ECE 253: HW1

Academic Integrity Policy: Integrity of scholarship is essential for an academic community. The University expects that both faculty and students will honor this principle and

in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind.

By including this in my report, I agree to abide by the Academic Integrity Policy mentioned above.

****

To run all files at once, load folder named “main” into Current Folder section in MATLAB(This contains all scripts as well as all images used). Then run “main” in the MATLAB console. This file will output results for the whole HW. To run each section individually, follow directions in each section below.

****

Ajinkya Bagde
U07800884
10/17/2017

Problem 1: MATLAB Basics

To run, type “prob1” into MATLAB console
Output of prob1.m below:

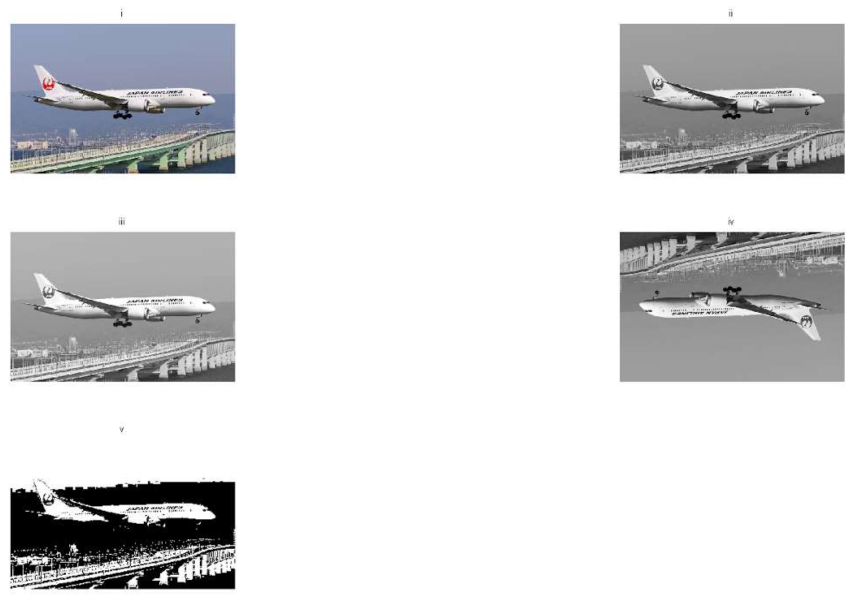
```
C =

     0     9     0     1
     0    25     4     3
     0     0     0     9
     6    32     0     0
     0     8     0     0


inner_product =

   200

Maximum value is 32 located at these location(s):
4, 2
Minimum value is 0 located at these location(s):
1, 1
2, 1
3, 1
5, 1
3, 2
1, 3
3, 3
4, 3
5, 3
4, 4
5, 4
```

```
%%Problem 1: MATLAB Basics
clear;
clc;

A = [3 9 5 1;4 25 4 3;63 12 23 9;6 32 77 0;12 8 5 1];
B = [0 1 0 1;0 1 1 1;0 0 0 1;1 1 0 1;0 1 0 0];

%%(i) Point-wise multiply A with B and set it to C.
C = A.*B

%%(ii) Calculate the inner product of the 2nd and 5th row of C.
inner_product = dot(C(2,:),C(5,:))

%%Find the minimum and maximum values and their corresponding row and column indices in
%%matrix C. If there are multiple min/max values, you must list all their indices.

%To find max value in matrix
value = max(C(:));
%Now to find all instances of that value, and get indices
extrema = find(C(:) == value);
[row,col] = ind2sub(size(C),extrema);
fprintf('Maximum value is %d located at these location(s):\n',value)
fprintf('%d, %d\n', [row,col]')

%Same thing for min
value = min(C(:));
%Now to find all instances of that value, and get indices
extrema = find(C(:) == value);
[row,col] = ind2sub(size(C),extrema);
fprintf('Minimum value is %d located at these location(s):\n',value)
fprintf('%d, %d\n', [row,col]')
```

Problem 2: Simple Image Manipulation
To run, type ‘prob2’ into MATLAB console

Output of prob2.m below. To run, make sure ‘JAL.jpg is in the current folder’:

To flip an image horizontally → **horiz_flip = image(:,end:-1:1)**. This basically reverses the order of columns.
To flip an image vertically → **vert_flip = image(end:-1:1,:)**. This basically reverses the order of rows .



i)

JAPAN AIRLINES

ii)

JAPAN AIRLINES

iii)

JAPAN AIRLINES

iv)

v)

JAPAN AIRLINES

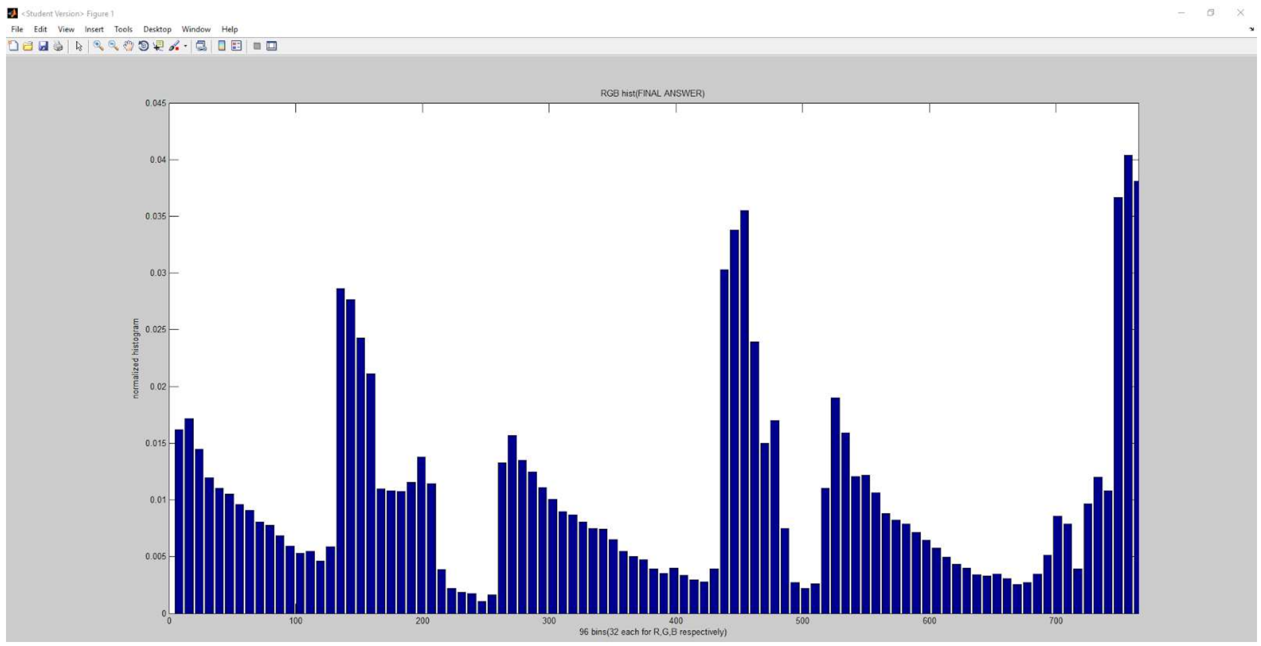
```
%Problem 2: Simple image manipulation

%(i) Download any color image from the Internet with a spatial resolution of no more than (720
%X 480). Read this image into MATLAB. Call this image A.
A = imread('JAL.jpg');

%(ii) Transform the color image to grey-scale. Verify the values are between 0 and 255. If not,
%please normalize your image from 0 to 255. Call this image B.
B = rgb2gray(A);

%(iii) Add 20 to each value of image B. Set all pixel values greater than 255 to 255. Call this
image
%C.
C = double(B);
C = B + 20;
C(C > 255) = 255;

%(iv) Flip image B along both the horizontal and vertical axis. Call this image D.
%end:-1:1 - reverses order of elements in an array/matrix
%B(:,end:-1:1) is a horizontal flip
%B(end:-1:1,:) is a vertical flip
%If we were dealing with a RGB image, we would use (:,end:-1:1,:),etc
D = B(end:-1:1,end:-1:1);

%(v) Calculate the median of all values in image B. Next, threshold image B by the median
value
%you just calculated i.e. set all values greater than median to 1 and set all values less than or
%equal to the median to 0. Name this binary image E.
median_B = median(B(:));
E = B;
E(E <= median_B) = 0;
E(E > median_B) = 1;
E = logical(E);

%Show images in subplot
subplot(3,2,1)
imshow(A)
title('i')

subplot(3,2,2)
imshow(B)
title('ii')
```

```
subplot(3,2,3)
imshow(C)
title('iii')

subplot(3,2,4)
imshow(D)
title('iv')

subplot(3,2,5)
imshow(E)
title('v')
```

Problem 3: Histograms

To run:

```
>>img = imread('geisel.jpg');
>>compute_norm_rgb_histogram(img)
```



Labels are a bit hard to see:

x -axis range: 0 → 765

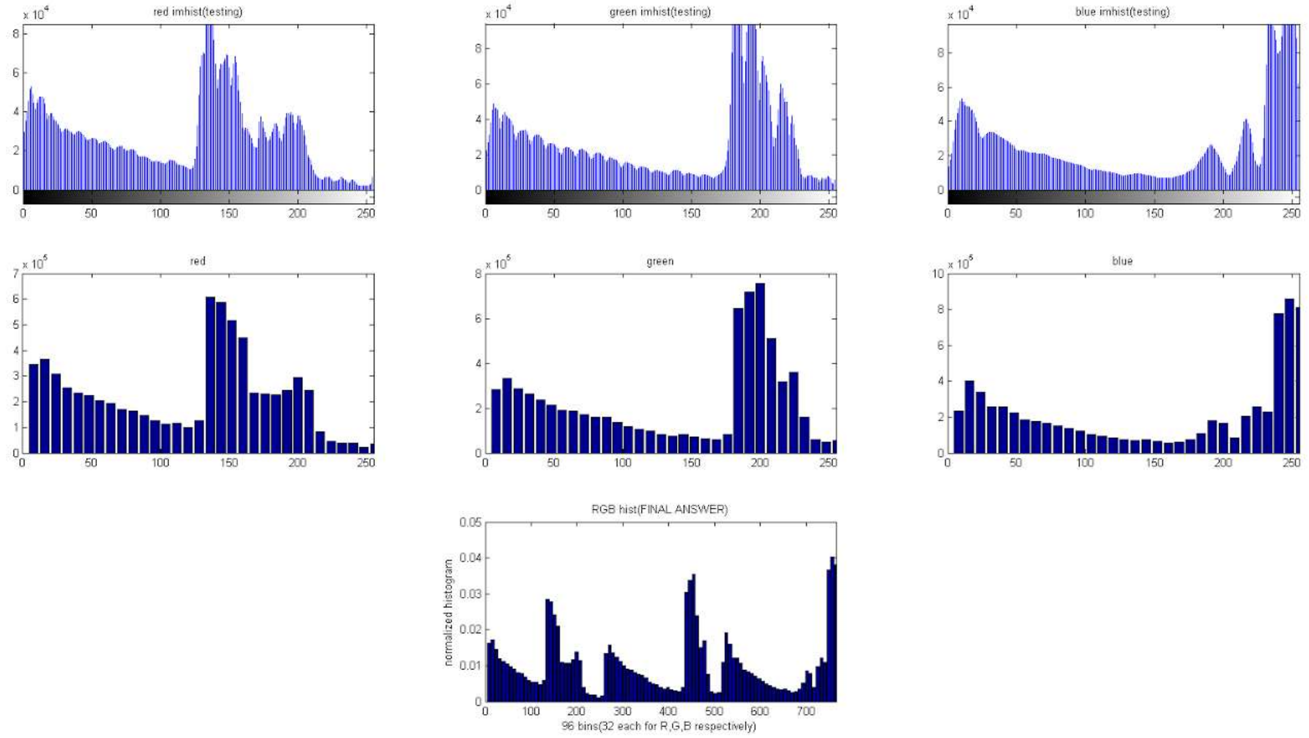
x-axis label: '96 bins(32 each for R,G,B respectively)'

y-axis label: 'normalized histogram'

Title: 'RGB hist(FINAL ANSWER)'

Approach was as follows:

Image was separated into R,G,B portions, and each portion was converted into an array to make processing easier. Then each array was parsed and the values were placed into 32 bins(32 bins for each color). At this point, individual histograms were constructed, shown below. Then the R,G,B histograms were concatenated together, and normalized. That is the final result shown above.


red imhist(testing)
green imhist(testing)
blue imhist(testing)
x 10^4
red
green
blue
x 10^5
RGB hist(FINAL ANSWER)
normalized histogram
96 bins(32 each for R,G,B respectively)

```
%Problem 3: Histograms


function rgb_hist = compute_norm_rgb_histogram(input_image)

%First we have to read in the image
%We pass the function the image --> compute_norm_rgb_histogram(image_name)

%imshow(input_image)

%Now seperate image into individual RGB components
red = input_image(:,:,1);
green = input_image(:,:,2);
blue = input_image(:,:,3);

%convert to arrays to work with easier
red_array = reshape(red,1,[]);
green_array = reshape(green,1,[]);
blue_array = reshape(blue,1,[]);

%Now compute invidual histograms(5-bit, 32 bins)
red_hist = zeros(1,32);
green_hist = zeros(1,32);
blue_hist = zeros(1,32);

for bin = 1:32
    for x=1:length(red_array)       %All arrays are the same size
        t_min = (255/32)*(bin-1);
        t_max = (255/32)*(bin);
        if (red_array(x) <= t_max && red_array(x) >= t_min)
            red_hist(1,bin) = red_hist(1,bin) + 1;
        end
        if (green_array(x) <= t_max && green_array(x) >= t_min)
            green_hist(1,bin) = green_hist(1,bin) + 1;
        end
        if (blue_array(x) <= t_max && blue_array(x) >= t_min)
            blue_hist(1,bin) = blue_hist(1,bin) + 1;
        end
    end
end

%Concatenate histograms together and normalize
rgb_vector = cat(2,red_hist,green_hist,blue_hist);
```

```
rgb_hist = rgb_vector./sum(rgb_vector);


subplot(3,3,1)
imhist(red)
title('red imhist(testing)')
subplot(3,3,2)
imhist(green)
title('green imhist(testing)')
subplot(3,3,3)
imhist(blue)
title('blue imhist(testing)')

subplot(3,3,4)
x=(1:32)*8;
bar(x,red_hist)
xlim([0 255])
title('red')
subplot(3,3,5)
x=(1:32)*8;
bar(x,green_hist)
xlim([0 255])
title('green')
subplot(3,3,6)
x=(1:32)*8;
bar(x,blue_hist)
xlim([0 255])
title('blue')

subplot(3,3,8)
x=(1:96)*7.968;
bar(x,rgb_hist)
xlim([0 765])
title('RGB hist(FINAL ANSWER)')
xlabel('96 bins(32 each for R,G,B respectively)')
ylabel('normalized histogram')
```

Problem 4: Chroma Keying

To run:

```
>>chroma_keying('dog.jpg',[130,100,200],'beach.jpg')
>>chroma_keying('travolta.jpg',[130,100,200],'ocean.jpg')
```

Approach to solving this problem is as follows:

A function called “chroma_keying.m” was made for the purposes of this problem. It takes in an input image, mask parameters, and a background image

Image was read in, and separated into R,G,B layers respectively. The histograms for each layer were plotted and analyzed - this helps pick thresholding parameters for the mask

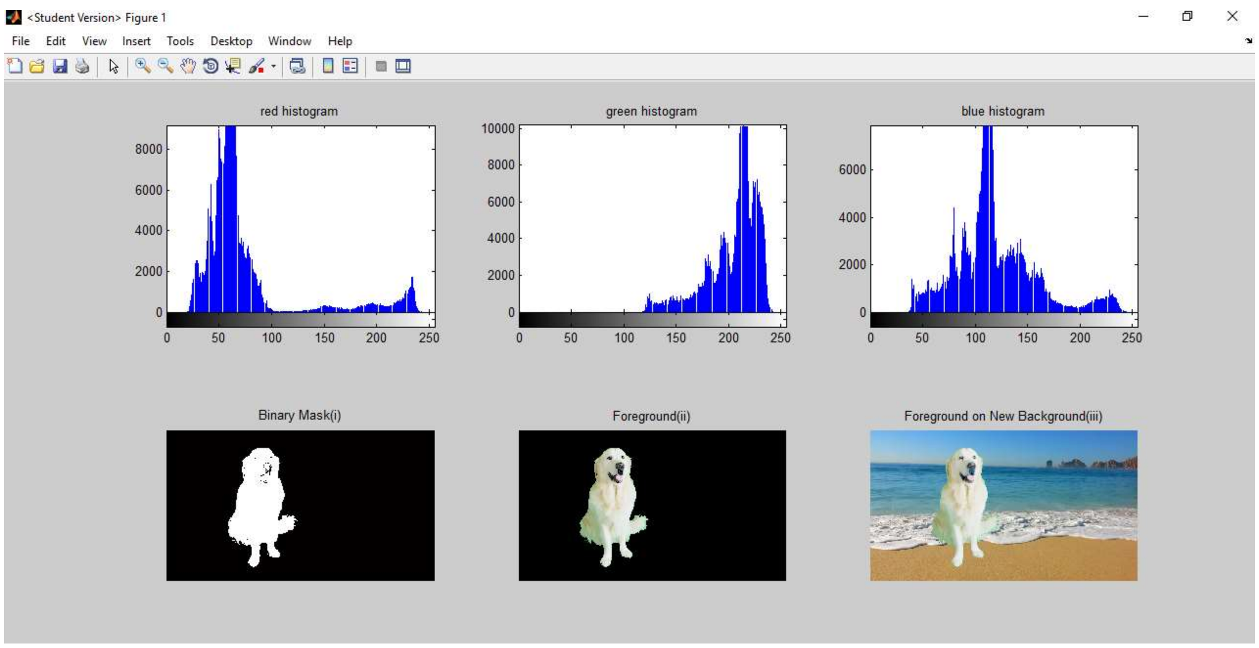
i) mask parameters are as so(for both images these parameters work decently):
   mask_params = [130,100,200]
   Thresholding happens as follows:

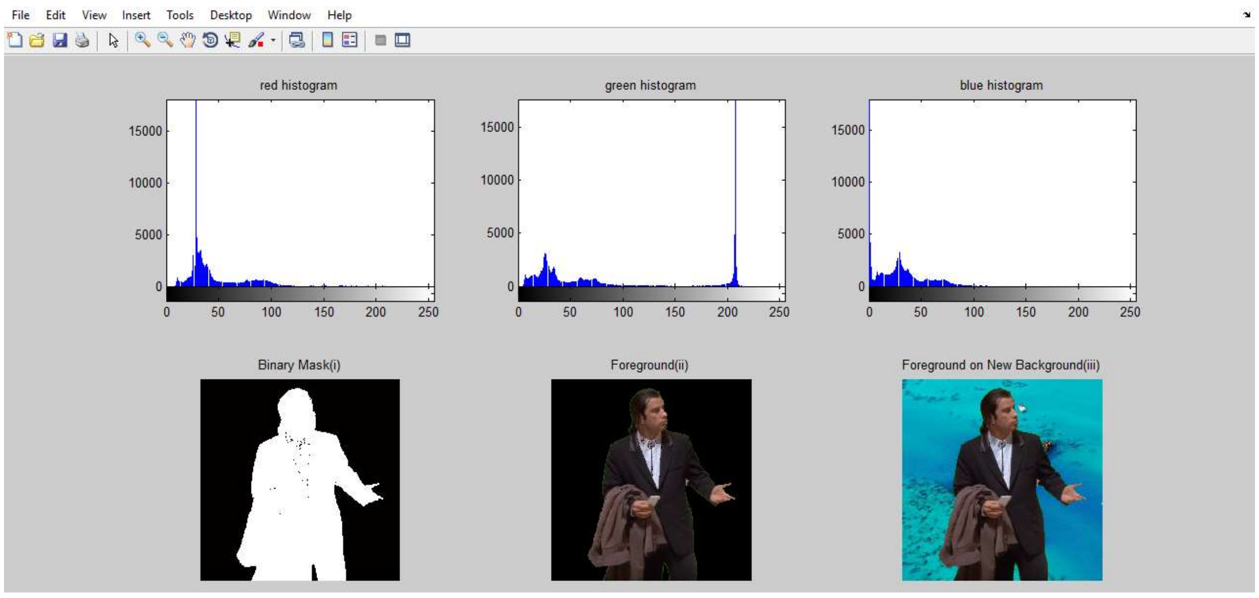
```
mask = (input_img(:,:,1) < mask_params(1)) & (input_img(:,:,2) > mask_params(2)) &
(input_img(:,:,3) < mask_params(3));
mask = imcomplement(mask);
binary_mask = uint8(mask);
```

Basically the goal is to mask out as much of the green background as possible.

ii)To show the original foreground image on a black background, each layer of the original image is point-wise multiplied with the mask created in i)

iii)Now to show the original foreground image on a new background, we pick a background that is the same size as each input image to keep things easy. ‘beach.jpg’ for chosen for the ‘dog.jpg’ input image, and ‘ocean.jpg’ was chosen for the ‘travolta.jpg’ input image, since the two had different sizes. Now these backgrounds are point-multiplied by the complement of the binary-mask - what this does is make the pixels where the original foreground will be placed, a value of “0”, or black. Now the original foreground can be added back in, and it is shown clearly on the selected background.


<Student Version> Figure 1
File Edit View Insert Tools Desktop Window Help
red histogram
green histogram
blue histogram
Binary Mask(i)
Foreground(ii)
Foreground on New Background(iii)


File Edit View Insert Tools Desktop Window Help
red histogram
green histogram
blue histogram
Binary Mask(i)
Foreground(ii)
Foreground on New Background(iii)

```
%Problem 4: Chroma Keying

function [] = chroma_keying(input_image, mask_params, background_image)

%First read in input and background images
input_img = imread(input_image);
img_background = imread(background_image);


%Process input image. Get histograms for individual RGB channels
%This way we can set thresholds for the mask

input_img_R = input_img(:,:,1);
input_img_G = input_img(:,:,2);
input_img_B = input_img(:,:,3);

%Binary image showing foreground mask
mask = (input_img(:,:,1) < mask_params(1)) & (input_img(:,:,2) > mask_params(2)) &
(input_img(:,:,3) < mask_params(3));
mask = imcomplement(mask);
binary_mask = uint8(mask);

%Use mask on original image, get RGB image on black background
original_foreground(:,:,1) = input_img(:,:,1).*binary_mask;
original_foreground(:,:,2) = input_img(:,:,2).*binary_mask;
original_foreground(:,:,3) = input_img(:,:,3).*binary_mask;

%Now place RGB image on another background
new_background(:,:,1) = original_foreground(:,:,1) + img_background(:,:,1).*(1-binary_mask);
new_background(:,:,2) = original_foreground(:,:,2) + img_background(:,:,2).*(1-binary_mask);
new_background(:,:,3) = original_foreground(:,:,3) + img_background(:,:,3).*(1-binary_mask);


subplot(2,3,1)
imhist(input_img_R)
title('red histogram')
subplot(2,3,2)
imhist(input_img_G)
title('green histogram')
subplot(2,3,3)
```

```
imhist(input_img_B)
title('blue histogram')
subplot(2,3,4)
imshow(mask)
title('Binary Mask(i)')
subplot(2,3,5)
imshow(original_foreground)
title('Foreground(ii)')
subplot(2,3,6)
imshow(new_background)
title('Foreground on New Background(iii)')
```