# 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**.

The bit stream length using the raw image encoding is 250000, and the run length code using 8-bit is 301688, while the run-length code using Huffman is 117374 (using same dictionary for both black and white runs), and 100981 (using separate dictionaries for black and white runs). Therefore, the Huffman coding run-length code is better than raw coding, but the 8-bit run-length coding is not.



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**.

The run length code using 8-bit is only 196680 and the Huffman is 134892 and 120565 respectively for same and separate dictionaries.

This is because letters are more vertical than they are horizontal; when the image is rotated the runs are longer because there is more space between each black and white (strokes and space-between-lines). Therefore the 8-bit run length is shorter because it is more efficient for longer run lengths; however the Huffman coding is worse because it is better for shorter runs. Because the 8-bit RLE costs 8 bits regardless of run length; the Huffman RLE costs less bits for shorter runs, but more if run lengths are longer than run >=10 (escape code + 8 bits for length)



***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

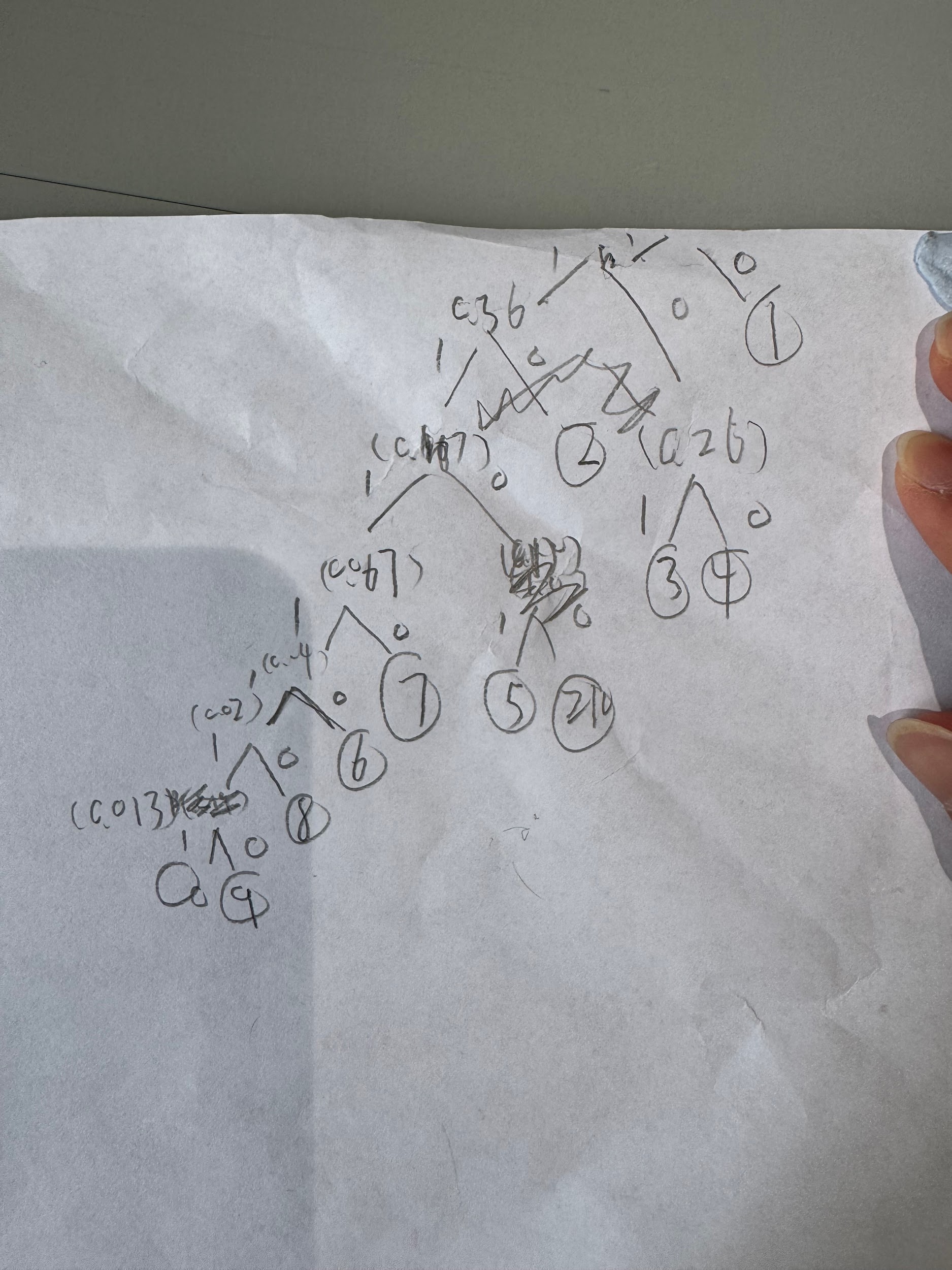
*% % % % Do not change the code below % % % %*

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.

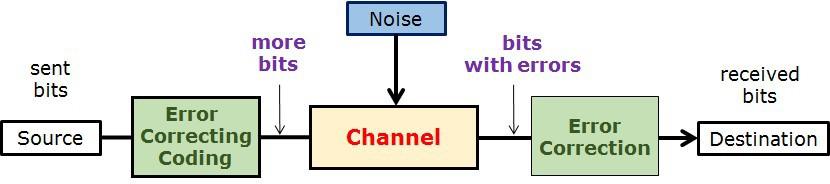




***Fill in the answers, commit the revised codes to GitHub***

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

***Part II: Channel Coding***

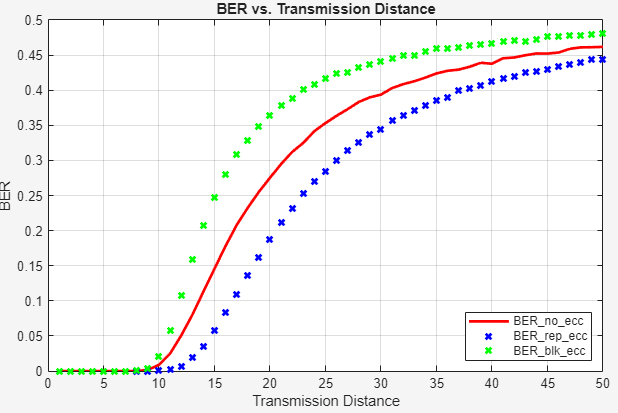


# Task 3 – (n,k) block code decoder and Error Correction Capability

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**”.

% % Modify the code below

if (S(1)==1 && S(3)==1)

msgblk(1)=not(msgblk(1));%when one bit error is in msgblk(1)

elseif (S(1)==1 && S(4)==1)

msgblk(2)=not(msgblk(2));%when one bit error is in msgblk(2)

elseif (S(2)==1 && S(3)==1)

msgblk(3)=not(msgblk(3));%when one bit error is in msgblk(3)

elseif (S(2)==1 && S(4)==1)

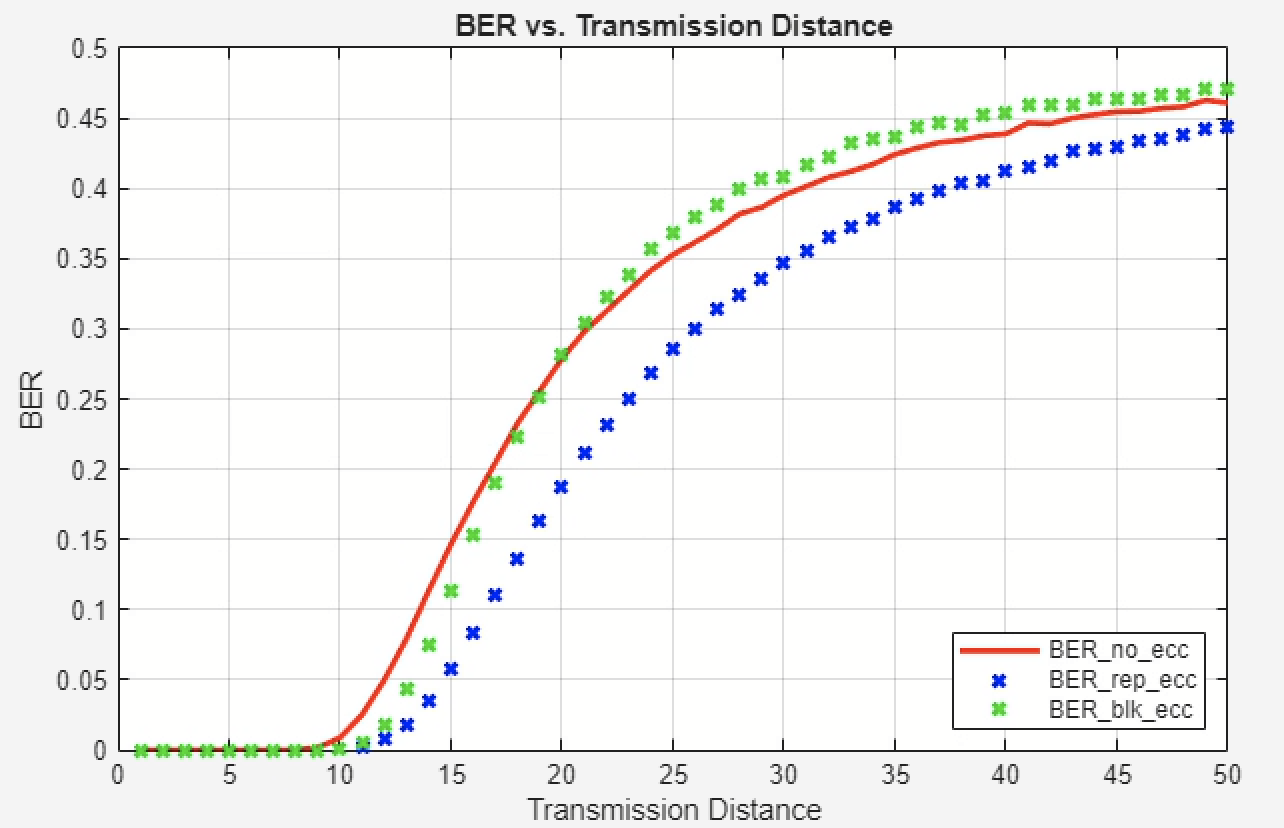
msgblk(4)=not(msgblk(4));%when one bit error is in msgblk(4)

end

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

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

Using repetition code was still the best because the BER increases the slowest as the transmission distance increases. From comparing the graph the BER plot using repetition code always has the lowest BER at any given distance.

******



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

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

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