Multimedia
Module No: CM3106
Laboratory Worksheet Lab 7 (Week 8): MATLAB Discrete Cosine Transform and JPEG Compression

Dr. Kirill Sidorov

Aims and Objectives

After working through this worksheet you should be familiar with:

- Using transforms to decorrelate pixel intensities.
- Discrete cosine transform.
- JPEG compression process.
- The basic use of MATLAB to investigate the above.

None of the work here is part of the assessed coursework for this module

MATLAB DCT and JPEG Compression

1. **Preliminaries**.

   (a) Download the JPEG Image Compression MATLAB Zipped Files from the CM0340 web pages. Uncompress and install in an appropriate MATLAB accessible directory.

2. **Basic Transform Coding Example**.

   (a) Open and examine `simpletrans.m`. This[1] implements the basic example we considered in the lectures: taking advantage of correlation between the colours of adjacent pixels.

   The `simpletransquant.m` demo additionally uses quantisation.

   - Run these demos and explain the output.
   - Plot the histograms (using `hist` command) for the original pixel colours, and for the differences. What do the observed histograms tell you about coding efficiency?
   - Try using different quantisation constants and note the compression ratios vs image quality.

3. **Discrete Cosine Transform**

   (a) Open `DCT1Deg.m` which performs 1D DCT using the build in MATLAB function `dct`, as well as by multiplying the signal by the matrix of DCT coefficients (produced by `dctmtx`). Hence convince yourself that DCT is nothing more than a linear transform. Try it on other signals. Carry out the inverse DCT (using built-in `idct` as well by multiplying by the inverse of the DCT coefficients matrix; in the process convince yourself that, since it is orthogonal, `D' = inv(D)`). Make a larger DCT matrix (*e.g.* `dctmtx(64)`), display it as an image, and plot some of its rows (columns) to get some intuition for the DCT basis functions.

   (b) Investigate how well DCT approximates lines. The program `dctlines.m` illustrates this. Try changing the number of preserved coefficients and note the difference. Why is this relevant to image compression?

   (c) Get some intuition for 2D DCT. Examine `dct2basis.m` and see how the 2D basis images are constructed out of 1D basis vectors. Convince yourself that 2D DCT is still merely a linear transform: open and examine `dct2manual.m` which vectorises the 2D basis matrices (as well as the signal) and hence performs 2D DCT as a simple matrix multiplication. Note that MATLAB also has the built-in commands `dct2` and `idct2` for 2D DCT.

[1]This exercise was also one of the last in the previous lab. If you have done it last week already, skip it, obviously.

(d) Investigate the energy compaction property of DCT. The file `dctenergy.m` illustrates this. Observe, for various images, the distribution of energy in its 2D DCT transform. Where is it the highest? What frequencies does it correspond to? How is it useful for image compression? Try truncating the DCT coefficients (leaving just the top left corner, setting the rest to zero) and reconstructing the image. What effect does it have on the reconstructed image?

(e) Investigate the Gibbs phenomenon. Open `gibbs1.m` which approximates a step function with cosine harmonics. Alter the number of preserved harmonics (variable `f`) and note how their linear combination approximates the step function, especially paying attention to the overshoot around the discontinuity. Compute the squared difference between the original and the reconstructed signal as the function of the number of preserved harmonics. Explain how is this relevant to image compression? The file `gibbs2.m` allows you to observe the same phenomenon in the 2D case. *Optional:* besides increasing the number of harmonics, how would you combat this effect?

(f) Play with `dctdemo.m` to examine how well an image can be approximated with different number of DCT coefficients.

4. **JPEG Compression**

- The JPEG examples are found in the `jpeg` directory.
- The directory should contain these files:
  - `im2jpeg.m` — JPEG encoder
  - `jpeg2im.m` — JPEG decode
  - `mat2huff.m` — Huffman coder
  - `huff2mat.m` — Huffman decoder
- You may need to compile (using `mex`) the `unravel.c` for your platform.
- Examine the above code. In `im2jpeg.m` pay particular attention to the variable `m`, `order` and functions `mat2huff.m`, `dctmtx`. Use MATLAB help to find out more about `blkproc` and `im2col`.
- Load in an image in MATLAB and encode it using `im2jpeg.m`. Note the structure used and the formats used.
- Decode the above and store in a different image variable.
- Compare the the two images by the following:
  - Visually: display both images in MATLAB.
  - Compute the difference between the images and display this result.
  - Compute the squared error between the images and display this result.
- Change the quantisation matrix `m` to different values and compare results as above.
  - See also the JAVA Applet demo for values for `m`.
  - Implement a constant value quantisation by *2,4,6, 8, 32, or 128*.
  - Invent some values for `m` yourself — for JPEG to work according to specification `m` should implement a *low pass filter* but you could try other values?
- Why does it make sense to compress images in small blocks? Why do not we simply apply DCT to the entire image instead? Investigate whether in JPEG pipeline compressing images in blocks is indeed more economical: modify the provided JPEG code to operate on the entire image at once and compare the results.