1 Introduction

1.1 Overview

A transform coder consists of three distinct parts: The transform, the quantizer and the entropy coder. In this laboration you will study all three parts and see how the choice of transform/quantizer/entropy coder affects the performance of the transform coder.

The laboration runs in a MATLAB environment. In MATLAB grey-scale images are naturally represented as matrices. During the laboration certain test images will be compressed. The test images can be thought of as original images in the sense that they are stored as raw samples with 8 bits/pixel. This usually gives more grey-levels than the human eye can discern on a computer screen.

1.2 Preparations

We assume that you have studied the chapters on transform coding, quantization and entropy coding in the course literature. Read through this paper before the laboration and see if there are any questions you can answer beforehand. We also suggest that you read the manual (help <funcname>) for each new function before you use it.
2 Simple Data Reduction

We will start by looking at two simple data reduction methods: Reducing the number of grey-levels in the image, and down-sampling of the image. To reduce the number of grey-levels from 256 to 128 (i.e. going from 8 bits per pixel to 7 bits per pixel) do the following:

```matlab
>> l = pgmload('lenna.512.pgm');
>> imshow(l,[0 255])
>> l2 = 2*floor(l/2);
>> figure,imshow(l2,[0 255])
```

Q.1 What is the lowest number of bits per pixel that gives an acceptable image quality?

Down-sampling can be done by using the function `imresize`. To down-sample (and then up-sample) the image by a factor 2 in each dimension, do:

```matlab
>> l3 = floor(imresize(l, 0.5, 'bicubic'));
>> l4 = imresize(l3, 2, 'bicubic');
>> figure,imshow(l4,[0 255])
```

Since this down-sampling reduces the number of pixels by a factor of 4, we have essentially gone from 8 bits per pixels to 2 bits per pixel.

Q.2 What is the lowest number of bits per pixel that gives an acceptable image quality?

Q.3 Comparing grey-level reduction and sub-sampling at the same bit-rate, which method gives the best result?

3 Image Transformations

In image coding, the whole image is not transformed with a single transform. Rather, the image is divided into small blocks (typically 8x8 pixels) that are transformed
separately.

Q.4 Give a few reasons why block-based transform methods are preferable in image coding.

Q.5 Name a few criteria for choosing a good transform for image coding.

The transforms that we will use in this laboration (Hadamard and cosine transforms) are separable transforms, i.e. the two-dimensional transform is composed of two one-dimensional transforms, one applied in the vertical direction and one in the horizontal direction. Usually, the transform components are arranged in a block of the same size as the image block. The components are sorted in increasing frequency order, with the DC component in the upper left corner. The transform functions in this lab (bdct and bht) arrange the components of the transformed block into column vectors instead, to make it easier to do quantization and entropy coding.

To view the basis functions of the 8x8 DCT, do:

```matlab
>> dbasis = ibdct(eye(64), [8 8], [64 64]);
>> imshow(dbasis, [])
>> imgrid([8 8])
```

You might need to enlarge the window to see clearly. Compare to figure 12.4 in Sayood’s book. Have a look at the Hadamard basis functions too, by replacing ibdct with ibht.

Take a look at the histograms (corresponding to the probability density functions) of the original image and the transformed image.

```matlab
>> ihist(l(:))
>> ldct = bdct(l, [8 8]);
>> ihist(ldct(:))
```

Q.6 Which image (original or transformed) should be easier to code, and why?
4 Quantization

We are now ready to perform quantization of our transform data. We will only be looking at uniform quantization, using the function `bquant`. Quantize all transform components with the same quantization step:

```matlab
>> Q1=50;
>> ldctq = bquant(ldct, Q1);
>> ldctr = brec(ldctq, Q1);
>> lr = ibdct(ldctr, [8 8], [512 512]);
>> figure, imshow(lr, [0 255]);
```

Now you have the quantized transform `ldctq`, the reconstructed transform `ldctr` and the reconstructed image `lr`. Compute the distortion between the original image and the reconstructed image:

```matlab
>> dist = mean((I(:)-lr(:)).^2)
```

In the same way, compute the distortion between the original transform and the reconstructed transform and compare it to the image distortion.

Q.7 What is the relationship between the image distortion and the transform distortion, and why?

Usually, distortion in images is measured by the PSNR (Peak-to-peak signal to noise ratio). For an 8 bits/pixel grey-scale image, it is given by

```matlab
>> psnr = 10*log10(255^2/dist)
```

Vary the quantization step `Q1` and look at the reconstructed images.

We can of course use different quantization steps for different transform components. The function `jpgqmtx` returns the suggested quantization vector for the JPEG standard. To see the different steps, do

```matlab
>> reshape(jpgqmtx, 8, 8)
```

The quantization step for the DC component is in the top left corner.

By multiplying this quantization vector with different constants, you can vary the quantization, for example:

```matlab
>> Q2 = 3;
```
Vary the quantization parameters $Q_1$ and/or $Q_2$ so that you get the same distortion for both methods. Try this for a couple of different distortions.

\[Q.8\] Which method gives the best looking images?

5 \hspace{1em} \textbf{Entropy Coding}

We are going to study two entropy coding methods: Simple Huffman coding and JPEG style run-length coding.

In order to do Huffman coding we first need to calculate the histogram of the data we want to code. Assuming that we want to use a single Huffman code for all quantized transform components, we can do:

\[
\begin{align*}
\text{>> } p &= \text{ihist(ldctq(:));} \\
\text{>> huffman(p)}
\end{align*}
\]

This returns the number of bits required to code the quantized data. Note that the \texttt{huffman} function doesn't include the cost to code the huffman tree, and thus the real number of bits needed would be slightly higher.

If we instead want to have a separate huffman code for each transform component we do:

\[
\begin{align*}
\text{>> } p &= \text{ihist(ldctq');} \\
\text{>> sum(huffman(p))}
\end{align*}
\]

If we have a large number of different huffman trees, the cost to describe them all will be significant.

A more advanced entropy coding method is the one used in the JPEG standard coding method. The quantized transform components are rearranged in zig-zag scan order, and all consecutive zeros are run-length encoded. Finally, the (run-length, non-zero coefficient) pairs are huffman coded. Here you can do this with the function \texttt{jpgrate}:
>> rates = jpgrate(ldctq, [8 8]);
>> sum(rates)

Q.9 Why does the JPEG style coding perform better than huffman coding?

By using the ratevis function you can display where in the image most of the bits are spent

>> ratevis(rates, [8 8], [512 512])

6 Own experiments

Now that you know how to use all three parts (transform, quantizer, entropy coder) it is time for you to experiment freely. Choose a transform method (Hadamard or DCT), a block size, a quantization method and an entropy coding method. Vary the coarseness of the quantization and measure data rate and SNR. Plot a curve in the diagram below. Repeat with different sets of transform, quantization and coding methods and try to establish which coder configuration that is the best.
7 MATLAB functions

Here is a list of all the special MATLAB functions you will use in the laboration:

<table>
<thead>
<tr>
<th>Function</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bht</td>
<td>Block based Hadamard transform</td>
</tr>
<tr>
<td>ibht</td>
<td>Inverse block based Hadamard transform</td>
</tr>
<tr>
<td>bdct</td>
<td>Block based Discrete Cosine Transform</td>
</tr>
<tr>
<td>ibdct</td>
<td>Inverse block based DCT</td>
</tr>
<tr>
<td>bquant</td>
<td>Block uniform quantizer</td>
</tr>
<tr>
<td>brec</td>
<td>Block uniform reconstruction (inverse quantization)</td>
</tr>
<tr>
<td>jpgqmtx</td>
<td>JPEG quantization vector</td>
</tr>
<tr>
<td>ihist</td>
<td>Histogram calculation</td>
</tr>
<tr>
<td>huffman</td>
<td>Huffman encoding</td>
</tr>
<tr>
<td>jpgrate</td>
<td>JPEG style run-length encoding</td>
</tr>
<tr>
<td>ratevis</td>
<td>Rate visualization</td>
</tr>
</tbody>
</table>
pgmload  Read image from a PGM file
imshow   Display an image