EA-1 Lab Project

1 Introduction to Image Processing in MATLAB

Images are stored as matrices in computer memory. What we perceive in the physical world as light intensity is represented as values in these matrices. Since Linear Algebra is based on matrix and vector operations, it has proven very useful for image processing and analysis. In this lab project, we will explore the application of Linear Algebra on image processing to a basic extent.

In a matrix that represents an image, each entry is called a "pixel". For black-and-white images (also called "grayscale" images) the value of each pixel is represented by an integer value in the range [0, 255]. (This is because each pixel is represented by 8 bits in computer memory. A bit is a binary number that is either 0 or 1. 00000000 in binary is 0 and 11111111 in binary is 255. All the other bit combinations correspond to the numbers in between.) The pixel value 0 corresponds to the absolute black color, the value 255 to the absolute white color, while values in between represent different shades of gray.

In order to represent a color image, we would need more bits and a more complex structure than a grayscale image. Each color image is represented by three matrices, each one corresponding to red, green and blue colors, respectively. These matrices are called "channels" of the color image. In other words, a color image of size $m \times n$ can be represented as a $3D$ matrix of size $m \times n \times 3$. Each one of the three layers of this matrix corresponds to the intensity of a different color in the image. The linear combination of these channels is perceived as a color image by human eye. Representation of a color image in a computer is shown in Figure 1.

The reason for MATLAB and Linear Algebra being taught together in EA-1 is that MATLAB is a very useful tool for Linear Algebra problems. Since Linear Algebra constitutes the mathematical basis for image processing, MATLAB is commonly used to create and test image processing algorithms and applications.

As an introduction, here is a list of basic built-in MATLAB functions that are useful for reading

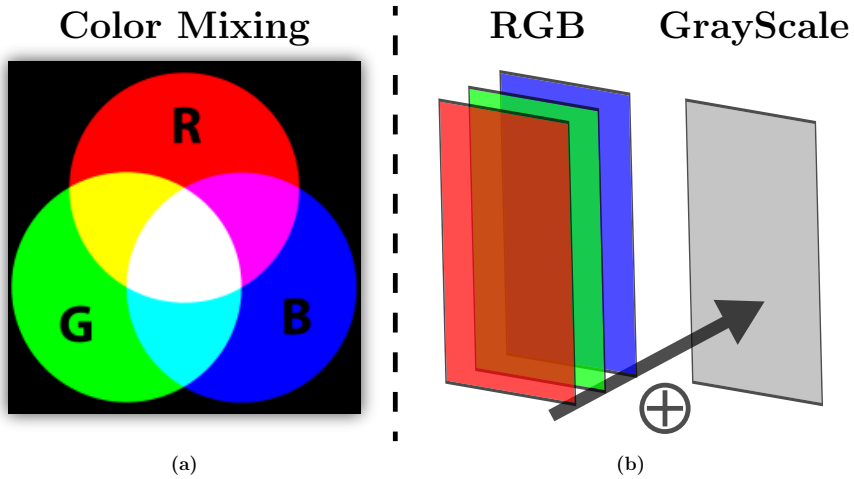

Figure 1: (a) Illustration of the Color Mixing concept in the RGB Color Space; (b) Illustration of transforming an RGB image (3 Channels) to Grayscale (1 Channel). In MATLAB this can be achieved using the function `rgb2gray()`.

and processing images:

- `imread(filename)`: Reads an image with name `filename` from the computer. Note that `filename` must be a string that contains the full path (filename) of the image in your computer. If your current working directory in MATLAB contains this image, then the format `name.ext` is sufficient. Here, `ext` denotes the extension of the image. There are multiple image extensions with many different characteristics (e.g., `.bmp`, `.jpg`, `.tif`, `.png`). In this project, we will use `.png` images which are rather low quality and commonly used for web applications. The `imread()` function reads an image as a `uint8` matrix. It stands for *unsigned 8-bit integer* format, which is a technical term for numbers that are represented by 8 bits, hence with integer values between 0 and 255. (as mentioned above)

- `rgb2gray(img)`: Converts the matrix stored in the variable `img` to a grayscale image, i.e., accepts a $3D$ matrix and returns a $2D$ matrix of values that correspond to different shades of gray. Each value in the grayscale image is determined as a linear combination of the values in the three channels of the image, (red, green and blue), as seen in Figure 1. The coefficients of this linear combination are predefined in the function such that the final shades of gray in the output are properly balanced. In MATLAB, the order of color channels is red, green and blue. So, if you have a $3D$ matrix `img` of size $m \times n \times 3$ the three channels can be obtained with the commands `img(:,:,1)`, `img(:,:,2)`, `img(:,:,3)`, respectively.

- `imshow(img)`: Displays the matrix stored in the variable `img` as an image in the current figure. Note that for the image to be shown properly, it needs to be represented by values between 0 and 255. So, if your data is not in `uint8` format (you can see that in the MATLAB workspace) you need to rescale the values of the image to the range [0, 255] and apply the `uint8()` function to convert it to `uint8` format before displaying it. (`imshow()` also works with images that have values in [0, 1] range. In either case, if values end up being outside of the range limits they are going to be truncated to the minimum and maximum values.)

  You can test the `imread()` and `imshow()` functions using the commands:

  ```
  x = imread('cameraman.tif');
  figure, imshow(x);
  ```

  This should open up a new figure window and display the "Cameraman" image, that is stored internally in MATLAB and commonly used as a standard test image in image processing field.

- `im2double(img)`: Converts a `uint8` matrix `img` into a `double` matrix. (Remember that `double` stands for double precision, that can represent any real number, up to the computer's precision.) This operation also rescales the values in the image to the [0, 1] range. With the double precision format and rescaling, the image becomes ready for processing because most built-in MATLAB commands require double precision variables. In addition, double precision limits the round-off errors during calculations.

  Now test the command `y = im2double(x)`. Note the change in variable type from `uint8` to `double` in MATLAB workspace.

- `imwrite(img,filename)`: Saves the matrix stored in the variable `img` as an image in the file path defined by the string `filename`. The image should first be rescaled to the [0, 255] range and transformed into `uint8` format before using this function.

  Now test the commands:

```
y = uint8(255*y);
imwrite(y, 'myimage.png');
```

This should save the image in your current MATLAB directory and name it `myimage.png`. You can view this image outside of MATLAB with any image viewing application in your computer.

As you see, MATLAB allows you to read and process your images just like Photoshop does. There are many more functions and toolboxes specialized for image processing in MATLAB.

2 Problem Description of the Project

In this project, you will write a set of user-defined functions in order to perform a very basic implementation of face recognition. Face recognition is a very active and popular research topic that is used for a range of different fields from application development to security systems. Each face recognition system requires a face database. (Otherwise it would not know whom to recognize!) As the face database in this project, we will use the faces of 100 National Basketball Association (NBA) players.

The main idea of face recognition is to find the face image in the database that is the most similar to the query image. (Query image is the image that is given to the system as an input to be recognized.) Assume that we have access to a database of face images of people whose identity is known and these images have been labeled with their identity. (In real life, this label can be anything; a name, full name, SSN etc.) Now, if one sends a query face image for recognition, the most straightforward way to recognize this face is to look through the images in the database and find the one which is most similar to the query image. Since the images in the database are labeled, finding the image with the most similar appearance implies that the identity of the person in the image can be found, assuming that the face of this person exists in the database. Obviously, the problem is not that simple and there is ongoing research for images that have been corrupted by noise, occlusions, facial expressions, lighting variations, etc. However, in this project we will deal with uniform and straightforward images.

In the database, we have access to 100 RGB images (color images) of 100 different players. The size of each one of the R, G and B channels in each image is $m \times n = M$. We need to read these images, convert them to grayscale, vectorize (see Fig. 2) them and store them in the columns of a database matrix D of size $M \times 100$. In the database, each column should contain a vectorized image of $m \times n = M$ pixels and there should be 100 such columns. (because there are 100 players) Furthermore, we also need a label vector $\mathbf{p}$ of size 100×1 that holds the name of the k-th player at its k-th entry, where $k = 1 \dots 100$.

Since we know the location of each player in the $\mathbf{p}$ vector, the easiest way to perform face recognition would be to reorder the columns of the database D such that the vectorized image of the k-th player is at the k-th column of the D matrix. Then, when we have a query image, we can find the column of the D matrix that holds the vectorized image which is most similar to our query image and then identify the player as player-k with name Name-k.

2.1 How do we measure similarity between images?

Similarity between images can be defined by using specific cost functions that measure the difference (*distance*) between two images. Assuming that we have two vectorized images in vectors $\mathbf{x}$ and $\mathbf{y}$ which are both of size $M \times 1$, the most commonly used metrics that measure their difference are

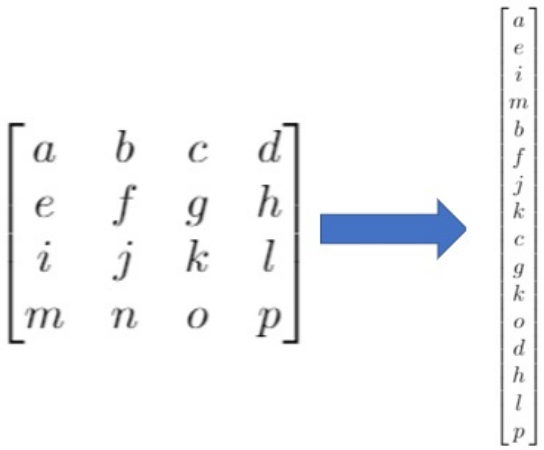

Figure 2: Vectorizing a matrix

the Mean Squared Error (MSE) and the Peak Signal-to-Noise Ratio (PSNR) which are directly related to each other. Their definitions are:

$$\mathrm{MSE} = \frac{1}{M}\sum_{i=1}^{M}(x_i - y_i)^2\,, \tag{1}$$

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{\mathrm{MAX}_{\mathbf{x}}^2}{\mathrm{MSE}}\right), \tag{2}$$

where $\mathrm{MAX}_{\mathbf{x}}$ is the maximum possible value of an entry in the vectors $\mathbf{x}$ and $\mathbf{y}$. In image processing, this value is usually 255 when the images are of type `uint8` or 1 when the images are of type `double` and normalized.

For the MSE metric, lower values mean higher similarity while for the PSNR metric, higher values mean higher similarity. Therefore, if the vectorized image at the k-th column of the database matrix D gives the minimum MSE or the maximum PSNR with the vectorized query image, this means that the image on the k-th column of D is the one with the closest *distance* to our query image. We can display the identity of the k-th player along with the query image and the k-th image for visual comparison.

2.2 Goal of this Project

The goal of this project is to implement an algorithm that reorders the columns of a player image database, as described above, so that the program correctly recognizes the player in a given query image. Most of the code is actually written for you. You need to create a number of user-defined functions that will make the given main script run smoothly. (Think of this as finishing up a puzzle that is more than halfway complete.)

The main steps of the algorithm are:

1. Construct the image database matrix D. At first, vectorized player images will be randomly placed in this database matrix.

2. Unscramble (sort) the database by reordering its columns. The vectorized image from the file `player1.png` should end up on the 1-st column of the database, the vectorized image from `player2.png` should end up on the 2-nd column of the database and so on.

3. Verify that the database has been properly sorted. To do this, you need reuse the same sorting method but providing the result of the previous sorting as its input. If the columns of the database do not change after the second sorting, this means the columns are successfully sorted from Player 1 to 100. (i.e. the database is unscrambled) The sorting procedure will take some time to run. However, once your function is implemented correctly and the columns are properly sorted, the script will automatically store your sorted database matrix and will not try to sort your database the next time you run your code.

4. Plot the order of the indices before and after sorting them.

5. Read the image of player k. (your query image)

6. Find the column in both databases (unsorted and sorted) that is the most similar to your query image by comparing the MSEs.

7. Calculate the PSNR between your query image and all images in both databases. (unsorted and sorted).

8. Pass these results as input arguments to the function `identifyPlayer()` in order to see the wrong and correct identification of your image by the unsorted and sorted databases, respectively.

All these steps have already been implemented for you in the main script `LabProject.m` but some of the user-defined functions used in it are missing. Please carefully read Section 2.3 to understand what you need to implement and follow the instructions in Section 3 to write the necessary functions. The described steps of the algorithm are visualized in Figure 3.

Before you start, the following items are given to you:

- A set of RGB images for all the players. (in the folder `Player_Images`).
- The complete function `createImageDatabase(imagePath)` which reads all the RGB images in the folder `imagePath`, converts them to grayscale, vectorizes them and places them randomly in the columns of the database matrix D.
- The main script `LabProject.m` which you need to run for the face recognition task. This script is complete, so except for the player number at the beginning, you **should not** make any changes on it. The script has been designed to run without any warnings once you have written all the correct functions, as described in Section 3.
- The empty function `unScrambleDatabase.m` which you will need to fill.
- The complete function `identifyPlayer()` that holds the label vector **p** and displays the results presented in Figures 5 and 6.

Note: Functions `createImageDatabase()` and `identifyPlayer()` are provided in `.p` file format. This format is essentially the same as a MATLAB `.m` file but cannot be accessed for editing or viewing. These two functions are already written and placed in the main script.

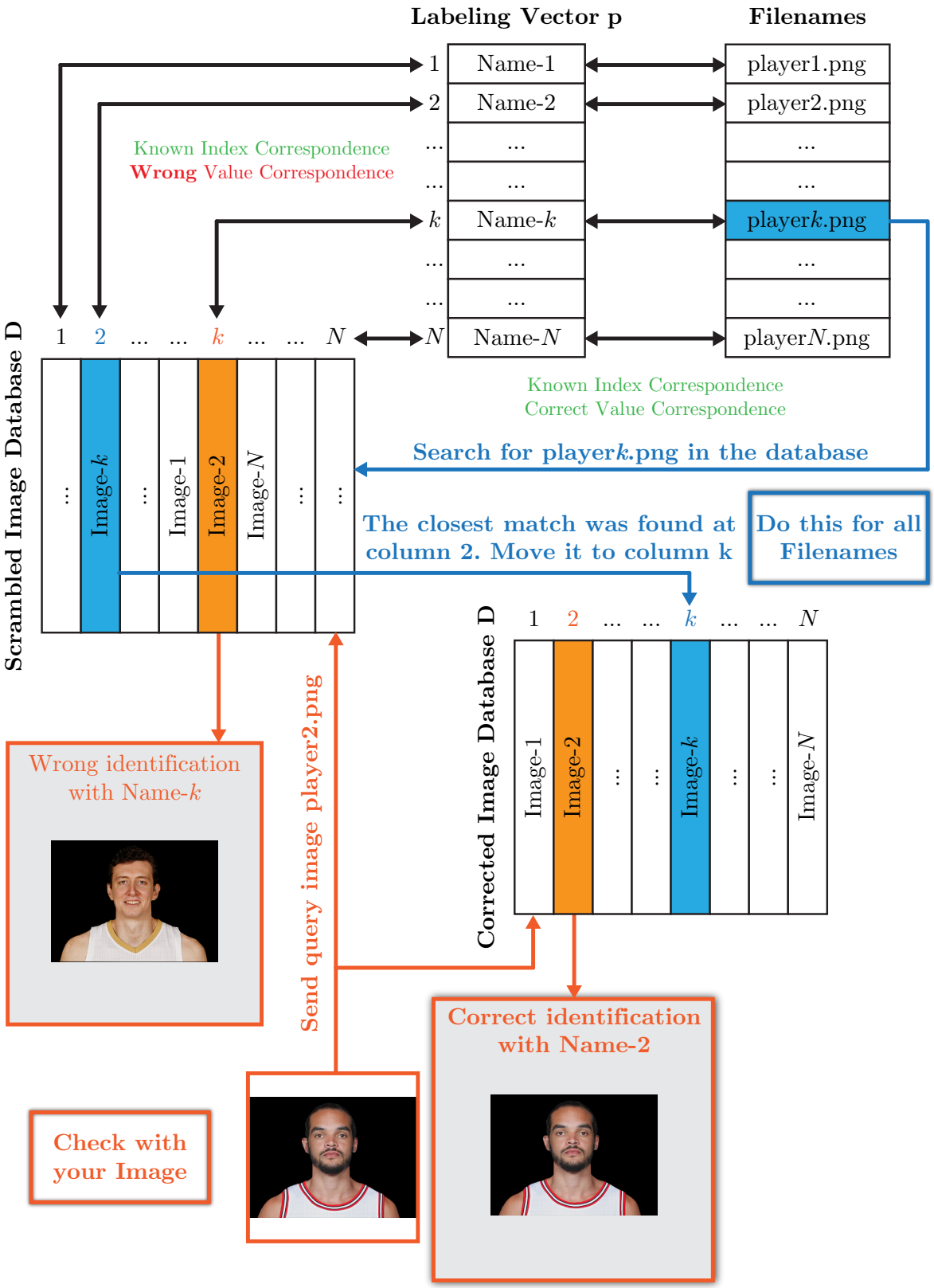

Figure 3: Illustration of the logic of the Face Recognition algorithm.

2.3 How to Proceed

1. Look at the images located under the folder `Player_Images` and pick the image of your favorite NBA player. This will be your query image. Assign the number of his image to the variable `PlayerNumber` at the `Initialization` section of the main script `LabProject.m`.

2. Try to run `LabProject.m`. You will notice that you get a set of `warnings`. This is because some user-defined functions that are used in the script are missing and the code is not working properly.

3. Implement all the functions explained in Section 3 and test the script `LabProject.m` until all the warnings disappear. Your whole script should run properly (without any warning/error) once you have implemented all the functions correctly.

4. If all the functions are written correctly and `LabProject.m` runs properly, you should see 3 output figures:

   (a) The indices of the vectorized images in the unsorted (scrambled) and sorted (unscrambled) databases, as shown in Figure 4.

   (b) The output from the unsorted database which recognizes the wrong player, as shown in Figure 5.

   (c) The output from the sorted database which recognizes the correct player, as shown in Figure 6.

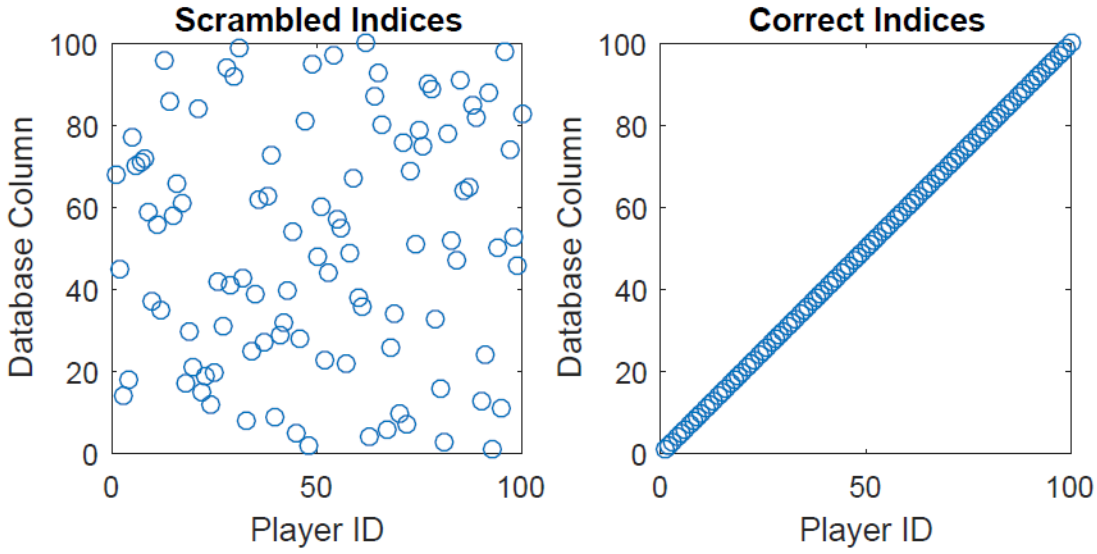

Figure 4: Indices of the unsorted and sorted databases.

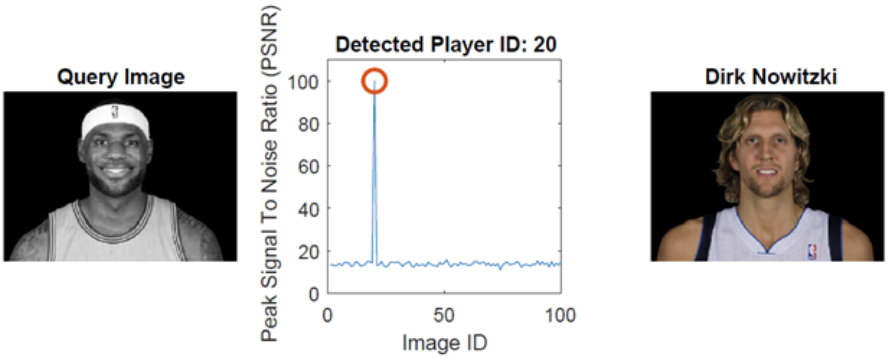

Figure 5: Example for erroneous recognition using the unsorted database.

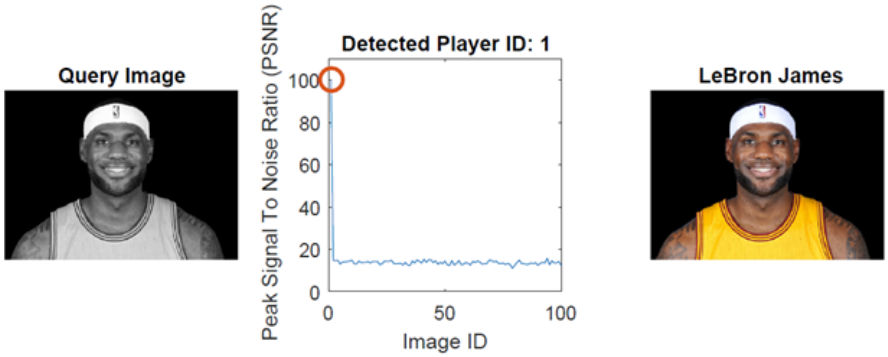

Figure 6: Example for correct recognition using the sorted database.

3 Set of Functions to be Implemented

You will create a set of user-defined functions. Make sure all of them are in the same folder with the player images, the given functions and the main script. You will write 8 functions in total. (7 of them from scratch and 1 by filling in the given empty function.)

- `function vecOut = makeVector(matrixIn)`

  This function takes a matrix (`matrixIn`) as input and returns the vectorized version of this matrix (`vecOut`) as output. In this function, you should:

  1. Check that the input is numerical (use the built-in function `isnumeric()`), otherwise report a relevant `error` message.
  2. Check the dimensions of the input matrix (use the built-in function. `ndims()`) If it has more than two dimensions the function should report a relevant `error` message.
  3. Vectorize the input matrix and assign the vector to the output variable `vecOut`. (There is a very easy notation for this, look it up!)

- `function MSE = calcMSE(x1,x2)`

  This function takes two column vectors (`x1`, `x2`) as inputs, calculates their Mean Squared Error (MSE) (use Eq. 1) and assigns the result to the output variable `MSE`. No error checking is necessary in this function. In order to make sure that the inputs are vectors, use `makeVector()` function above before calculating the MSE.

- `function PSNR = calcPSNR(x1,x2,maxX)`

  This function takes two column vectors (`x1`, `x2`) and a maximum value for their elements as inputs and calculates their *Peak Signal-to-Noise Ratio* (PSNR) (use Eq. 2). It assigns the calculated value to the output variable `PSNR`.

  In this function, you should:

  1. Assign a default value of 1 to `maxX`.
  2. If the MSE of the two input vectors is 0, set the PSNR to 100.
  3. If the MSE of the two input vectors is not 0, calculate the PSNR using Eq. 2.

  Note that you need to use `calcMSE()` in this function.

- `function PSNRs = computePSNRs(imgVec, imageDatabase)`

  This function takes a vectorized image (`imgVec`) and a database matrix of vectorized images (`imageDatabase`) as inputs and computes the PSNR between `imgVec` and each image (column) in the database matrix. The calculated PSNRs for all images in the database should be stored in an array of correct size and assigned to the output variable `PSNRs`. Note that you need to use `calcPSNR()` in this function.

- `function minPos = findMinimumErrorPosition(imgVec, imageDatabase)`

  This function takes a vectorized image (`imgVec`) and a database matrix of vectorized images (`imageDatabase`) as inputs and finds the position (column index) of the vectorized image in the database matrix that produces the smallest MSE with the input `imgVec`. The result (column index) is assigned to the output `minPos`. In this function, you should:

  1. Calculate the MSE of `imgVec` and each column in the `imageDatabase`.
  2. Find the position (column index) where the MSE is minimum. You can use the built-in functions `min()` and `find()`.

  Note that you need to use `calcMSE()` in this function.

- `function image = readImage(imgName)`

  This function takes the path `imgName` as a string input and reads the image in that path. If the input is a color image, the function converts it to grayscale and normalizes it. The resulting image is assigned to the output variable `image`. (If you are not sure what a path is, check Line 19 of `LabProject.m` to see the notation.)

  In this function, you should:

  1. Read the image from the input path `imgName` using the built-in function `imread()`.
  2. If the image is three-dimensional (which means it is a color image), convert it to grayscale using the MATLAB's built-in function `rgb2gray()`.
  3. Convert the image from `uint8` type to `double` type and normalize its intensity in the range $(0, 1)$ for processing purposes. This can be done by using the MATLAB's built-in function `im2double()`.

  Note that all these built-in functions have already been introduced above. You can test your `readImage` function using the commands:

```
x = readImage('cameraman.tif');
figure, imshow(x);
```

This should open up a new figure which will display the "Cameraman" image which is stored internally in MATLAB.

- `function [newDatabase, indices] = unScrambleDatabase(imagePath, database)`

  This is the given empty function to fill in. It takes the `imagePath` and the scrambled (unsorted) `database` matrix as inputs. The function reads all the images from the `imagePath` and searches for their position (column index) in the scrambled database. After finding the indices, it unscrambles (sorts) the database by reordering its columns, using these indices. In this function, you should:

  1. Initialize the variable `indices` as an array of 0's of proper dimensions. (Use the number columns in the database matrix.)
  2. Using a loop:
     (a) Read the ii-th image in the path `imagePath` with the function `readImage()` you created above. Note that you need to use the code in Line 19 of `LabProject.m` to get the images. (playerNumber will be replaced with ii.)
     (b) Vectorize the image
     (c) For the vectorized ii-th image, find the minimum error position in the database using the function `findMinimumErrorPosition()` you created above.
     (d) Store the minimum error position in the ii-th position of the array `indices`. This should store the correct database indices to the `indices` array.
  3. After finding all the correct indices with the loop, reorder the columns in the database so that the columns are sorted by the player image numbers, as explained above. This can be implemented with a very simple indexing operation using the `indices` array.
  4. Finally, assign the sorted database to the output variable `newDatabase`.

- `function plotIndices(scrambledIndices, correctIndices)`

  This function takes the `scrambledIndices` and the `correctIndices` as inputs and creates a figure with two plots for comparison. The result should look similar to Figure 4. Note that each time you run the program, the database is created in a randomized order. Therefore, the scrambled indices will look different at each run. The correct indices, however, should always be the same and look exactly like Figure 4. In this function you should:

  1. Create a new figure using MATLAB's built-in function `figure()`.
  2. Use the built-in function `subplot()` to put both plots in the same figure.
  3. Plot the indices. Use *Marker Type* `'o'` as another input to create the scatter plot.
  4. Make the `axis` of the figure `square`.
  5. Limit both x and y `axis` in the range of [0, number of indices].
  6. Use the `title`, `xlabel` and `ylabel` given in Figure 4.
  7. Make sure that your resulting figure looks very similar to Figure 4.

  Note that you need to remember many plotting tools that we covered earlier in the quarter. If you have forgotten how to plot arrays in MATLAB, look it up!

4 Lab Project - Submission Instructions

After you write all the functions and the main script runs as described, please submit your project. Your submission should include the following:

1. A zipped (compressed) folder containing all the project files. You can only leave the `Player_Images` folder out, since it is too large.

2. A published version of your code in html format. In order to produce this, go to the "PUBLISH" tab on MATLAB and click on the "Publish" button. Let the code run and produce all the results (this might take some time). Now, you should have a folder named "html" under your MATLAB working directory. Rename this folder as "html_your_name", zip (compress) it and submit this compressed folder as well. (Make sure the figures that your code will produce are in that folder!)