COT-5930 - MATLAB Assignment 1

Christian Garbin

Question 1

Select four grayscale images *available im MATLAB*, read their file name and size with `imfinfo`, store the resulting information in an array of structures, save the array to a `.mat` file.

Step 1 - define the list of images we will use

```
files = ["pout.tif", "coins.png", "cameraman.tif", "rice.png"];
num_files = length(files);
```

Step 2 - preallocate an array to store image information

References:

- Preallocating arrays is recommended by MATLAB to increase performance.
- Stack Overflow discussion on how to initialize arrays of structures.

```
image_data = repmat(struct('name', "", 'width', 0, 'height', 0), 1, num_files);
```

Step 3 - Populate the array with image data and show what we read

References:

- Applying a function to all elements in an array.

```
for i = 1:num_files
    info = imfinfo(files(i));
    image_data(i).name = files(i);
    image_data(i).width = info.Width;
    image_data(i).height = info.Height;
end

arrayfun(@(x) disp(x.name + ', ' + x.width + ', ' + x.height), image_data)
```

```
pout.tif, 240, 291
coins.png, 300, 246
cameraman.tif, 256, 256
rice.png, 256, 256
```

Step 4 - Save to a file

```
file_name = 'image_data.mat';
save(file_name, 'image_data')
```

Step 5 - verify that we saved it correctly

Not that we doubt MATLAB's ability to save a file, but we doubt we are using it correctly.

References:

- Removing a variable from the workspace, to make sure it's not "working" by mistake, using a value set by the code above.
- Checking conditions with assert.

```
clear image_data
load(file_name)
assert(length(image_data) == 4)
arrayfun(@(x) disp(x.name + ', ' + x.width + ', ' + x.height), image_data)
```

```
pout.tif, 240, 291
coins.png, 300, 246
cameraman.tif, 256, 256
rice.png, 256, 256
```

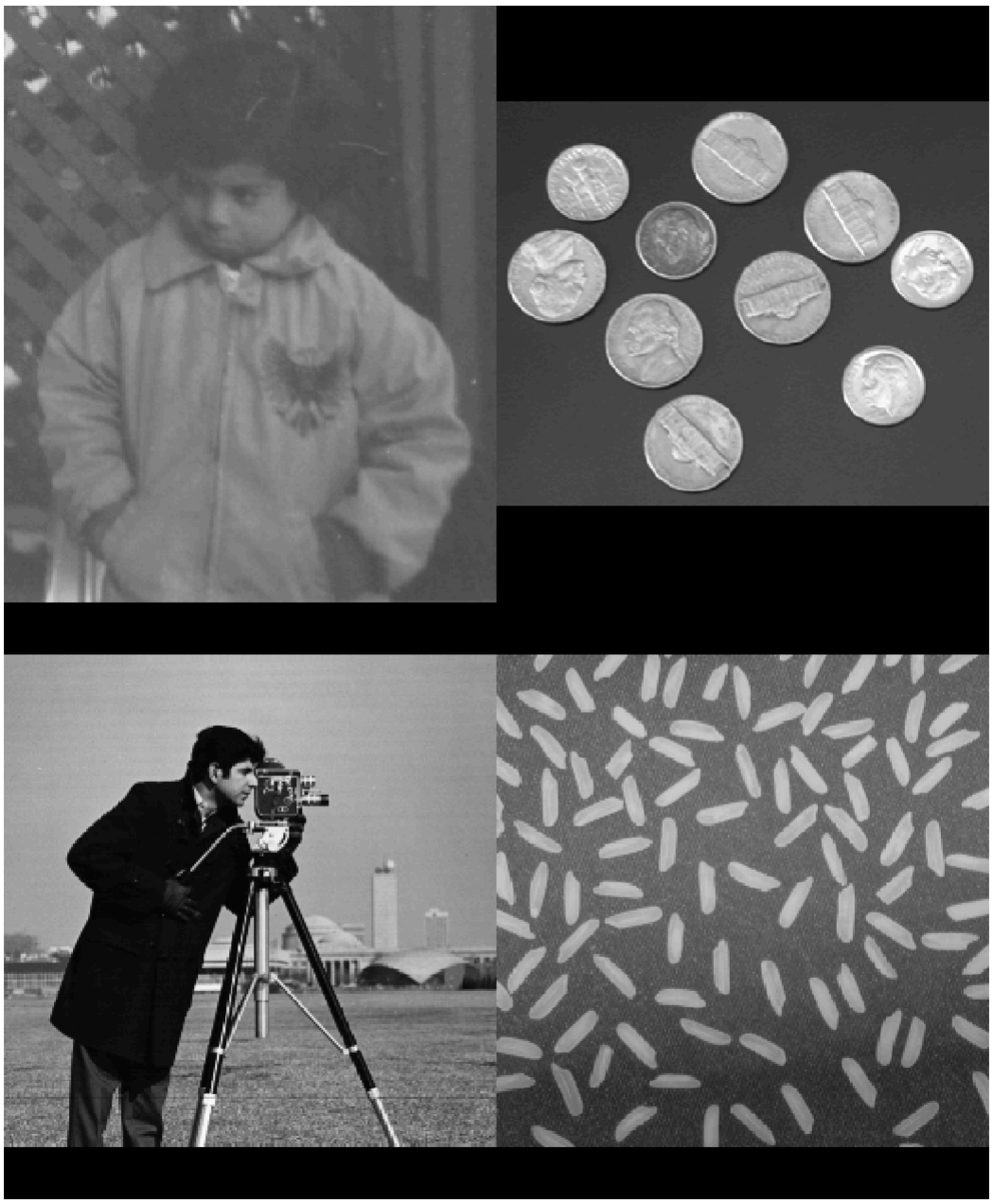
Question 2

Load the `.mat` saved in question 1, display the images in a 2x2 arrangement.

References:

- Extracting a field from a structures or array of structures (requires the Mapping toolbox).

```
clear image_data % make sure it's not "working" by accident
load('image_data.mat')
files = extractfield(image_data, 'name');
montage(files, 'Size', [2 2])
```



Question 3

Read, open, and display a grayscale image *available in MATLAB*. Calculate and display its maximum, minimum, and average gray level.

References:

- Why we need "figure, " before showing an image. It is not strictly necessary in this case, but a good habit to have, in case the code is extended to show more pictures in the future.

```
img = imread("pout.tif");
```

```
fprintf("Gray levels: maximum=%d, minimum=%d, average=%f", ...
    max(img(:)), min(img(:)), mean(img(:)))
```

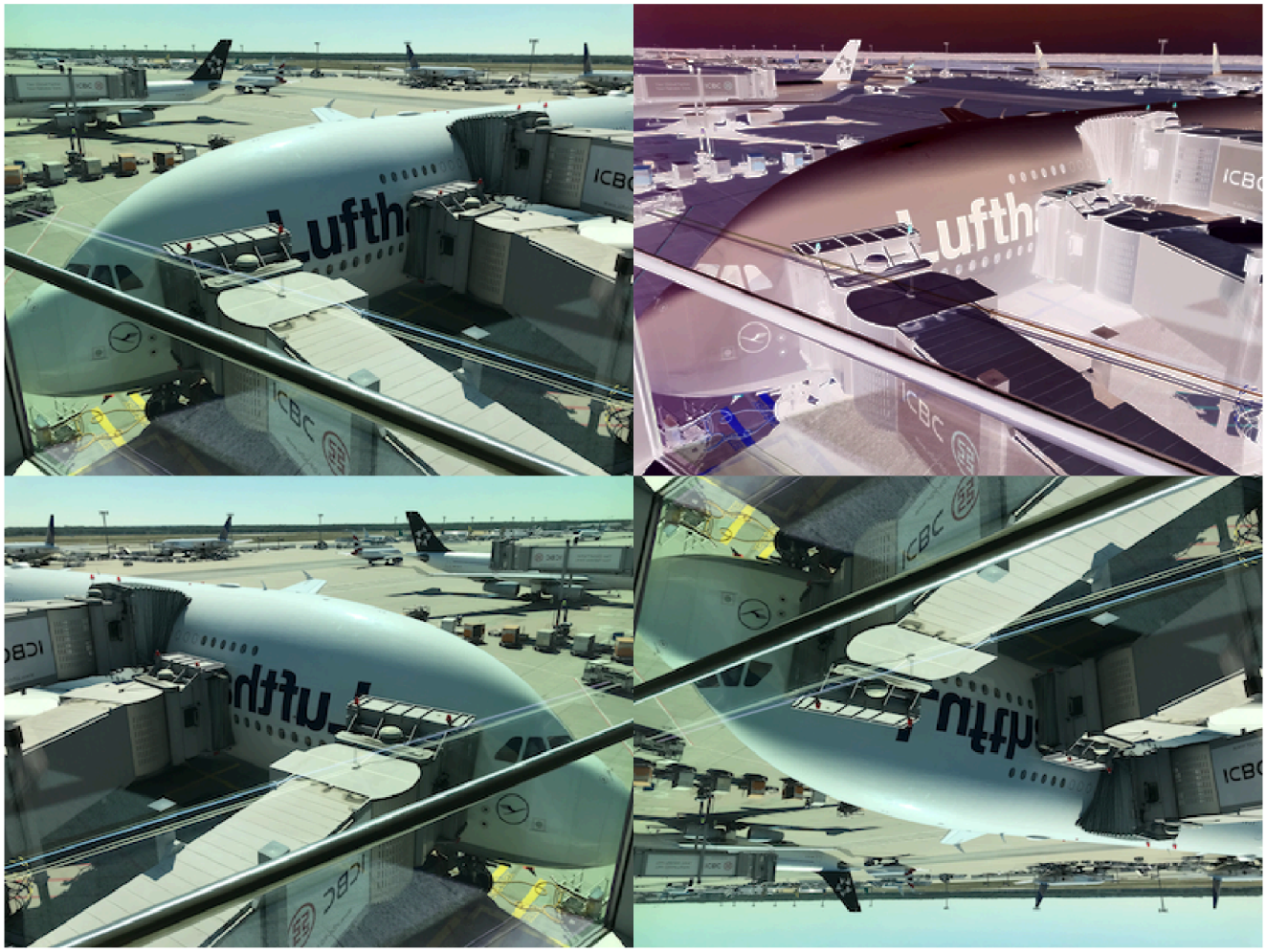
```
Gray levels: maximum=224, minimum=74, average=110.303694
```

```
figure, imshow(img)
```

Question 4

Ask for the name of a picture, show in a 2x2 arrangement the original image, the image flipped upside-down, the image flipped left side-right, the negative of the image.

```
[file, path] = uigetfile({'*.png;*.jpg;*.jpeg;*.tif;*,bmp', 'Pictures'},...
               'Select an image');
if file ~= 0
    img = imread([path file]);
    montage({img, imcomplement(img), fliplr(img), flipud(img)}, 'Size', [2 2]);
end
```



Question 5

A beginner MATLAB programmer wrote the code below to darken a grayscale image by reducing the intensity of each pixel by half.

```
I = imread('coins.png');
tStart = tic; % do not count the time to read the file
[height, width] = size(I);
J = uint8(zeros(size(I)));
for i=1:height
    for j=1:width
        J(i,j) = 0.5 * I(i,j);
    end
end
figure, imshow(I)
```

```
figure, imshow(J)
```

```
tEnd = toc(tStart);
fprintf("Elapsed time - beginner's code: %fs\n", tEnd)
```

```
Elapsed time - beginner's code: 1.188535s
```

Questions about the code

a) Does the script work as expected?

Yes

*b) Will it still work if you remove the **uint8** typecasting in line 3? Why (not)?*

No, because the default type for `zeros` is double.

c) What is the main problem with this code? Be specific!

It uses for loops to apply the same operation on a matrix (array). It should use vectorized operations instead.

d) Rewrite the code to improve its efficiency while keeping it readable. Hint: vectorize for loops!

See next code sections.

*e) Compare the performance of the original code against the modified version you wrote, using **tic** and **toc**.*

See timing printed after code section.

Better code 1 - Vectorized operations

Solves the main problem with the original code: replace the `for` loops with vectorized operations.

```
Ib = imread('coins.png');
tStart = tic;
Jb = Ib * 0.5;
figure, imshow(Ib)
```

```
figure, imshow(Jb)
```

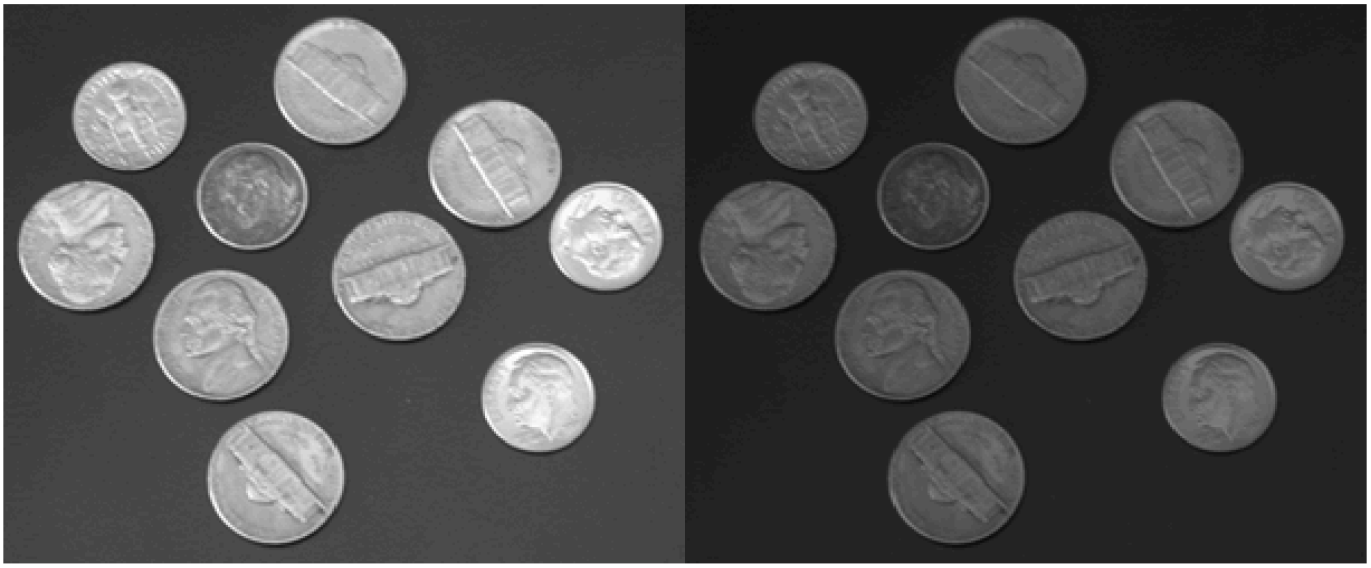
```
tEnd = toc(tStart);
fprintf("Elapsed time - vectorized: %fs\n", tEnd)
```

```
Elapsed time - vectorized: 0.305028s
```

Better code 2 - No temporary variable, improved UI

Improves the UI by showing the original and modified images side by side. As a bonus, it is improves the time to show the images to the user (one call to a graphing/windowing function instead of two). It does not improve as much as it does in the script (`.m`) file (see notes and timing in that file), presumably because in this case (live script), it does not need to open a separate window.

```
Ic = imread('coins.png');
tStart = tic;
montage([Ic, Ic * 0.5])
```



```
tEnd = toc(tStart);
fprintf("Elapsed time - vectorized + montage: %fs\n", tEnd)
```

```
Elapsed time - vectorized + montage: 0.250471s
```