# 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\_raw\_data =  
  
 250000  
  
  
size\_run\_length =  
  
 301688

size\_huffman =  
  
 117374 100981

Run length encoding is not better. There is most likely a lot of lower number needing to be represented, which require more bits in the run length code than in the raw data.

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**.

size\_raw\_data =  
  
 250000  
  
  
size\_run\_length =  
  
 196680

size\_huffman =  
  
 134892 120565

The run length code is now shorter, but the huffman code is longer.  
When transposed, the letters will be lying down, which means that we will alternate less between black and white pixels.

The huffman code is worse now. This must mean that more unique run lengths with longer huffman codes occur less frequently.

***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 % % % %*

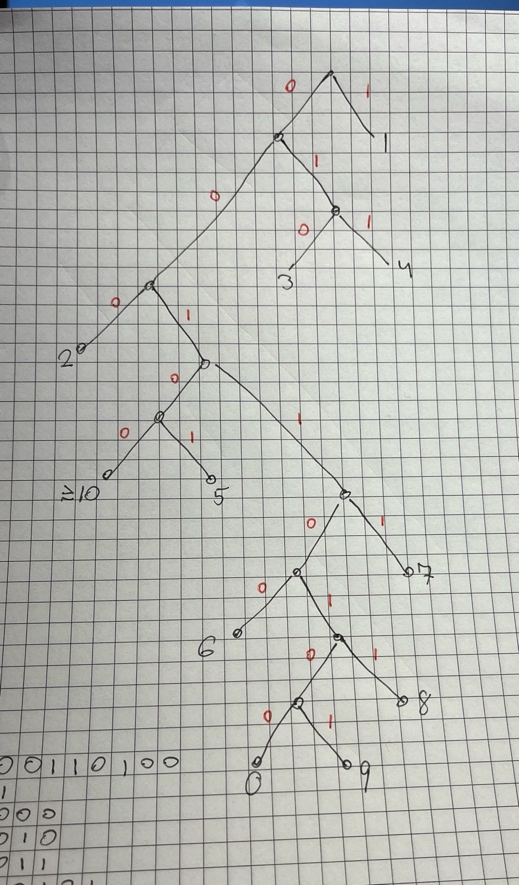
and

*% % % % 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***

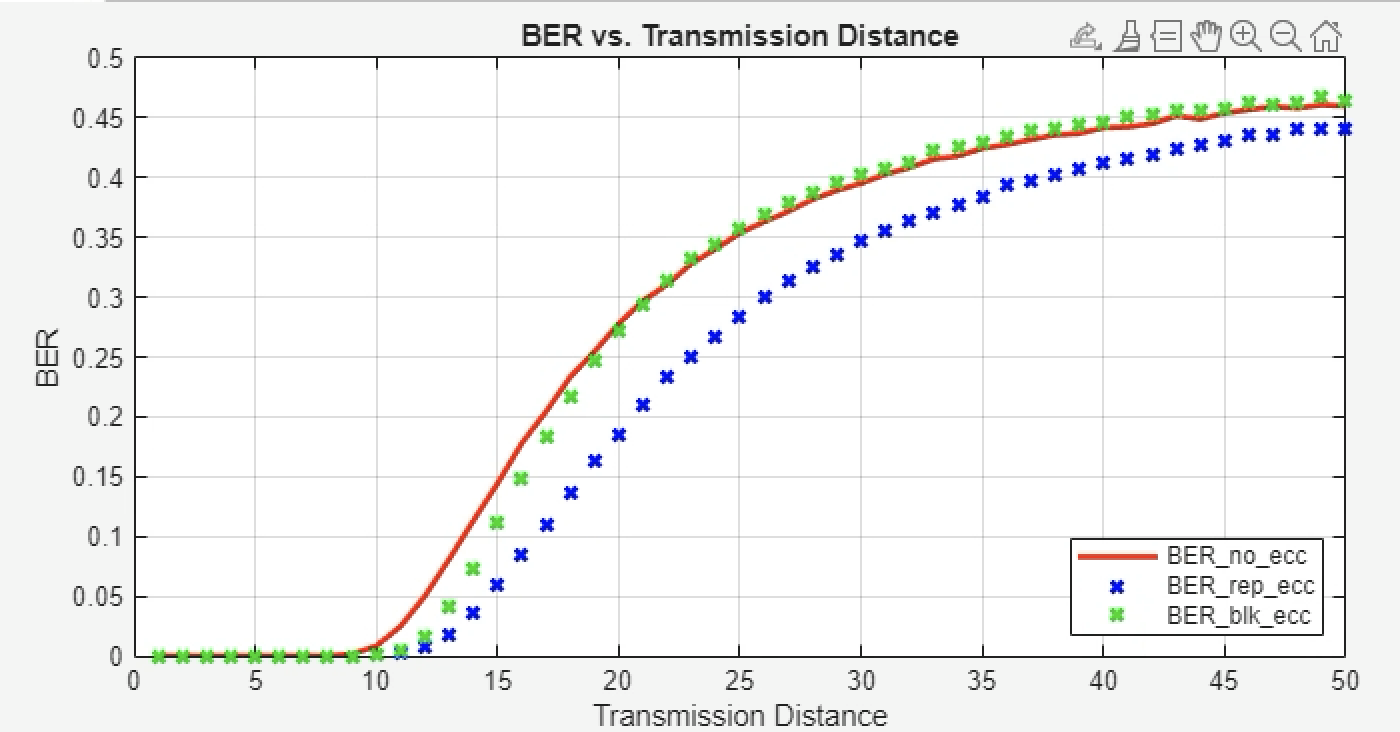


# Task 3 – (n,k) block code decoder and Error Correction Capability

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**”.

if (S(1)==1) && (S(2)==0) && (S(3)==1) && (S(4)==0)

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

elseif (S(1)==1) && (S(2)==0) && (S(3)==0) && (S(4)==1)

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

elseif (S(1)==0) && (S(2)==1) && (S(3)==1) && (S(4)==0)

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

elseif (S(1)==0) && (S(2)==1) && (S(3)==0) && (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**.

The repetition code scheme seems to perform better, no matter the distance. The block code decoder performs better than with no error correction at shorter distances but doesn’t make a difference at longer. This is probably because, it can only correct the error when 1 out of the 8 bits are wrong. Anything more than that (which probably happens more frequently at longer distances), and the algorithm cannot identify the correct bits to correct and will correct wrong ones instead.

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

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

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