Blake Johnson

9663980

Aditya 12pm

Homework 8

Matlab code with parts a-g labelled:

%Load in the matlab images so that they are accessible to the workspace

in01 = load('IMG\_7401.mat');

in05 = load('IMG\_7405.mat');

orig1 = in01.I;

orig5 = in05.I;

%One argument sanity check.

%Calls myJpeg to make sure that function can be called with only the image

%as an argument, with QF defaulting to 5. IT DOES WORK!

[J0, C0, rms0] = myJpeg(orig1);

figure

imshow(J0);

disp("Compression Factor 0");

disp(C0);

disp("RMSE 0");

disp(rms0);

%TESTING BEGIN

%-------------

%Calls myJpeg on the first image (IMG\_7401)

[J1, C1, rms1] = myJpeg(orig1, 1);

[J2, C2, rms2] = myJpeg(orig1, 5);

[J3, C3, rms3] = myJpeg(orig1, 10);

%Calls myJpeg on the second image (IMG\_7405)

[J4, C4, rms4] = myJpeg(orig5, 1);

[J5, C5, rms5] = myJpeg(orig5, 5);

[J6, C6, rms6] = myJpeg(orig5, 10);

%Display the first image with QF = 1, 5, 10

%QF = 1

%image = IMG\_7401

figure

imshow(J1);

disp("Compression Factor 1");

disp(C1);

disp("RMSE 1");

disp(rms1);

%QF = 5

%image = IMG\_7401

figure

imshow(J2);

disp("Compression Factor 2");

disp(C2);

disp("RMSE 2");

disp(rms2);

%QF = 10

%image = IMG\_7401

figure

imshow(J3);

disp("Compression Factor 3");

disp(C3);

disp("RMSE 3");

disp(rms3);

%Display the second image with QF = 1, 5, 10

%QF = 1

%image = IMG\_7405

figure

imshow(J4);

disp("Compression Factor 4");

disp(C4);

disp("RMSE 4");

disp(rms4);

%QF = 5

%image = IMG\_7405

figure

imshow(J5);

disp("Compression Factor 5");

disp(C5);

disp("RMSE 5");

disp(rms5);

%QF = 10

%image = IMG\_7405

figure

imshow(J6);

disp("Compression Factor 6");

disp(C6);

disp("RMSE 6");

disp(rms6);

%PART A

%Pads the image to be a factor of 8x8.

function J = padImage(image)

[M, N] = size(image);

%Find modulo of M and N to determine if rows and columns are divisible

%by 8

rowMod = mod(M,8);

colMod = mod(N,8);

%if M (rows) is not divisible by 8

if(rowMod ~= 0)

%adds (8 - remainder) to M. If remainder when dividing by 8 was 3

%for example, this would add 5 to M, making it divisible by 8.

newM = M + 8 - rowMod;

%M is divisible by 8

else

newM = M;

end

%if N (rows) is not divisible by 8

if(colMod ~= 0)

%adds (8 - remainder) to N. If remainder when dividing by 8 was 3

%for example, this would add 5 to N, making it divisible by 8.

newN = N + 8 - rowMod;

%M is divisible by 8

else

newN = N;

end

%if padding was necessary for either M or N

if(newM ~= M || newN ~= N)

%create a new matrix of zeros with padded dimensions

output = zeros(newM, newN);

%Copy the corresponding indexes from the original image

for m=1:M

for n=1:N

output(m,n) = image(m,n);

end

end

%Output padded image

J=output;

else

%Output unmodified image (no padding necessary)

J=image;

end

end

%PART B

%Level shifts the image by the mean (128)

function J= levelShift(image)

J = padImage(double(image))-128;

end

%PART C and D

%Perform DCT and threshold coding using quantization matrix Q, on 8x8 blocks

function J = quantization(image, QF)

%Make sure image has been level shifted.

im = levelShift(image);

%Declare/initialize quantization matrix from slides

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];

%Check quality factor, and implement changes on Q, if any.

if(QF > 5)

%Downscales image by multiplying Q matrix. This is scaled by the

%taking the distance from 5 (our default value for Q), dividing by

%two so it doesnt scale too crazy, and adding one (to differentiate

%from 5 in the case of 4 and 6, and then squaring the result.

Q = Q\*(((QF-4)/2)+1)^2;

elseif(QF < 5)

%Upscales image by dividing Q matrix. This is scaled by the

%taking the distance from 5 (our default value for Q), dividing by

%two so it doesnt scale to crazy, and adding one (to differentiate

%from 5 in the case of 4 and 6, and then squaring the result.

Q = Q/(((6-QF)/2)+1)^2;

end

%Performs block process on our image, calculating the dct of the 8 by 8

%blocks and then divides the element by the corresponding element in Q.

dct = @(block\_struct) dct2(block\_struct.data)./Q;

J = blockproc(im,[8 8],dct);

end

%PART E

function J = compressionFactor(image)

imageRound = round(image);

[M,N] = size(imageRound);

counter = 0;

totalPixels = M\*N;

%Counts the number of pixels in the image withvalue 0

for m=1:M

for n=1:N

if(imageRound(m,n) == 0)

counter = counter + 1;

end

end

end

%Returns ratio of zeros to total pixels.

J = (counter/totalPixels);

end

%PART F

%Perform inverse process from part C and D, and shift the level back.

function J = reconstruct(image, QF)

%Declare/initialize quantization matrix from slides

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];

%Check quality factor, and implement changes on Q, if any.

if(QF > 5)

%Downscales image by multiplying Q matrix. This is scaled by the

%taking the distance from 5 (our default value for Q), dividing by

%two so it doesnt scale too crazy, and adding one (to differentiate

%from 5 in the case of 4 and 6, and then squaring the result.

Q = Q\*(((QF-4)/2)+1)^2;

elseif(QF < 5)

%Upscales image by dividing Q matrix. This is scaled by the

%taking the distance from 5 (our default value for Q), dividing by

%two so it doesnt scale to crazy, and adding one (to differentiate

%from 5 in the case of 4 and 6, and then squaring the result.

Q = Q/(((6-QF)/2)+1)^2;

end

IDCT = Q;

%Performs a block process on our image, calculating the inverse dct of the 8 by 8

%blocks and then multiplies the element by the corresponding element in Q.

invdct = @(block\_struct) idct2(block\_struct.data.\*IDCT);

recon = blockproc(image,[8 8],invdct);

%Shift the image back by the mean value, and return.

J=uint8(recon+128);

end

%PART G

%Calculates root mean square error, same as in HW7

function X = RMSE(original, reconstruct)

[M, N] = size(original);

sum = 0;

for m=1:M

for n=1:N

sum = sum + (double(original(m,n)) - double(reconstruct(m,n)))^2;

end

end

X = sqrt(sum/(M\*N));

end

function [J, C, rms] = myJpeg(image, varargin)

%Checks to see if only one argument was passed.

if(nargin == 1)

%Calls myJpeg with default QF.

[J, C, rms] = myJpeg(image, 5);

%Checks to see if both the image and the quality factor were passed

%into the function

elseif(nargin == 2)

%Performs jpeg calculations (quantize and reconstruct)

quantize = quantization(image, varargin{1});

recon = reconstruct(round(quantize), varargin{1});

J = recon;

%Calculates compression factor

C = compressionFactor(quantize);

%Calculates root mean square error.

rms = RMSE(image, recon);

end

end

IMG\_7401

QF = 1



QF = 5



QF = 10



Table for IMG\_7401

|  |  |  |  |
| --- | --- | --- | --- |
| Quality Factor (QF) | 1 | 5 | 10 |
| Compression Factor | 0.7053 | 0.8818 | 0.9844 |
| RMS Error | 0.5822 | 4.5055 | 15.7626 |

IMG\_7405

QF = 1



QF = 5



QF = 10

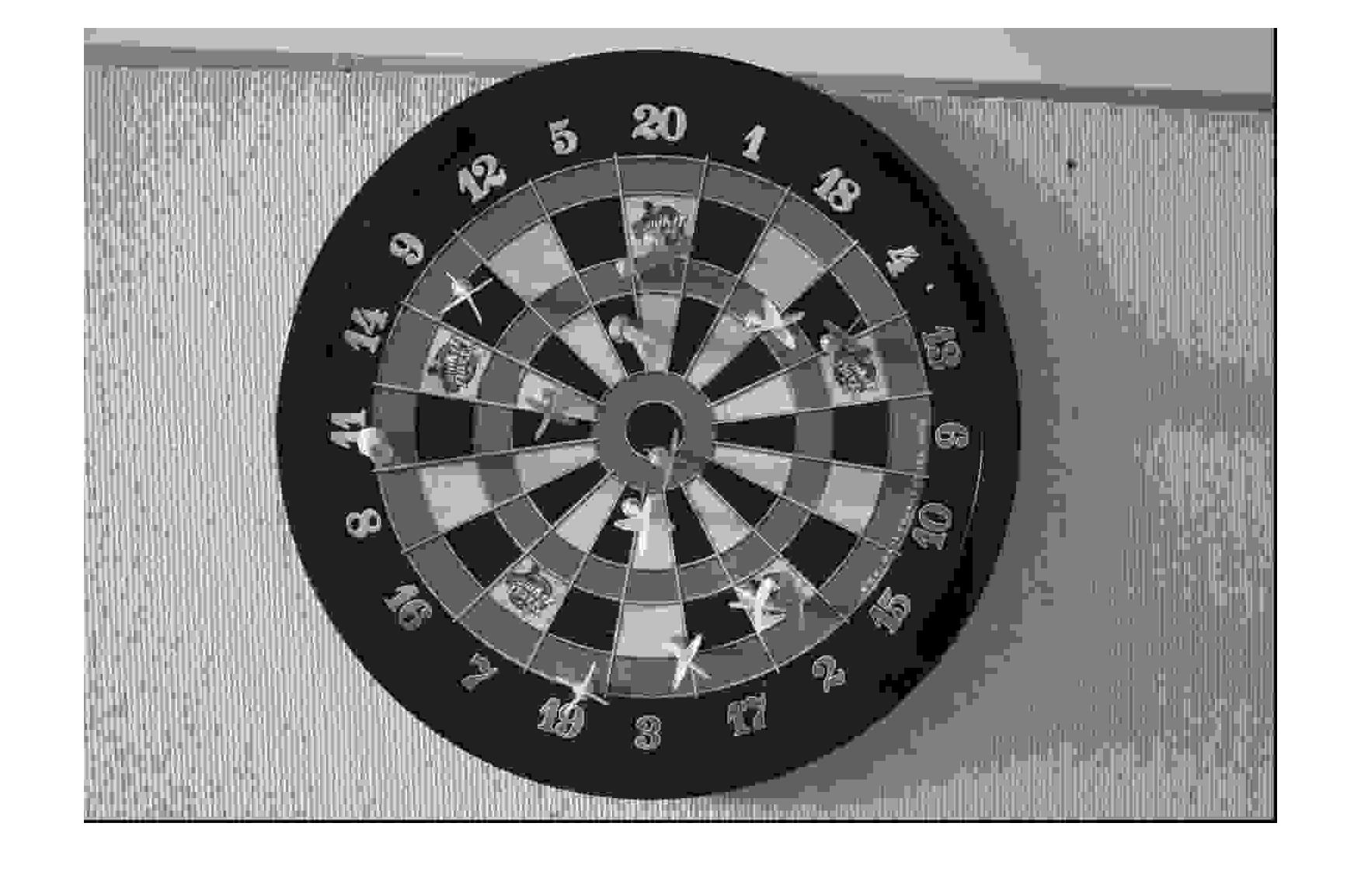


Table for IMG\_7405

|  |  |  |  |
| --- | --- | --- | --- |
| Quality Factor (QF) | 1 | 5 | 10 |
| Compression Factor | 0.6311 | 0.8435 | 0.9763 |
| RMS Error | 0.6815 | 5.3388 | 17.2115 |