CSE152A – Computer Vision – Assignment 2 (SP15)
Instructor: Ben Ochoa
Maximum Points : 65, Maximum Bonus: 5
Deadline : 11:59 p.m., Tuesday, 12-May-2015

Instructions:

- This assignment should be solved, and written up in groups of 3.
- Individual work is not allowed.
- There is no physical hand-in for this assignment.
- Coding for this assignment should be done in `MATLAB`
- All code developed for this assignment should be included in the appendix of the report.
- You may do problems on pen and paper; just scan and include it in the report.
- In general, `MATLAB` code does not have to be efficient. Focus on clarity, correctness and function here, and we can worry about speed in another course.
- Submit your assignment electronically by email to Akshat Dave `[akdave@ucsd.edu]` with the subject line *CSE152A-Assignment-2*. The email should contain one attached file named [`CSE_152A_HW2_<student1-id>_<student2-id>_<student3-id>.zip`]. This zip file must contain the following two artifacts:
  1. A pdf file named [`CSE_152A_HW2_<student1-id>_<student2-id>_<student3-id>.pdf`] containing your writeup. Please mention all the authors' full names and student identities in the report.
  2. A folder named [`CSE_152A_HW2_<student1-id>_<student2-id>_<student3-id>_code`] containing all your MATLAB code files

1 Colors [10 points]

Write a MATLAB function $I_{out} = \texttt{TransformHSV}(I_{in}, \theta, \sigma, \nu)$ to manipulate the color of an image in `HSV` color space [2 points]. The function should do the following:

- Convert the image from RGB to HSV color space
- Apply a clockwise rotation of the hue channel by θ degrees
- Multiply the saturation channel by σ
- Multiply the value channel by ν
- Convert the transformed HSV image back into RGB color space

Do not use the built-in `rgb2hsv.m` or `hsv2rgb.m` functions in MATLAB, except to compare your results. Test your function on the image `bat_rob.jpg`, and generate results for thefollowing combinations of parameters:

- $\theta = 0^\circ,\ \sigma = 1,\ \nu = 1$ [2 points]
- $\theta = 60^\circ,\ \sigma = 1,\ \nu = 1$ [2 points]
- $\theta = 0^\circ,\ \sigma = 0.5,\ \nu = 1$ [2 points]
- $\theta = 0^\circ,\ \sigma = 1,\ \nu = 0.5$ [2 points]

Include the generated images in your writeup as a 2×2 grid using `subplot.m`.

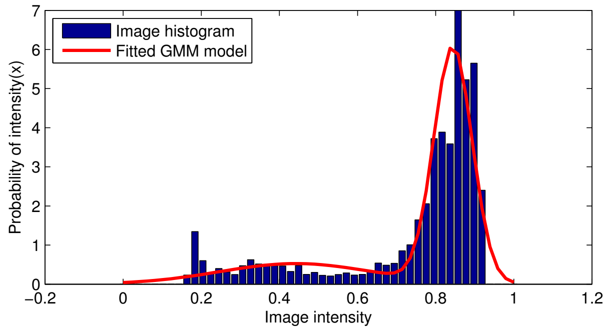

Figure 1: Fitted Gaussian Mixture Model to a sample image histogram

2 Binarization [10 points]

Write a `MATLAB` function to implement the "peakiness" detection algorithm described in class (CSE152-notes-link). This algorithm should automatically determine an intensity level to threshold an image to segment out the foreground from the background. The output of your function should be a binary image that is 0 for all background pixels and 1 for all foreground pixels. Apply this function to the image `can_mod.jpg` and turn in the output image in your report.

Notes:

- Load in an image and convert it to grayscale using `rgb2gray.m`.
- Normalize the image and convert it to double precision using `im2double.m`
- You can use the `MATLAB` function `hist.m` to create a histogram of pixel intensities as a first step to the peakiness detection algorithm. Using 15 bins and a minimum distance of 3 bins between peaks should work.

3 BONUS : Gaussian Mixture Models [5 points]

Instead of the peakiness algorithm, try fitting a Gaussian Mixture Model (GMM) to the grayscale image. GMMs are useful in approximating multimodal distributions. Let one of the modes represent the foreground and the other the background. You can use the `MATLAB` function `gmdistribution.fit` to fit the model to the data. Include the image histogram along with the fitted distribution overlaid, as shown in fig. 1. Note that it will suffice to use a 2 mixture model. Once you have fitted your distribution, a segmentation can be achieved by

$$B(x, y) = \arg\max_c P(I(x, y)|c)P(c) \tag{1}$$

where $P(I(x, y)|c)$ is the probability of intensity at pixel (x, y) for class $c \in \{0, 1\}$, and $P(c)$ is the (prior) probability of class c, and $B(x, y)$ is the output binary image. State your assumptions (if any) regarding the class priors. Compare your results using this method to the peakiness method above.

In addition to the class slides, some more details on gaussian mixture models can be found in the lecture notes (ECE271-notes-link) slides, which also contain some information about the EM-algorithm used by `gmdistribution.fit` to fit the model.

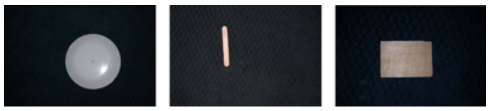

Figure 2: Sample images

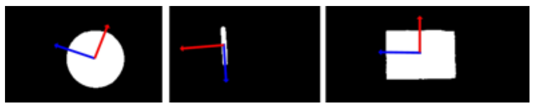

Figure 3: Sample images with principle directions

4 Connected Components [15 points]

(a) Write `MATLAB` code to implement the connected component labeling algorithm discussed in class, assuming 8-connectedness. Your function should take as input a binary image (computed using your algorithm from question 1) and output a 2D matrix of the same size where each connected region is marked with a distinct positive number (e.g. 1, 2, 3). On the image `can_mod.jpg`, display an image of detected connected components where connected region is mapped to a distinct color (using the function `imagesc` in `MATLAB`). You may need to modify the `MATLAB` recursion limits using `set`(0,'RecursionLimit',1000)

(b) How many components do you think are in the image `coins.png`? What does your connected component algorithm give as output for this image? Include your output on this image in your report. it may help to reduce the size of the image before running the connected components algorithm but after converting the image to a binary image `imresize(binary, 0.25, 'bilinear')`

5 Take your own images [5 points]

For this part, you will be using images you take with your own camera. Choose three objects of different shapes, preferably with non-shiny surfaces. Possibilities include paper or cardboard cut-outs, bottle tops, pencils, Lego pieces, etc. Take 3 pictures of these objects individually with a solid background. The object should be clearly distinguishable from the background (e.g. bright object & dark background). Include these three images in your report as in Figure 2. In addition, show similar output images as in Problem 4 for each of your new images (there is only 1 connected component in this case).

6 Image moments and rectification [10 points]

Write three functions which compute the moments, central moments, and normalized moments of a marked region (CSE152-notes). Each function should take as input a 2D matrix (output of part 2) and 3 numbers j, k, and d. The output should be the (j,k) moment $M_{j,k}$, central moment $\mu_{j,k}$, normalized moment $m_{j,k}$ of the region marked with positive number d in the input matrix.

Using these functions, on each of the three images, draw the centroid of each object. Also compute the eigenvectors of the centralized second moment matrix (CSE152-notes) and draw the two eigenvectors on the centroid. This should indicate the orientation of each object. See Figure 3 for the results on the example images. Turn in the outputs for your own images.

7 Image alignment [10 points]

The orientation computed from Problem 6 can be used to roughly align the orientation of the region (i.e. in-plane rotation). Write a function to rotate the region around its centroid so that the eigenvector corresponding to the largest eigenvalue (i.e. the blue vector in Figure 3) will be horizontal (aligned with $[1,0]^T$). This might look like in Figure 4.

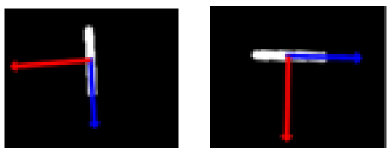
(a) (b)

Figure 4: (a) Original binarized image with principal directions (b) Aliged sample image with major eigenvector aligned along $[1,0]^T$

Your function should take as input a 2D matrix (output of Problem 4) and output a matrix of the same size in which all marked regions are aligned. Turn in the aligned outputs for your images.

Note: After finding the rotation matrix R to rotate the largest eigenvector to $[1,0]^T$, we rotate all points (x,y) belonging to that region using the following transformation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R \begin{bmatrix} x - \hat{x} \\ y - \hat{y} \end{bmatrix} + \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix}$$

where $[\hat{x},\hat{y}]^T$ are the centroid coordinates of the region For simplicity, just ignore the cases when part of the aligned region falls outside of the image border or is overlapped with other regions. You can avoid these issues when capturing your images (e.g. put your objects a bit far apart). Finally, note that the rotation matrix can be created trivially from the eigenvectors.