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**.

Size of raw data= 250000bits
Size of run_length= 301688bits
The run_length encoding is worse than raw encoding, as the file size is increased by 51688bits
The situation happened because the image is full of texts, which made the image very complexed. With many short runs, the overhead of 8 bits run length code become larger than just storing the raw 1 bit pixel

---

2) 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**.

Before transpose -> after transpose
Raw: 250000bit -> 250000bits
Run length: 301688bits -> 196680bits
Huffman single: 117374bits -> 134892bits
Huffman two: 100981bits -> 120565bits

The transposed image made the image have string vertical alignments
Before transpose, the horizontal scan cut across vertical pattern, this cause many short runs. While after transposing the vertical pattern enables long runs

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:

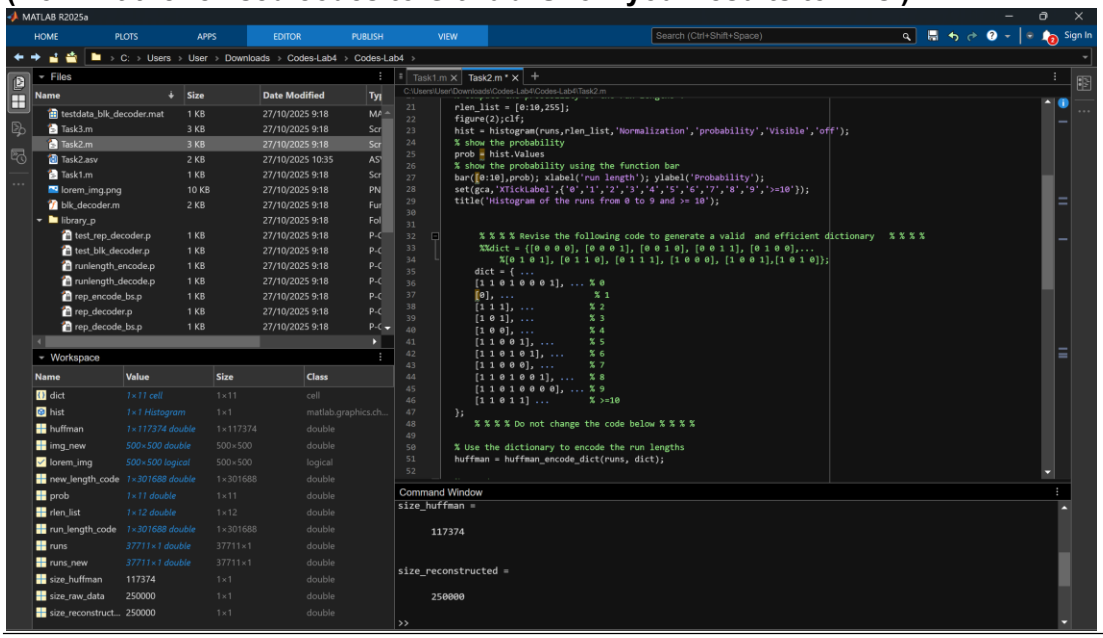
   % % % % Revise the following code to generate a valid and efficient dictionary % % % %

   and

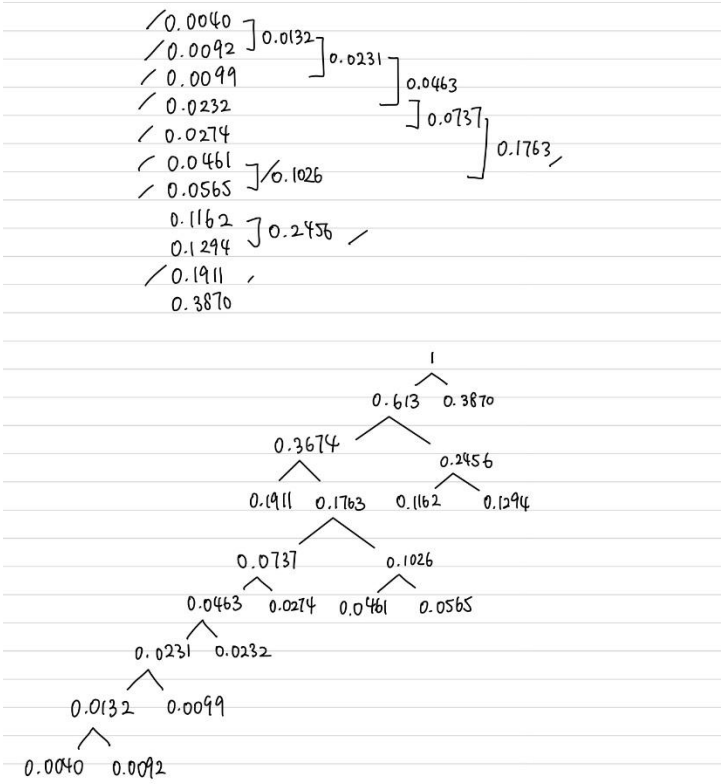
   % % % % Do not change the code below % % % %

   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.)



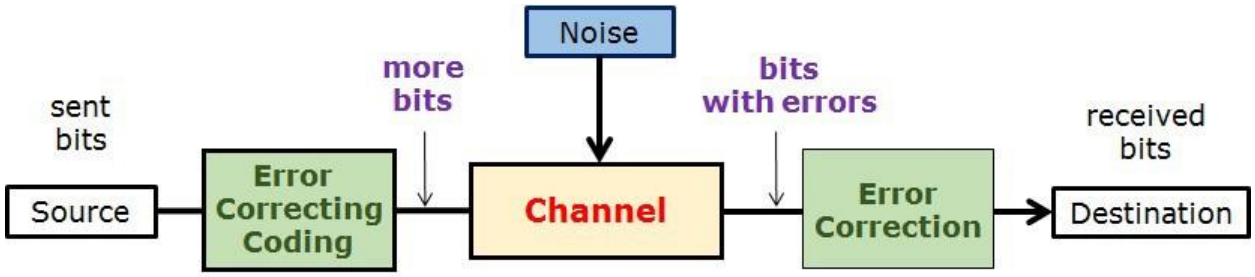
2) Attach the corresponding Huffman tree of the revised optimal dictionary.



<u>Fill in the answers, commit the revised codes to GitHub</u>

<u>and Show your result to the TA.</u>

Part II: Channel Coding

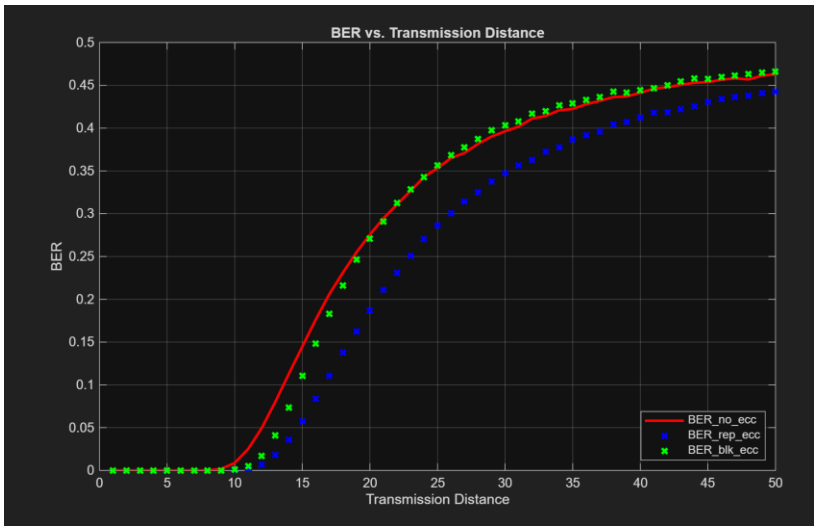


Task 3 – (n,k) block code decoder and Error Correction Capability

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)

2) Write down/Insert a screenshot of the modified code in "**blk_decoder.m**".

```
if S == [1 0 1 0]
        msgblk(1) = not(msgblk(1));
    elseif S == [1 0 0 1]
        msgblk(2) = not(msgblk(2));
    elseif S == [0 1 1 0]
        msgblk(3) = not(msgblk(3));
    elseif S == [0 1 0 1]
        msgblk(4) = not(msgblk(4));
    end
end
```

(Commit the revised codes to GitHub.)

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

The Hamming code (blue curve) performs the best, as it maintains the lowest BER across the range of transmission distances as it maintains the lowest Bit Error Rate (BER) across all transmission distances,

Its superior performance comes from efficiently correcting single-bit errors using a smart parity-check system

Fill in the answers, commit the revised codes to GitHub
and Show your result to the TA.

----------End----------