COMP 411 Digital Image and Video Processing

Individual Project: Image compression

JPEG is the most fundamental and commonly used image compression standard nowadays. It achieves relatively high compression ratio with acceptable subjective quality of the image. In this project, you are required to write JPEG codec in Scilab/Matlab and answer questions related to the compression performance.

1. Generate an image and add your student ID and your English name on it. Paste the image below and write down its file size. **(10 marks)**

|  |
| --- |
| (Paste the generated image here.) |
| (Write down the file size of the image here.)  480,000 bytes (without tiff header) |

1. Write a **JPEG codec (encoder and decoder)** function in Scilab/Matlab to compress the image you generated in Question 1 and write down the file size. **(50 marks)**

Note: After the compression, the file size of the compressed image should be **less than half of the original file size**.

|  |
| --- |
| (Paste the codes of the JPEG encoder here.)  function [header\_dict, header\_ac\_seperator, header\_resolution, header\_quality, bit\_stream] = jpeg\_encoder(img, quality\_coef)  header\_quality = quality\_coef;  bit\_stream = [];  AC\_lengths = [];  header\_resolution = size(img);  % RGB to YCbCr  Y = rgb2ycbcr(img);  Y\_dsp = Y;  % Downsample Cb and Cr channels  Y\_dsp(:,:,2) = 2 \* round(Y\_dsp(:,:,2)/2);  Y\_dsp(:,:,3) = 2 \* round(Y\_dsp(:,:,3)/2);  Q = [16 11 10 16 24 40 51 61 ;  12 12 14 19 26 28 60 55 ;  14 13 16 24 40 57 69 56 ;  14 17 22 29 51 87 80 62 ;  18 22 37 56 68 109 103 77 ;  24 35 55 64 81 104 113 92 ;  49 64 78 87 103 121 120 101 ;  72 92 95 98 112 100 103 99];  zigzag = [1 2 6 7 15 16 28 29;  3 5 8 14 17 27 30 43;  4 9 13 18 26 31 42 44;  10 12 19 25 32 41 45 54;  11 20 24 33 40 46 53 55;  21 23 34 39 47 52 56 61;  22 35 38 48 51 57 60 62;  36 37 49 50 58 59 63 64];  first = 1;  % seperate image as 8x8 blocks  [row, column, ~] = size(Y\_dsp);  Y\_dct = zeros(size(Y\_dsp));  for channel = 1:3  for j = 1:8:row - 7  for k = 1:8:column - 7  % 2D DCT  Y\_dct(j:j+7, k:k+7, channel) = dct2(Y\_dsp(j:j+7, k:k+7, channel));  % Quantization using table Q  Y\_dct(j:j+7, k:k+7, channel) = round(Y\_dct(j:j+7, k:k+7, channel) ./ (Q \* quality\_coef));  % Zigzag scan and convert to 1D array  Y\_lin = zeros(1, 64);  for i = 1:8  Y\_lin(zigzag(i, :)) = Y\_dct(j+i-1, k:k+7, channel);  end  % Extract DC (first digit diff)  if first == 1  DC\_prev = Y\_lin(1);  DC = Y\_lin(1);  first = 0;  else  DC = Y\_lin(1) - DC\_prev;  DC\_prev = Y\_lin(1);  end  % Extract AC  AC = Y\_lin(2:end);  % RLE encoding  AC = rle(AC, "en");    AC\_lengths = [AC\_lengths, length(AC)];  bit\_string = [DC, AC];  bit\_stream = [bit\_stream, bit\_string];  end  end  end  % Huffman encoding on all [DC + AC]'s  [unique\_symbols, ~, symbol\_indices] = unique(bit\_stream);  symbol\_counts = accumarray(symbol\_indices(:), 1);  symbol\_counts = symbol\_counts / sum(symbol\_counts);  dict = huffmandict(unique\_symbols, symbol\_counts);  bit\_stream = huffmanenco(bit\_stream, dict);  bit\_stream = dec2bin(bit\_stream);  header\_dict = dict;  header\_ac\_seperator = rle(uint8(AC\_lengths), "en");  end |
| (Paste the codes of the JPEG decoder here.)  function img = jpeg\_decoder(header\_dict, header\_ac\_seperator, header\_resolution, quality\_coef, content)  inverse\_zigzag = [1 2 6 7 15 16 28 29;  3 5 8 14 17 27 30 43;  4 9 13 18 26 31 42 44;  10 12 19 25 32 41 45 54;  11 20 24 33 40 46 53 55;  21 23 34 39 47 52 56 61;  22 35 38 48 51 57 60 62;  36 37 49 50 58 59 63 64];  Q = [16 11 10 16 24 40 51 61;  12 12 14 19 26 58 60 55;  14 13 16 24 40 57 69 56;  14 17 22 29 51 87 80 62;  18 22 37 56 68 109 103 77;  24 35 55 64 81 104 113 92;  49 64 78 87 103 121 120 101;  72 92 95 98 112 100 103 99];  bit\_stream = huffmandeco(bin2dec(content), header\_dict);  header\_ac\_seperator = rle(header\_ac\_seperator, "de");  block\_row = header\_resolution(1) / 8;  block\_col = header\_resolution(2) / 8;  img = zeros(header\_resolution);  head = 1;  DC\_prev = bit\_stream(1);  current\_row = 1;  current\_col = 1;  current\_channel = 1;  for i = 1:length(header\_ac\_seperator)  cut = uint32(header\_ac\_seperator(i));  DC = bit\_stream(head);  AC = bit\_stream(head+1:head+cut);  % RLE decoding  AC = rle(AC, "de");  % Calculate DC  if head ~= 1  DC = DC\_prev + DC;  DC\_prev = DC;  end  block = [DC AC];  % inverse zigzag  try  block = block(inverse\_zigzag);  catch  continue  end  % inverse quantization  block = block .\* Q \* quality\_coef;  % inverse DCT  block = idct2(block);  block = uint8(block);  img(current\_row:current\_row+7, current\_col:current\_col+7, current\_channel) = block;  current\_col = current\_col + 8;  if current\_col > block\_col \* 8  current\_col = 1;  current\_row = current\_row + 8;  end  if current\_row > block\_row \* 8  current\_row = 1;  current\_channel = current\_channel + 1;  end  head = head + cut + 1;  end  img = uint8(img);  img = ycbcr2rgb(img);  end |
| (Write down the file size of the compressed image here.)  333,534 bytes (with header) |

1. Generate the difference image using the formula below and thresholding the difference image so that the differences can be seen clearly.

Where and are the original image and compressed image, respectively.

Can you see any visual differences between the compressed image and the original image? Regarding the workflow of JPEG codec, explain why they are different. **(20 marks)**

|  |
| --- |
| (Paste the codes for generating the difference figure here.)  threshold = 3  decoded\_img = jpeg\_decoder(header\_dict, header\_ac\_seperator, header\_resolution, header\_quality, content);  diff = abs(uint8(img - decoded\_img));  diff(diff>threshold) = 255; |
| (Paste the compressed image and the thresholded difference image here.)  Graphical user interface, application  Description automatically generated |
| (Put your answer and explanation here.)  Each 8x8 blocks after DCT and quantization are rounded, so some precise floating point values are truncated, hence; there are slightly difference between original image and compressed image.  % Quantization using table Q  Y\_dct(j:j+7, k:k+7, channel) = round(Y\_dct(j:j+7, k:k+7, channel) ./ (Q \* quality\_coef)); |

1. PSNR (Peak Signal and Noise Ratio) is used to objectively measure the difference between the original image and the compressed image (namely measure the quality of the compressed image). PSNR in dB can be calculated as below,

Where is the maximum possible pixel value of the image. is the Mean Square Error, . *m* and *n* are the number of rows and columns of and .

Write **PSNR measure** function in Scilab/Matlab to calculate the PSNR of the recovered image and paste the PSNR value below. **(20 marks)**

|  |
| --- |
| (Paste the codes of PSNR measure here.)  function value = my\_psnr(original, compressed)  original = double(original);  compressed = double(compressed);  [row, column, channel] = size(original);  if channel == 3  original = rgb2gray(original);  compressed = rgb2gray(compressed);  end  mse = sum(sum((original - compressed).^2))/(row\*column);  value = 10\*log10(255^2/mse);  end |
| (Paste the PSNR value here.)  PSNR value (dB): 67.0936 |