ECE462 Multimedia Systems

Laboratory Assignment 1: Colour Image Processing

Winter 2024

TA: Behrad TaghiBeyglou

Procedure - Week 1

1 Images in MATLAB

1. Create a black image that is 10×10 pixels in size. The black colour should be made using RGB intensities (0,0,0).
   Hint: This step simply involves the creation of a three dimensional matrix of appropriate size and filling it with zeroes. The first two dimensions correspond to the spatial location of the pixels and the third dimension is used to specify the colour channel. You may find the MATLAB command `zeros()` useful here.

2. Using the black image from the previous step, create an image as shown in Figure 1. This should be done without using any `for` loops.
   Hint: Remember the ":" operator? Use this operator to address the range and step of pixels in question.

   $\boxed{\text{2 Marks}}$ **Display the image on screen and show the TA**

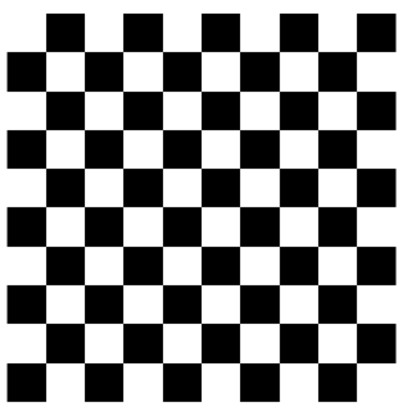


Figure 1: Image to be generated

(Remember, since you are creating pixel values as integers greater than one, it is implied that your image should be considered unsigned integers. As such, you should convert to `uint8` before displaying — use something like `image(uint8(x))` to display image `x`.)

3. Then, set the black squares within the central region of size 4×4 to the red colour and create the image in Figure 2. Again, this should be done without using any `for` loops.

   $\boxed{\text{2 Marks}}$ **Display the image on screen and show the TA**

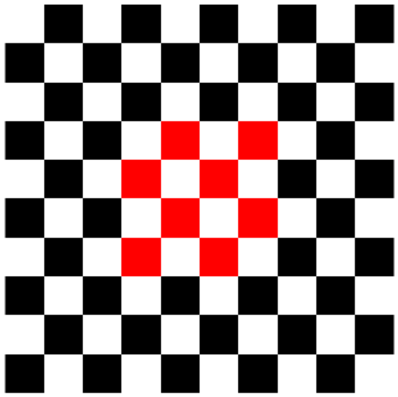


Figure 2: Image to be generated

4. Create a random RGB image of size 256×256 using `rand()`. Make sure each colour intensity is in the interval `[0,255]`. Using MATLAB's function `mean()`, find the mean intensity for all the R, G, and B components separately. In addition, for each of the colour components, use MATLAB's function `find()` to locate all pixels whose intensity value exceeds the mean. Finally, replace these pixels' intensity with the corresponding mean. Combine the three components to generate a new colour image.

   $\boxed{\text{3 Marks}}$ **Display all images on screen and show the TA**

2 Colour Image Transforms

RGB is not the only colour model in use. For example, JPEG images are almost always stored using a three-component colour space called $\mathrm{YC_bC_r}$. The Y, or luminance, component represents the intensity of the image. $\mathrm{C_b}$ and $\mathrm{C_r}$ are the chrominance components, with $\mathrm{C_b}$ specifying the "blueness" of the image and $\mathrm{C_r}$ giving the "redness". Unlike the RGB colour space, where all components are roughly equal in importance, $\mathrm{YC_bC_r}$ concentrates the most important information in one component. This makes it possible to get greater compression by including more data from Y component than from $\mathrm{C_b}$ and $\mathrm{C_r}$. One possible relation between the RGB and YC_bC_r models is represented in the following equations:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \frac{1}{256} \begin{bmatrix} 75 & 150 & 29 \\ -44 & -87 & 131 \\ 130 & -110 & -21 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}, \tag{1}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0133 & -0.0025 & 1.3838 \\ 1.0051 & -0.3336 & -0.6928 \\ 1.0079 & 1.7318 & 0.0047 \end{bmatrix} \begin{bmatrix} Y \\ C_b - 128 \\ C_r - 128 \end{bmatrix}, \tag{2}$$

1. Write a function for converting RGB images to the YC_bC_r colour space. Use the following template:

```
function ycc_image = rgb2ycc(rgb_image)

    rgb_image = double(rgb_image);

    R= _____________________________;

    G= _____________________________;

    B= _____________________________;

    Y = ((75*R + 150*G + 29*B)/256);

    Cb = ((-44*R - 87*G + 131*B)/256) + 128;

    Cr = ((130*R - 110*G - 21*B)/256) + 128;

    _____________________________ = Y;

    _____________________________ = Cb;

    _____________________________ = Cr;
```

   To start the MATLAB text editor, use the command `edit`. If you type in the first function declaration line above in the editor and then save the file, the filename will automatically default to the name of your function.

2. Following the same methodology write a function for converting YC_bC_r images to the RGB colour space. Name your function `ycc2rgb`.

3. Load the `jbeans.ppm` image available on the course web page (use the `imread()` function in MATLAB).

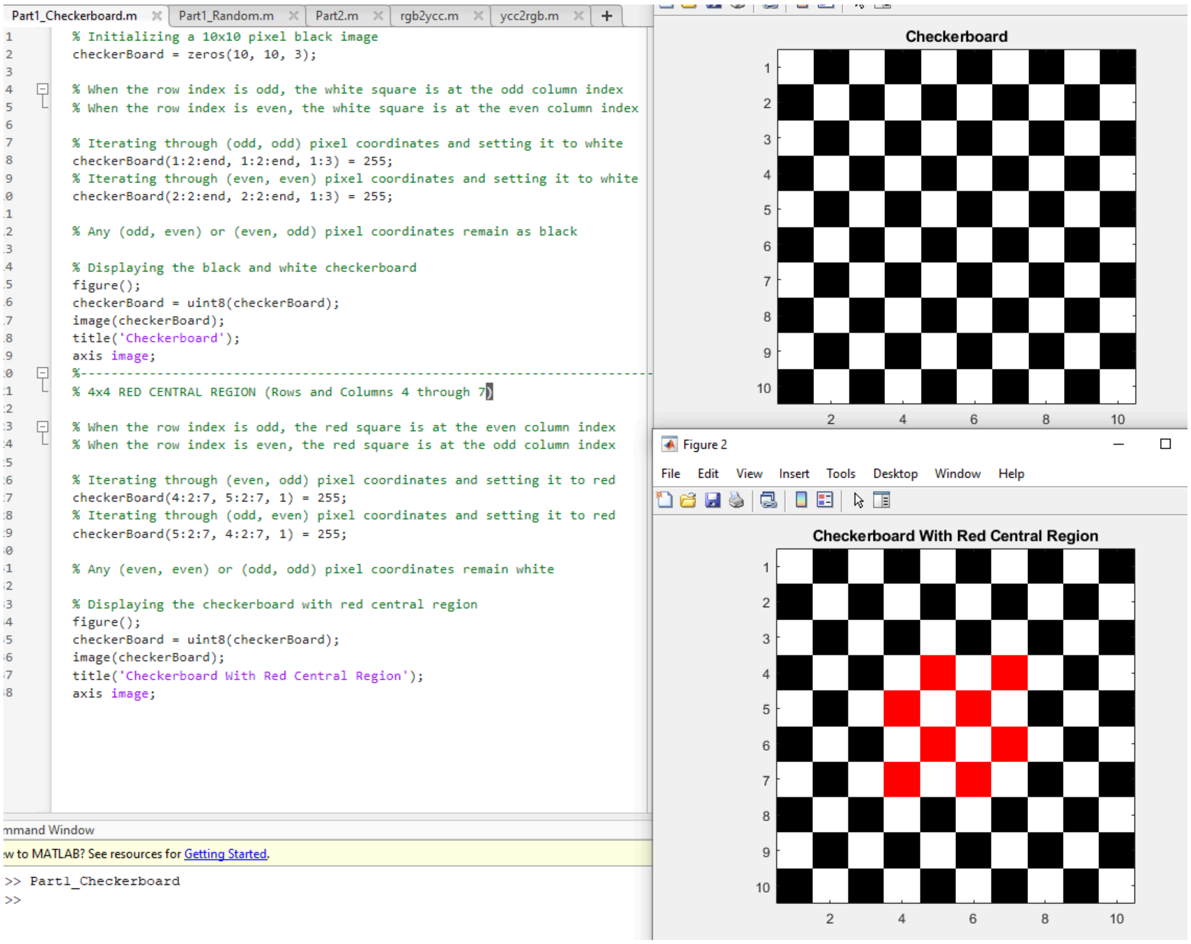
4. Show the following marked steps (to the end of the lab) to the TA at the same time. Confirm that all steps can be performed correctly *before* you show the TA.

   $\boxed{\text{3 Marks}}$ **Show the TA your `rgb2ycc` function and convert the `jbeans.ppm` image to the YC_bC_R colour space.**

2 Marks **Display the Y and the Cr channels of the image using the `imagesc` function and show the TA**

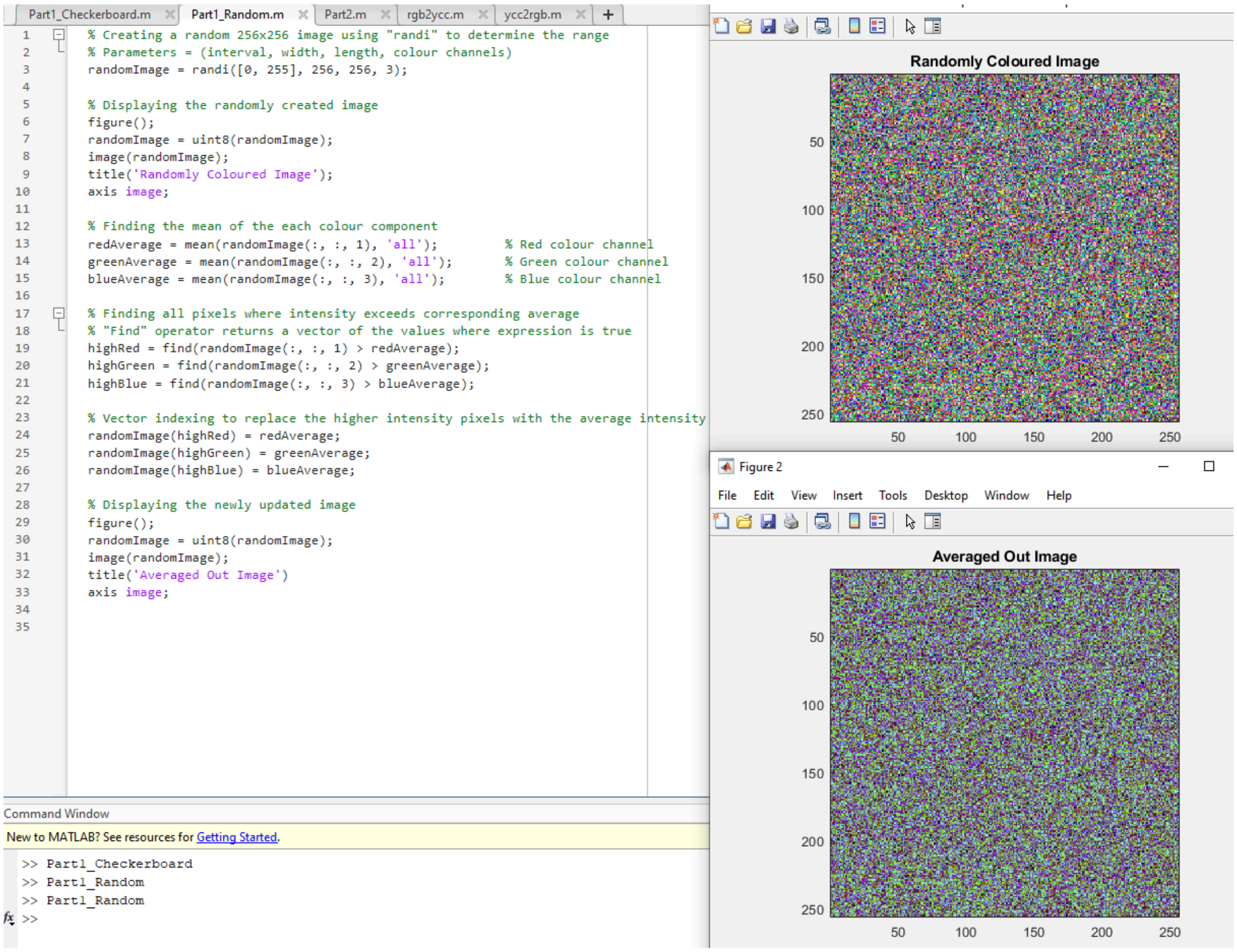
3 Marks **Show the TA your `ycc2rgb` function and convert the image back to the RGB colour space. Display the recovered image and show the TA**

Note that in the YC_bC_R colour space, the image will be in the `double` format since the channels will not necessarily contain integers and may be outside the range `[0,255]` — this is why you should use the `imagesc` function to display the Y channel rather than `image`. However, when you convert back to RGB, you want the image to be back into 8-bit unsigned integer format. Since the values may be non-integers (due to precision errors), you should round the values and convert to `uint8` using the `uint8` function.

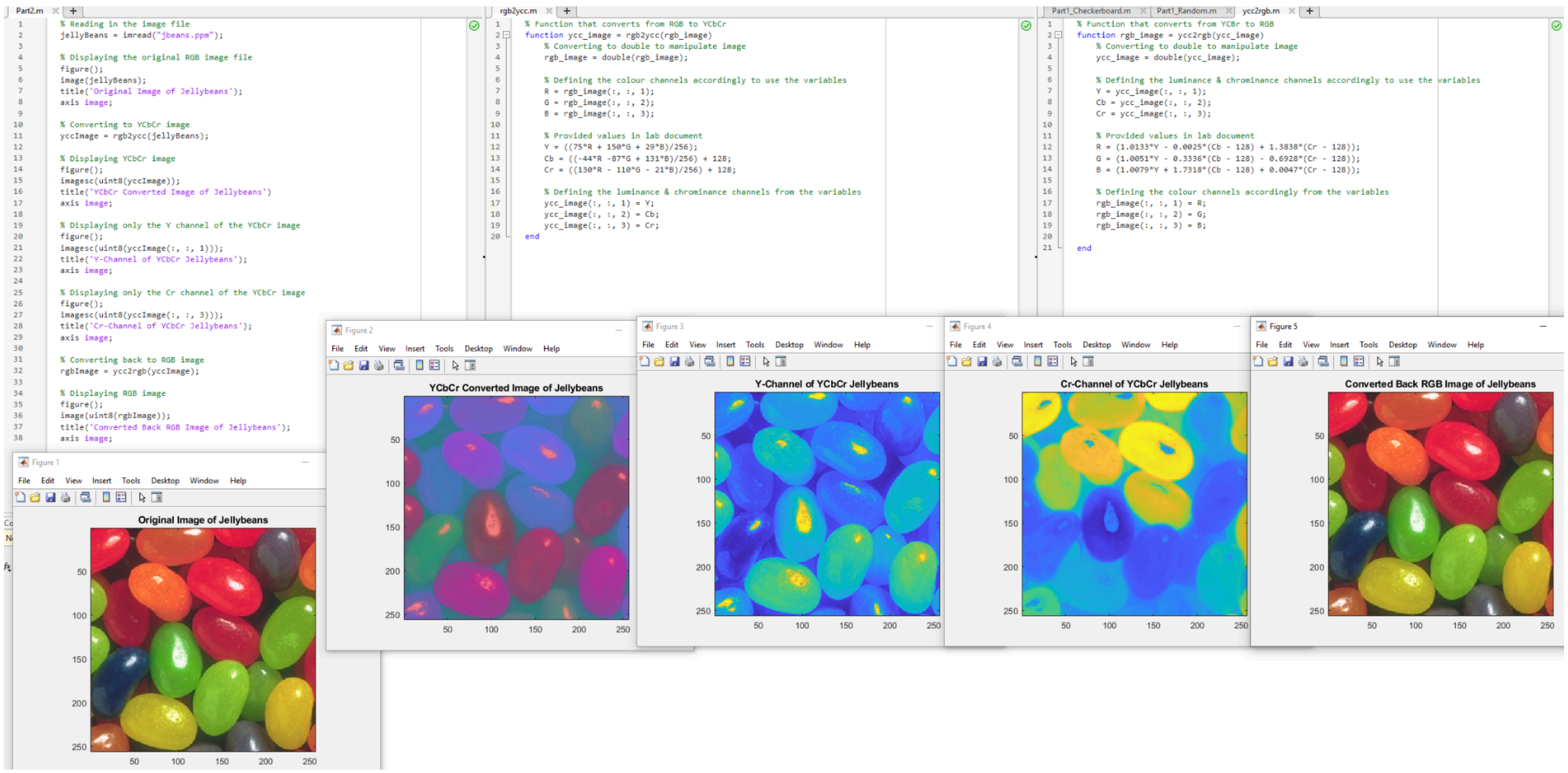


We first initialize a 10x10 black image as a base. For the black and white only image (checkerboard), it should be noted that whenever the row index value is odd, the white square corresponds to the odd column index value. And whenever the row index value is even, the white square corresponds to the even column index value. Whenever the (row index, column index) was (even, odd) or (odd, even), it was a black square. As we already have a black background, we manipulate pixel locations where a white square is required. We iterate through every other row and column index, starting at 1, and set it to white, for the (odd, odd) coordinates. We then again iterate through every other row and column index, starting at 2, and set it to white, for the (even, even) coordinates.

For the red central region, we see it only affects rows and columns 4 through 7. We note that whenever the pixel coordinates are (even, odd) or (odd, even), there is a red square. If the coordinates are (odd, odd) or (even, even), it remains as the originally set white square. So, we iterate through the (even, odd) pixels, by starting the rows at 4 and columns at 5 and setting the alternating ones to red until we reach 7. We iterate through the (odd, even) pixels, by starting the rows at 5 and columns at 4 and setting the alternating ones to red until we reach 7.



We first create a 256x256 image where each pixel is a random colour. We use randi to make each pixel a random colour in the specific range of [0, 255]. We then display this image. Then, we move on to the next step and find the individual averages of the red, green, and blue components over each colour channel and save them in variables. Once we have found all the averages, we can use the find operator and go through each of the colour channels individually and compare the intensity value to the previously found corresponding average. If the expression yields to be true (ie. the intensity at that pixel is higher than the average), it is saved in another variable. Once we have carried out the find operator on all the colour channels, we use vector indexing and iterate through the new variables that contain the higher intensity pixel coordinates, and replace the intensity value with the average value. We then display the new image.



We complete the rgb2ycc function by defining the R, G, and B variables according to the colour channels, so we can access them. After we do the conversion with the provided values in the lab document, we have the Y, Cb, and Cr values in the variables. We then assign these values to the appropriate luminance and chrominance channels.

To complete the ycc2rgb function, we define the Y, Cb, and Cr variables according to the luminance and chrominance channels, so we can access them. After we do the conversion with the provided values in the lab document, we have the R, G, and B values in the variables. We then assign these values to the appropriate colour channels.

First, the original file is accessed and displayed through imread(). We then call the rgb2ycc function to convert from RGB colourspace to YCbCr colourspace and display the image. To display the Y and Cr channels only, when we do image(), we can specifically access the Y channel (yccImage(:, :, 1)) and the Cr channel (yccImage(:, :, 3)).

We call the ycc2rgb to convert the previously converted YCbCr colourspace image back into the RGB colourspace and display it.