact that the block code scheme can only correct single bit errors, whereas the (3,1) repetition code can correct errors across multiple bits.

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

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

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