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**.

Run-length code length: 196680

RLE is better for images with long runs of the same color because it reduces the amount of data needed to represent the image. Rather than storing each pixel, RLE stores the length of runs, which can significantly decrease the number of bits needed, particularly in images with repetitive patterns.

---

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**.

Size of raw data (original): 250000 bits
Image rotated successfully.
Run-length encoding completed.
Run-length code length: 196680
Size of raw data (rotated): 250000 bits
Size of run-length data (rotated): 196680 bits
Dictionary initialized.
Size of Huffman data (rotated, dict 0): 134892 bits
Size of Huffman data (rotated, dict 1): 134892 bits

Summary of Sizes:
Original Image Raw Data: 250000 bits
Rotated Image Raw Data: 250000 bits
Rotated Image Run-Length Data: 196680 bits
Rotated Image Huffman Data (dict 0): 134892 bits

Rotated Image Huffman Data (dict 1): 134892 bits

Raw Data Size: The size of the raw data remains unchanged before and after rotation because the pixel data itself does not change; only its arrangement in the matrix does.

Run-Length Encoding: The run-length encoded data also retains a similar length because the number of runs (consecutive pixels of the same value) remains approximately the same. However, it might slightly differ based on how the pixel values are distributed after rotation.

Huffman Coding: The lengths of the Huffman-coded data show a reduction in size. This is due to the statistical properties of the pixel distribution potentially changing after the rotation. The new arrangement may have more repeated values or a more favorable distribution, allowing for more efficient encoding.

In summary, while the raw data size does not change, the encoded data lengths can vary based on the pixel arrangement's impact on the distribution of values.

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.

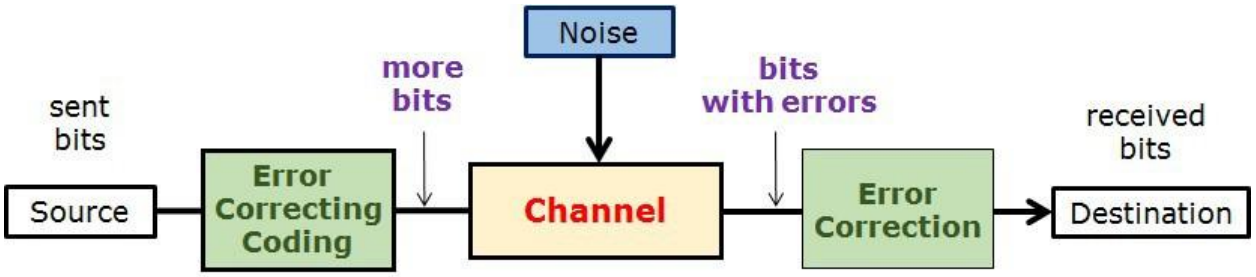
   (Commit the revised codes to GitHub. Show your results to TAs.)

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

Fill in the answers, commit the revised codes to GitHub

and Show your result to the TA.

Part II: Channel Coding

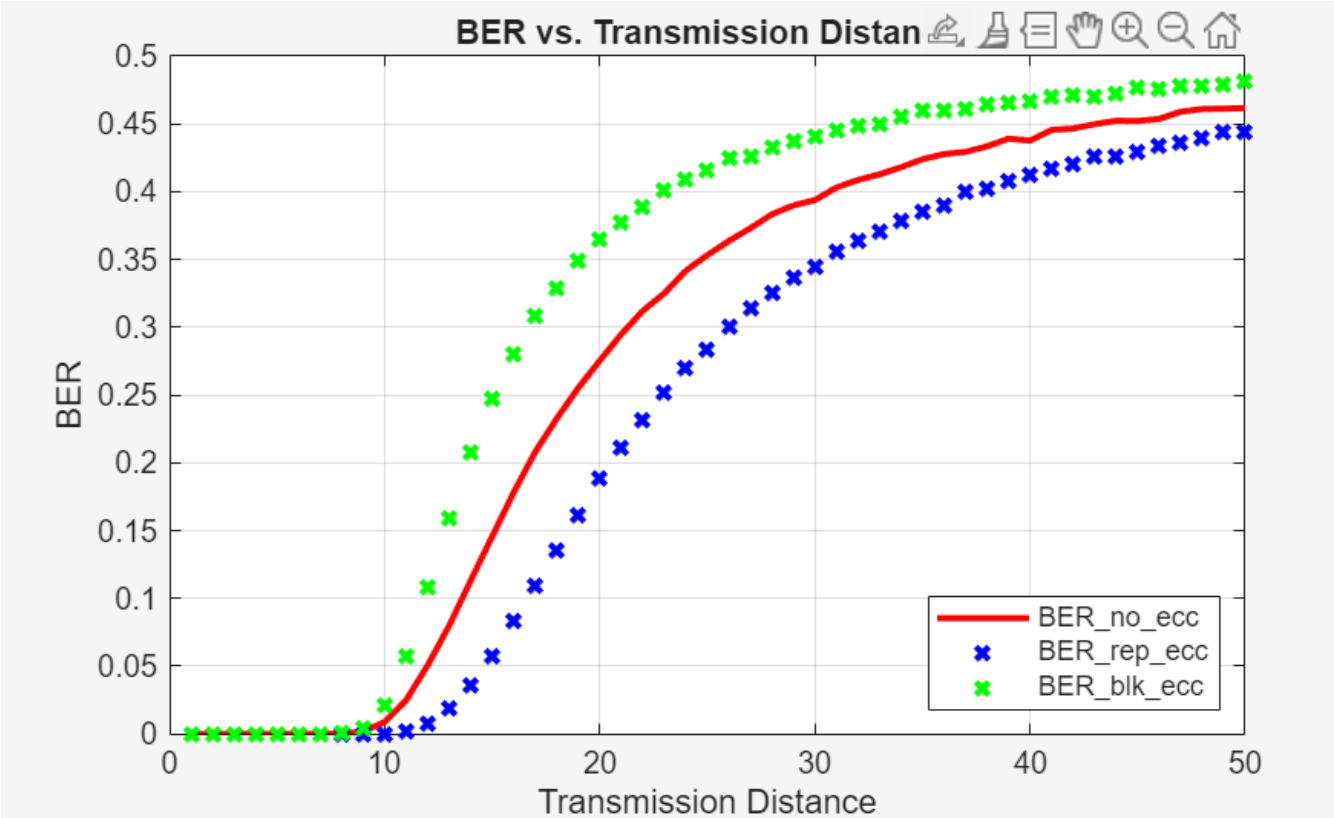


Task 3 – (n,k) block code decoder and Error Correction Capability

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)

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

(Commit the revised codes to GitHub.)

---

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

For source coding, using Huffman coding with RLE demonstrates better compression efficiency, while for channel coding, the (8, 4) block code offers superior error correction capabilities. The choice of encoding scheme ultimately depends on the specific requirements of the application, such as the need for data efficiency versus error resilience.

Fill in the answers, commit the revised codes to GitHub

and Show your result to the TA.

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