Advanced Communication Systems, ELEN90051

Workshop week 4: Transform coding in image and video

Department of Electrical and Electronic Engineering
The University of Melbourne

Created by Mike Biggar (updated 13 March 2018)

1. Introduction

In the workshop of week 4, you will focus on the use of DCT-based transform coding of images and video, in particular utilising JPEG coding and MATLAB. The objectives of this workshop are to:

- Reinforce the material covered in lectures
- Allow you to gain experience in use of the JPEG image coder
- Establish an understanding between the different image file formats, especially between lossless and lossy coded formats
- Allow some insight into techniques used to reduce the bitrate to represent video, as well as still images.

2. Organisation

You are expected to **be prepared before attending the workshop sessions**. For this, **an individual pre-workshop report answering the questions in Section 4 below is to be submitted via the LMS.** See "Rules on Workshops and Report Submission" on the LMS for details about the report naming convention and submission deadlines. The pre-workshop report will be worth 5 marks. Keep a copy of your individual pre-workshop report, as it will help you complete the rest of the workshop. You may also wish to do a little research in advance to help you answer **Section 8**.

During the workshop, you will be **individually marked** by your demonstrator for preparedness, quality and initiative (worth 5 marks).

You will be randomly organised in groups of 2 students. Your group will generally be different from previous workshops.

A group project report (worth 90 marks), answering all questions of sections 5-8 and summarising any other insights you obtain, **is due by the start of the next workshop (week 5)**. Again, refer to "Rules on Workshops and Report Submission" for submission details.

Tip: In this workshop, you will create several image files of different subjects and at different compressions. You should take care to give these files sensible filenames so you can easily refer to and compare them when you are answering the later questions.

3. Image Copyright

All of the pictures you will use in this lab are owned by Mike Biggar, except "Obama_640.bmp" and "Stinson_640.bmp" which are cropped and scaled versions of http://upload.wikimedia.org/wikipedia/commons/e/e9/Official_portrait_of_Barack_Obama.jpg and http://www.carolinestinson.com/Images/Portraits/Stinson_ARH6635.jpg respectively. Please delete

all versions and copies of the pictures once you have completed your lab class (but ONLY once you're CERTAIN you've finished!).

4. Pre-workshop questions (5 marks)

a) Describe in a couple of sentences how (colour) image data is stored in a bmp file. Is the image compressed or uncompressed? How many bits are used to represent each pixel?
b) Describe briefly in your own words how the JPEG image coder/decoder works. Is JPEG compression lossy or lossless? Explain the difference.
c) What would you expect a heavily compressed JPEG image to look like? Explain why.
d) JPEG exploits redundancy within a picture. What sort of redundancy occurs between pictures in a video? How do you think that redundancy could be exploited to reduce the number of bits to be transmitted?

5. How JPEG works (20 marks)

a) Download the test image "KIsealion_640.bmp" from the LMS. How big is the file? How many bits are used to store each pixel? Set the current directory in MATLAB to be the same directory as the one in which you stored the image.
b) Use the *imread* operation to load the image into an array in MATLAB:

```
>> i_array = imread('KIsealion_640.bmp', 'bmp');
```

c) Use the *size* function to obtain the dimensions of the array. Explain the numbers you obtain.
d) Now save the image to JPEG form using different "quality" settings in *imwrite*. The quality parameter can be anything from 1 to 100, so to save at highest quality you would use a command like:

```
>> imwrite(i_array, 'KIsealion_Q100.jpg', 'jpg', 'quality', 100);
```

e) Repeat this for some different quality settings, then open the output image files (outside of MATLAB; just double-click them) and look at the resulting images. What do you see? Compare the file sizes with the quality settings you used to create them and the visual quality of the resulting images. What quality setting[1] is the smallest necessary to provide what you would describe as a "good" or "acceptable" quality image? What quality setting gives distortion which is visible, but not too extreme?
f) Take a subset of the (original) image, and plot a 1-D horizontal slice through it for one of the 3 colour components (say, "R")[2].

```
>> figure
>> plot(i_array(150,1:640,1), 'r');
```

g) Repeat this for the corresponding pixels in the most highly compressed versions of the test image. (That is, load the compressed files back into MATLAB in new arrays, and

[1] Do not try to be accurate to 1 in 100 when you try this. Restrict your choice of quality settings to numbers that are factors of 5 (10, 15, 20, 25, etc…) or even 10 (10, 20, 30,…).

[2] The numbers chosen here give a slice through the image that includes the contrasting whiskers on the sea lion. If you wish to check where the slice is, write a fixed pixel value (such as 220,0,0 for a red line) into the same locations in a version of the image copied to a temporary array or file, and view the result.

make the same plot in a different colour. You'll need to use the "*hold on*" command to keep an earlier plot on the same figure.)

How do you explain the results you obtain, in terms of what you know about how JPEG compression works? You may wish to change the location of the slice (i.e. other than line 150) and the number of samples plotted (i.e. less than 640) to help you see what is going on.

h) Plot the other two colour components ("G" and "B") along with "R" on the one graph.

Can you relate the different levels of each colour to the appearance of the picture at the point where you've taken a slice through it? Do you see the same compression process carried out on each colour component?

6. Dependence on images (20 marks)

a) Choose a level of compression you found in 5e above, for which the image suffered some visible distortion, but the distortion was not too extreme.

What value of "quality" did you use? What was the file size? How many bits per pixel (that is, file size divided by number of pixels) were required to represent the picture? How does this compare to the original?

b) Now repeat the process of loading a picture into MATLAB then writing it out at different compression rates (different values of "quality"), but use some of the additional test pictures: "Aerobatics_640.bmp", "Bowlcrowd_640.bmp", "Perth_640.bmp", "Heronwreck_640.bmp", "Obama_640.bmp" and "Stinson_640.bmp". (Use your experience with the first image to be selective in the number of "quality" values you try for each of these additional ones; don't try to do an exhaustive series of tests for every image. You also do not need to use all the above images if you think you understand what is going on and can answer the questions using just some of them. Include at least "Perth_640" and "Stinson_640", though.)

For the same values of "quality", do you think the picture quality (or visible coding distortion) is the same for different pictures? How does it differ between pictures? Do you think you could save some pictures with a smaller number of bits per pixel (bpp) than others, without it appearing significantly worse? What characteristics of pictures make them "hard" to compress (that is, require more bits for a given quality), and what characteristics make them easier to compress? Can you relate this back to what you know about how JPEG works?

7. Interframe video coding (30 marks)

a) Test images "MonashRd1_640.bmp" and "MonashRd2_640.bmp" are two pictures of the same (moving) scene taken a very short time apart. (Think of them as being two consecutive frames of a movie.) Load these images into arrays in MATLAB. You will see that the resultant arrays are 2 (instead of 3) dimensional. Have a look at the images to see why.

b) Take the difference between these images. We need to convert the image array data to type int16 to avoid problems with negative numbers in uint8.

```
>> f1int8_array = imread('MonashRd1_640.bmp', 'bmp');
>> f2int8_array = imread('MonashRd2_640.bmp', 'bmp');
```

```
>> int16_array1=int16(f1int8_array);
>> int16_array2=int16(f2int8_array);
>> int16_diff = int16_array2 - int16_array1 + 128;
```

In the last line of the above sample MATLAB code, we add an offset of 128. Why do you think we do this?

c) Convert the difference array back to an 8 bit array and export it as an image:

```
>> uint8_diff = uint8(int16_diff);
>> imwrite(uint8_diff,'DiffImage.bmp','bmp');
```

d) Open the file and have a look at it on the computer screen. What do you see, and can you explain it?

e) Now try compressing the offset difference array (uint8_diff in the above example) using JPEG coding, and adding the compressed picture back to the first image (int16_array1 in the above example code), before outputting it as a bitmap file and viewing it. (Hint: be careful to manage data types, and don't forget to remove the offset (128) when you combine the pictures back together.) You will find that you need a high quality setting with JPEG to get a reasonable reconstruction[3], and you will see artefacts in the image even with quality=100.

Although we are not focused on coding efficiency here (for the reasons discussed in the footnote), does the process you have carried out suggest a means of saving bits when encoding a sequence of similar video frames?

Some objects in images "MonashRd1" and "MonashRd2" are almost identical. However, because they have moved, we end up with some large difference signals that need to be encoded. This seems a bit wasteful, because we already have information about these objects in the earlier picture. (The objects are there, but in a different place to the second frame.) How could we describe the changes between our consecutive pictures, in a way that might mean we need fewer bits to describe the second one?

8. **Applications** (20 marks)

Though in a more sophisticated form, interframe coding such as you have been exploring in part 7 gives significant bitrate savings in many situations. For which of the following applications do you think it would be suitable, and why?

a) Very high quality applications such as Digital Cinema.
b) Video editing (where video sequences are spliced and joined).
c) Videoconferencing
d) Television broadcasting

– END –

[3] There are several reasons for this. JPEG is optimised for coding natural images with pixel values between 0 and 255. We are trying to use it to code a picture with mostly small positive and negative numbers around zero (which is why we added the offset of 128). In an encoder intended for inter-frame difference encoding, many blocks would have contents close to zero and simply would not be encoded (whereas we have forced JPEG to code a level of 128). Also, the quantisation matrices would be chosen to better suit the statistics of transform coefficients from a difference image. In our example, we have also limited the difference signal range to 256 levels, whereas they could be larger than this (in fact, this is the reason you see a "ghost" image of part of the taxi in the final reconstruction, even when you use Q=100 when encoding the difference image).