Computer Vision (600.461/600.661)
Homework 2: Color and Image Processing

Instructor: René Vidal

Due 09/23/2014, 11.59PM Eastern

1. **(20 points) Image filtering, enhancement, and edge detection**.

   (a) Study MATLAB functions `imread`, `brighten`, `contrast`, `histeq`, `imcontrast` and `imadjust`. Write a script `hw2q1a.m` that applies each one of these functions to the MATLAB image `peppers.png`. Plot the original and the transformed images and comment on what each function does [6pts].
   ANSWER:

```matlab
clear all;  % clear the workspace
close all;  % close all open figures
clc;        % clear the text in command window
% imread: Reads the image file and loads it into the memory
I = imread ('peppers.png');
I_gray = rgb2gray(I);
figure ('Name','Original Image');imshow(I);
figure ('Name','Gray Image'); imshow (I_gray);

% brighten: Depending on a positive or a negative argument, modifies the
% colormap and icreases or decreases the brightness of the image.
I_br = brighten (double(I_gray),0.5);
I_dr = brighten (double(I_gray),-0.5);
figure('Name','Brightened Image');imshow (I_br);
figure('Name','Darkened Image'); imshow (I_dr);

% Contrast: Returns an enhannced contrast image by creating a new colormap with ...
    an approximately equal intensity distribution
cmap = contrast(double(I_gray));
figure ('Name','Contrast Enhancement'); imshow (I_gray); colormap(cmap);

% histeq: Histogram equalization changes the contrast by changing the
% histo such that more colors are visible
I_histeq = histeq(I_gray);
figure ('Name','Histogram Equalization');imshow (I_histeq);

% imcontrast: Manual contrast adjustment tool by changing the lower and
% higher bounds of the histogram
figure ('Name','Adjusted Contrast Manually'); imshow (I_gray);
imcontrast (gca)

% imadjust: Increases the contrast by saturating 1% of data at lower and higher
% intensity bounds
figure('Name','imadjust'); imshow (imadjust(I_gray));
```

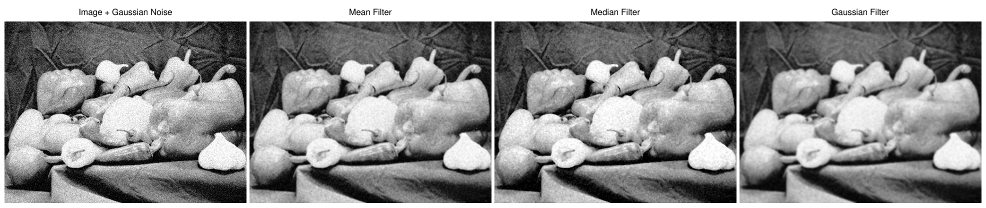
   TA comments: 0.5 points (each) were deducted for the following reasongs.

   - missing that brighten can also darken the image with negative parameter input.
   - using rgb2ind in contrast example. The example in matlab uses an indexed image but the ouput is clear and we can see the difference between the grayscale image and image output by contrast. If the output of contrast is not clear and it is difficult to see difference from grayscale, I've deducted points.
   - imadjust can be used with parameters [low_in, high_in] and [low_out,high_out] or without any parameters. I've deducted points if first format was used but clipping wasn't mentioned for the pixels that lie outside the range [low_in, high_in]

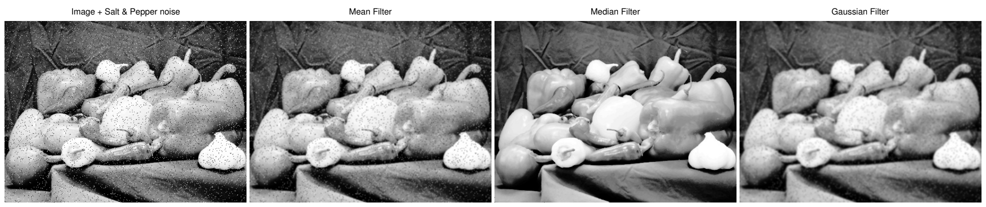
   (b) Study the functions `imnoise`, `medfilt2`, `conv2`, `filter2`, `fspecial`, `imfilter`, and `edge`. Write a script `hw2q2b.m` that does the following. Load image `peppers.png`. Convert it from RGB

to grayscale using the function `rgb2gray`. Add salt and pepper noise to the image. Filter the resulting image using a 3x3 mean filter, a 3x3 median filter, and Gaussian filter with $\sigma = 1.5$ pixels. Repeat, but this time add Gaussian noise with $\sigma = 1$ in the $[0, 255]$ range ($\sigma = 1/256$ in the $[0, 1]$ range) instead of salt and pepper noise. Plot each one of the images and comment on what works best [7pts].
ANSWER:

```
clear all;  clc; close all;
% read the image
img=imread('peppers.png');
%make grayscale
imgGray=rgb2gray(img);
% %equalize
% imgEq=histeq(imgGray);
imgEq=im2double(imgGray);

%add salt&pepper noise
imgNoiseSP=imnoise(imgEq,'salt & pepper');

%make filters' masks
hAvg=fspecial('average',3);
hGauss=fspecial('gaussian',9,1.5);  %filter support about 6*sigma

%filter image
imgNoiseSPAvg=imfilter(imgNoiseSP,hAvg,'symmetric');
imgNoiseSPMed=medfilt2(imgNoiseSP);
imgNoiseSPGauss=imfilter(imgNoiseSP,hGauss,'symmetric');

fprintf('MSE for Salt and Pepper Noise \n Avg: %e \t Med: %e \t Gauss: %e \n',...
    mean((imgNoiseSPAvg(:)-imgEq(:)).^2), mean((imgNoiseSPMed(:)-imgEq(:)).^2),...
    mean((imgNoiseSPGauss(:)-imgEq(:)).^2));

% display output
figure('Name','Image with Salt and Pepper Noise denoised using filters');
subplot(221), imagesc(imgNoiseSP); axis('image'); axis('off'); colormap('gray'); ...
    title('noisy image');
subplot(222), imagesc(imgNoiseSPAvg); axis('image'); axis('off'); ...
    colormap('gray'); title('avg. filter');
subplot(223), imagesc(imgNoiseSPMed); axis('image'); axis('off'); ...
    colormap('gray'); title('median filter');
subplot(224), imagesc(imgNoiseSPGauss); axis('image'); axis('off'); ...
    colormap('gray'); title('gauss filter');

%add gaussian noise
imgNoiseGauss=imnoise(imgEq,'gaussian',0,1/255);

%filter image
imgNoiseGaussAvg=imfilter(imgNoiseGauss,hAvg,'symmetric');
imgNoiseGaussMed=medfilt2(imgNoiseGauss);
imgNoiseGaussGauss=imfilter(imgNoiseGauss,hGauss,'symmetric');

fprintf('MSE for Gaussian Noise \n Avg: %e \t Med: %e \t Gauss: %e \n',...
    mean((imgNoiseGaussAvg(:)-imgEq(:)).^2), ...
        mean((imgNoiseGaussMed(:)-imgEq(:)).^2),...
    mean((imgNoiseGaussGauss(:)-imgEq(:)).^2));

% display output
figure('Name','Image with Gaussian Noise denoised using filters');
subplot(221), imagesc(imgNoiseGauss); axis('image'); axis('off'); ...
    colormap('gray'); title('noisy image');
subplot(222), imagesc(imgNoiseGaussAvg); axis('image'); axis('off'); ...
    colormap('gray'); title('avg. filter');
subplot(223), imagesc(imgNoiseGaussMed); axis('image'); axis('off'); ...
    colormap('gray'); title('median filter');
subplot(224), imagesc(imgNoiseGaussGauss); axis('image'); axis('off'); ...
    colormap('gray'); title('gauss filter');
```



The Gaussian filter has the best performance in term of the noise reduction. However, the resulting image is not as sharp as the other two filters.



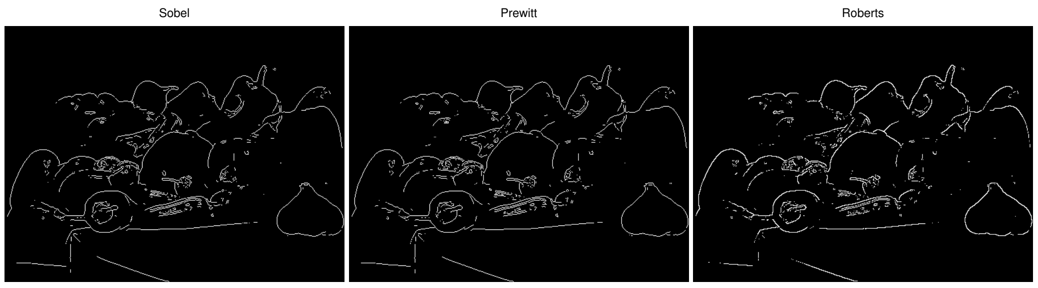
The median filter is the best among all three filters.

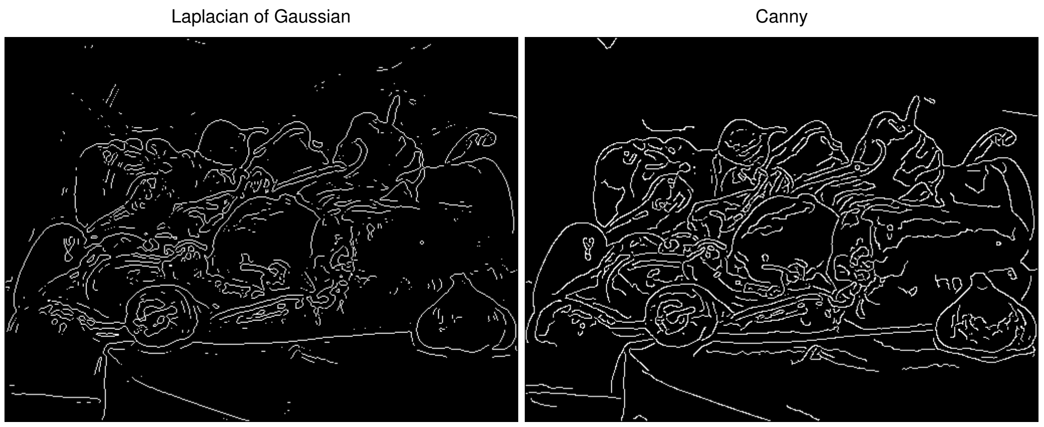
TA comments: Visually, it is difficult to see the difference between denoised images for gaussian noise under different filters. However, if you compute the Mean Square Error (mean residual between denoised image and original image), you will see that the gaussian filtering works best for gaussian noise. I've deducted 0.5 points for other answers.

(c) Write a script `hw2q1c.m` that does the following. Load image `peppers.png`. Convert it from RGB to grayscale using the function `rgb2gray`. Find the edges in the image using the MATLAB function `edge`. Use the following methods: Sobel, Prewitt, Roberts, Laplacian of Gaussian and Canny. Compare your results [7pts].

ANSWER:

```
clear all;  clc; close all;
% read the image
img=imread('peppers.png');
%make grayscale
imgGray=rgb2gray(img);
%extract edges with all the methods
edgesSobel=edge(imgGray,'sobel');
edgesPrewitt=edge(imgGray,'prewitt');
edgesRoberts=edge(imgGray,'roberts');
edgesLog=edge(imgGray,'log');
edgesCanny=edge(imgGray,'canny');

% display output
figure;
subplot(231), imshow(imgGray); title('grayscale image');
subplot(232), imshow(edgesSobel); title('sobel edges');
subplot(233), imshow(edgesPrewitt); title('prewitt edges');
subplot(234), imshow(edgesRoberts); title('roberts edges');
subplot(235), imshow(edgesLog); title('LoG edges');
subplot(236), imshow(edgesCanny); title('Canny edges');
```





The results for the Sobel, Prewitt and Roberts methods are similar and only the strongest edges (e.g. the boundary between the objects) are found. The LOG filter finds more edges but gives also more spurious edges (i.e. edges a few pixels long). However, the Canny edge detector gives results similar to the LOG detector. The edges are thin (i.e., well localized), spurious edges are absent and the detector could find also weaker edges, e.g. the change of intensity given by the specular reflections on the surface of the peppers.

TA comments: A lot of students have simply said that Canny edge detector is the best and the order of goodness is Canny $>$ LoG $>$ Sobel=Prewitt=Roberts. However, detecting a large number of edges is not the only criteria for a good edge detector. I expected you to discuss the kinds of edges detected by the different detectors and your observations.

2. **(30 points) Color-based face detection.** One way to detect faces in color images is to search for pixels that have a skin-like color. The figure below shows an example where clusters t_4, t_5 and t_6 of the normalized RGB color space represent primary and secondary face colors. In this exercise, you will implement this simple color-based faced detection algorithm.

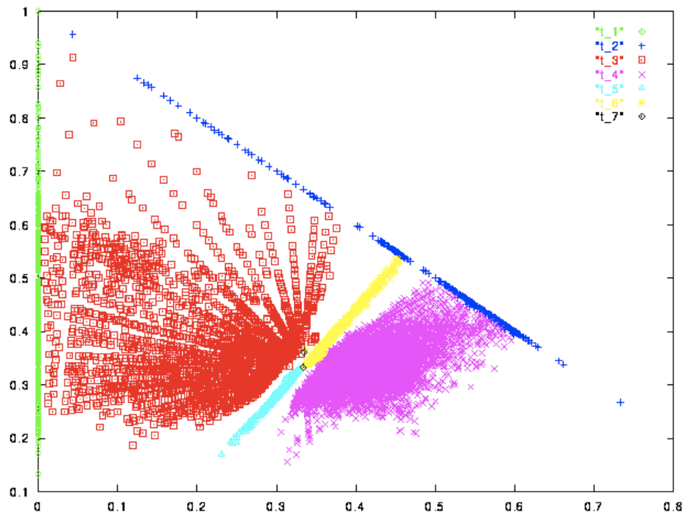


Figure 6.12: Skin color clusters obtained from training: the horizontal axis is R_{norm} and the vertical axis is G_{norm}. The cluster labeled as t_4 is the primary face color, clusters t_5 and t_6 are secondary face clusters associated with shadowed or bearded areas of a face. (Figure from V. Bakic.)

(a) Study the following MATLAB functions `colormap`, `hsv2rgb`, `rgb2gray`, `rgb2hsv`, `rgb2ntsc`, and `rgb2ycbcr`, and write a sentence or two about what each function does [5pts].

ANSWER:

i. `colormap`: Sets and gets the current colormap. In the latest version of MATLAB, the `jet` colormap is used as default.

ii. `hsv2rgb`: Converts a colormap (or image) from HSV (hue-saturation-value) coordinates to RGB (red-green-blue) coordinates.

iii. `rgb2gray`: Converts RGB ($m \times n \times 3$) images to gray-scale images ($m \times n$). In the background, it converts the image to HSV, and then eliminates the hue and saturation.

iv. `rgb2hsv`: Converts the RGB image to HSV. It is the complement function of `hsv2rgb`.

v. `rgb2ntsc`: Convert RGB color values to YIQ color space. The Y component represents the luma information. I and Q represent the chrominance information.

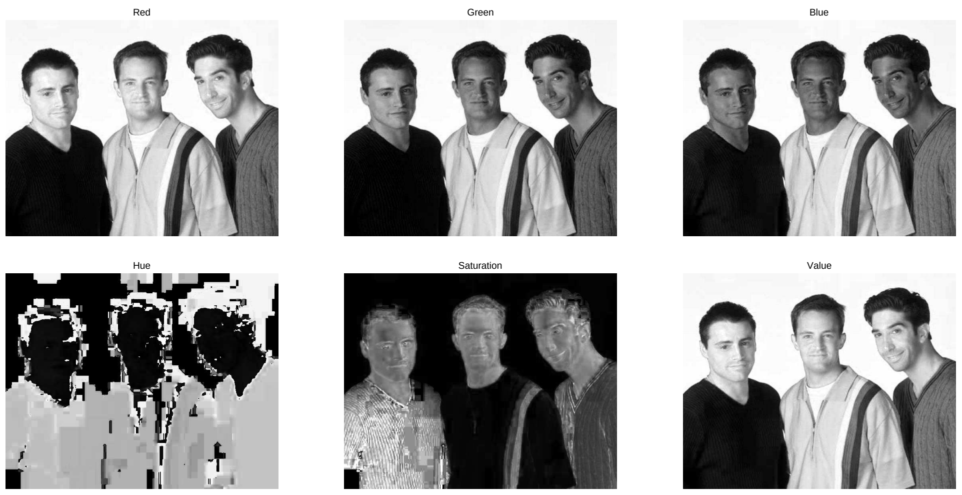
vi. `rgb2ycbcr`: Convert RGB color values to YCbCr color space. The output image is a $M \times 3$ matrix that contains the YCbCr luminance(Y) and chrominance (Cb and Cr) color values as columns.

TA comments: I've deducted 0.5 points if you have not expanded what HSV, NTSV, YIQ, YCbCr are. I have written 'What are HSV, YIQ, YCbCr color spaces ?' as a comment in the graded sheets. I expected you to mention HSV - Hue-Saturation-Value color space. YIQ has luminance (Y) and chrominance components (I,Q) and the same for YCbCr (luminance - Y and chrominance Cb, Cr). I wanted to make sure that you read up a little more than the first line in the matlab documentation.

(b) Download the image `http://www.allthetests.com/quiz19/picture/pic_1151313471_10.jpg`. Convert the RGB coordinates to HSV coordinates and plot the following grayscale images: R, G, B, H, S, V. Comment on what you see, e.g. in which regions of which image skin color is more visible [4pts].

ANSWER:

Since the skin color is constructed from more red components, therefore it is shown with higher intensities in the Red image and with lower intensities (darker) in the Green and the Blue images. The face color is darker in the Blue image compared to the Green image (green has a wider range of wavelength in the spectrum). In the Hue component of the image, the skin color has very low intensities shown in relatively dark areas. The Hue component corresponds the lower bound to red and yellow, and the higher values to blue and violet. Therefore, the skin color with more red components are shown as dark areas. Saturation describes the amount of white color in the image. As expected, the skin color has some white components, thus it is represented as gray. Lastly, the Value component is the gray scale image where the skin color is shown with relatively bright colors, but darker than the background.



TA comments: Using intensity value, V has the highest amongst H,S,V for skin pixels. However, there is no information from that. On the other hand, H is very low for skin pixels and backgroud. S is somewhat high for the clothes and skin pixels. Hence, both carry relevant information that can help detect skin pixels and 0.5 points (each) were deducted for missing this. Some discussion of the skin pixels in R,G,B was also expected.

(c) Convert the RGB coordinates to normalized rgb coordinates. Plot g versus r as in Figure 6.12. Find a region of the r-g space that corresponds to the face colors. That is, find a function f (maybe linear as shown in Figure 6.12) such that face pixels can be determined by a rule of the form

$$\text{pixel } (x, y) \text{ is a face pixel if } f(r(x, y), g(x, y)) \geq 0. \tag{1}$$

You can do this by trial and error (we will learn automatic methods later in the course). For instance, choose a function f, define the mask

$$M(x, y) = \begin{cases} 1 & \text{if } f(r(x, y), g(x, y)) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

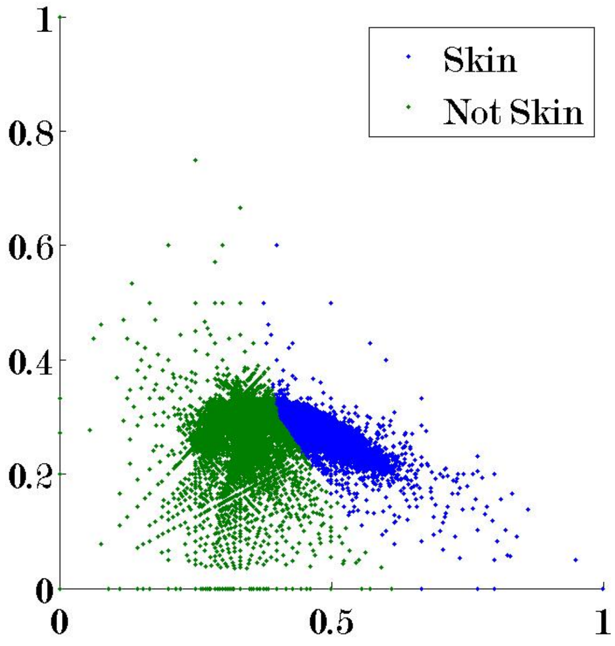
and plot the image $J(x, y) = I(x, y)M(x, y)$. Repeat this until faces are "correctly" detected in $J(x, y)$ [8pts].

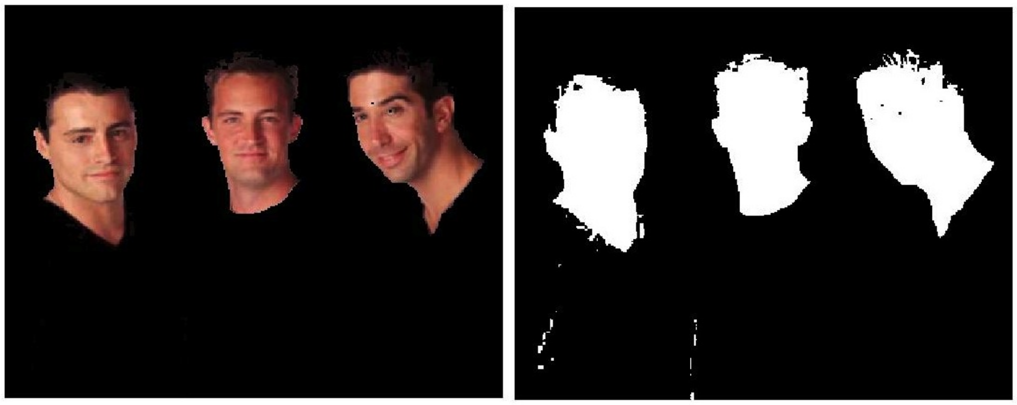
ANSWER:

```
% read the image
R = imread('pic_1151313471_10.jpg');
% [N,M,¬] = size(R);
% n = floor(N/2);
% m = floor(M/2);
% A = double(imresize(R,[n,m]));
A = double(R);

% get the R,G,B channels of the image
Ar = A(:,:,1);
Ag = A(:,:,2);
Ab = A(:,:,3);

% do the normalization
Arc = Ar./(Ar+Ag+Ab+1e-10); %1e-10 is to avoid 0/0
Agc = Ag./(Ar+Ag+Ab+1e-10);
Abc = Ab./(Ar+Ag+Ab+1e-10);


% Test your decision function
idx1 = double(Agc≥(-6.71*Arc+2.98)); %y ≥ -6.71x + 2.98
idx2 = double(Agc≥(-1.33*Arc+0.83));%y ≥ -1.33x + 0.83
% create mask
mask = idx1.*idx2;

% apply mask to image to show skin regions
B = zeros(size(A));
B(:,:,1) = mask.*Ar;
B(:,:,2) = mask.*Ag;
B(:,:,3) = mask.*Ab;
B = uint8(B);

% display outputs
figure ('Name','Segmented Skin - RGB'); imshow(B);
figure('Name','Mask - RGB'); imshow (mask);

% r-g plot with skin pixels in blue and others in red
figure; hold on;
plot(Arc(:).*(1-mask(:)),Agc(:).*(1-mask(:)),'r+');
plot(Arc(:).*mask(:),Agc(:).*mask(:),'b*');
xlabel('r'); ylabel('g'); title('r-g plot of pixels');
legend('non-skin','skin');
```

<u>TA comments:</u> Finding equations to detect the skin pixels in RGB color space is based on trial-and-error. However, I wanted to reward the students who tried more complicated functions and/or got better results. Hence, the deduction of 0.5 points with a comment 'you can do better' for the others. If I were solving this

problem, I would solve (d) first (as it is easier to find bounds that a function) and then visualize the skin vs non-skin pixels to see how the boundary looks like. I have deducted points if you did wrong normalization.

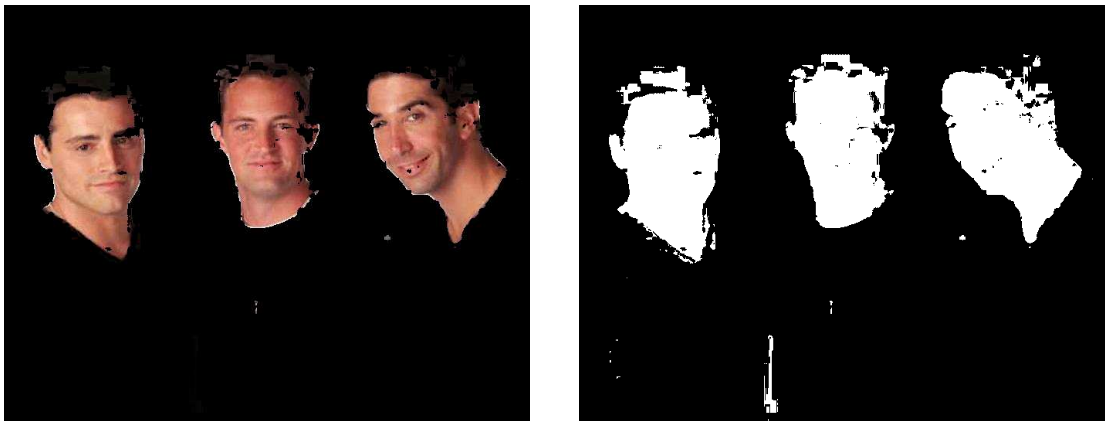




(d) Repeat part 2c using the hue and saturation coordinates of the HSV color representation. If $H(x,y)$ and $S(x,y)$ are respectively the hue and saturation at pixel (x,y), then the rules to determine a face pixel may be of the form $a \leq H(x,y) \leq b$ and $c \leq S(x,y) \leq d$ [8pts].

ANSWER:

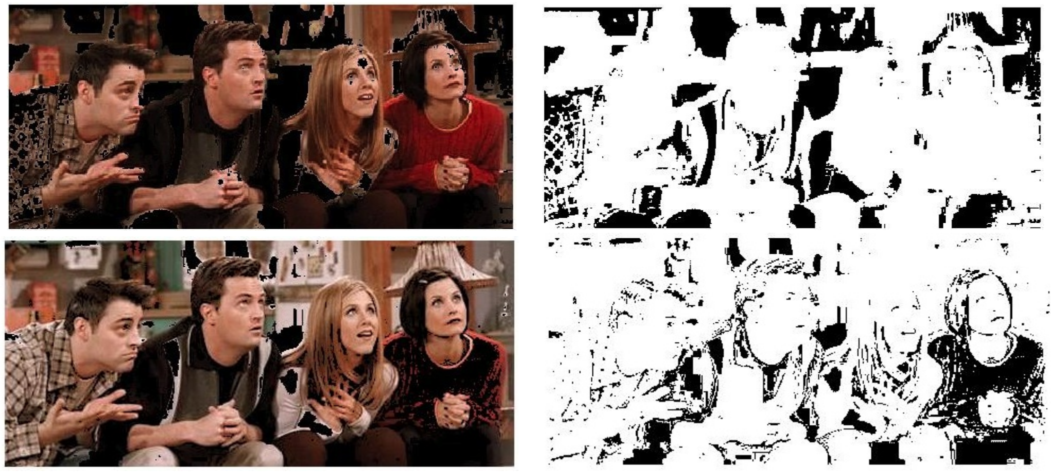
```
clear all; clc; close all;
% read the image
R = imread('pic_1151313471_10.jpg');
% convert into hsv image
A = rgb2hsv (R);
[n,m,¬] = size(A);
% get the H,S,V components
Ah = A(:,:,1);
As = A(:,:,2);
Av = A(:,:,3);
% decision function
a=0; b=0.50; c=0.10; d=0.80;
% generate mask
mask = (Ah ≥ a) & (Ah ≤ b) & (As ≥ c) & (As ≤ d);
mask = uint8 (mask);
% use mask to generate new image
R_seg(:,:,1) = R(:,:,1) .* mask;
R_seg(:,:,2) = R(:,:,2) .* mask;
R_seg(:,:,3) = R(:,:,3) .* mask;
R_seg = cat(3, R_seg(:,:,1), R_seg(:,:,2), R_seg(:,:,3));
% display results
figure ('Name','Segmented Skin - HSV'); imshow(R_seg);
figure('Name','Mask - HSV'); imshow (mask>0);
% h-s plot with skin pixels in blue and others in red
mask=double(mask);
figure; hold on;
plot(Ah(:).*(1-mask(:)),As(:).*(1-mask(:)),'r+');
plot(Ah(:).*mask(:),As(:).*mask(:),'b*');
xlabel('r'); ylabel('g'); title('h-s plot of pixels');
legend('non-skin','skin');
```



(e) Apply the rules you learned in parts 2c and 2d to the test image http://i168.photobucket.com/albums/u196/thesixfriends/friends45698.jpg. Which color representation works better and why [5pts]?

ANSWER:

The top figure is segmented in the RGB space and the bottom is the segmentation in the HSV space. It is clear that neither of the methods work properly with the predefined learned parameters. The reason lies on the change of illumination from one image to the next. However, there is a preference for using HSV over RGB which provides easier representation to locate the thresholds. For more studies please refer to http://www.cs.rutgers.edu/~elgammal/pub/skin.pdf.



,