**Project 2: Grey-level image processing, Feature Detection, and Local Feature Descriptor**

ESE 568 Computer Vision, SBU, ECE, SUNY at Stony Brook, Prof. Murali Subbarao
(10 points)

**Draft 1.0**

Given an input jpeg color image, convert it to an intensity or gray-level image by averaging the R, G, and B components. On the resulting gray level image, perform the following operations and write the output as a TIFF format image. Display the output and inspect it. All computations must be performed using the intensity data at the pixels. You are not permitted to use built in library functions of Matlab to perform filtering, edge detection, histogram computation, etc. Computations must be done explicitly by using gray-level values at each pixel.

1. (1 points) ***Histogram Equalization:*** Transform the input image so that the output image has an "equalized" histogram. Use the Cumulative Distribution Function of the histogram of the input image as the intensity transformation function.

   **Notes:** If h(k) is the histogram array for k=0,1,2,…,255, of an input image f(m,n) with size M*N, then the discrete probability for brightness k is p(k)=h(k)/(M*N). The cumulative distribution function c(k) is

   $c(k) = \sum_{i=0}^{k} p(i)$. The intensity/brightness transformation at a pixel (m,n) is

   $h(m,n) = 255 * c(f(m,n))$ where f is the input image and h is the output image.

2. (2 points**) Image Flitering/Convolution:** Implement convolution with an arbitrary 5x5 filter g(i,j). Read the 5x5 filter coefficients of g(i,j) from a file. Compute the output image h(i,j) only for its interior pixels which are more than 2 pixels inside from the border. Set the other pixels close to the border of h(i,j) to zero. Use the following filters to test your program:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | -1 | 0 | 0 | | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0 | -1 | -2 | -1 | 0 | | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| -1 | -2 | 16 | -2 | -1 | | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0 | -1 | -2 | -1 | 0 | | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0 | - 0 | -1 | 0 | 0 | | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |

For input image f(i,j) and filter g(k,l), the output image h(i,j) in the interior pixels is given by:

$$h(i,j) = \sum_{k=1}^{5} \sum_{l=1}^{5} g(k,l) * f(i-(k-3),\ j-(l-3))$$

where 5x5 is the filter size. In the above equation, i,j=3,4,5, …, N-2, where NxN is the size of the input and output images.

Compute the filtered image only in the interior regions with row/column indices in the range 3 to N-2 for an NxN image. Assign zeros to two columns/rows all along the 4 borders of the output image. In order to display the output image (which requires the pixel values to be in the range 0 to 255), rescale the intensity values in the output h(i,j) as follows. Compute the minimum and maximum of h(i,j) to be hmin and hmax. Then compute the output image as r(i,j)= (h(i,j)-hmin)*255/(hmax-hmin).
Then display the output image r(i,j).

3. (3 points) **Edge detection.** First smooth the input image (size NxN=100x100) with a 9x9 **separable** Gaussian filter. Then compute the gradient magnitude and threshold it to mark edge pixels. Test your program by detecting edges for different sigma (1.0 to 3.0) and different thresholds (20 to 50). Make sure that the one-dimensional Gaussian filter is normalized so that the sum of the 9 coefficients is 1.0.

   **Notes:** You should compute the one-dimensional Gaussian filter using the standard equation with sigma as a parameter in the range 0.5 to 3. After computing the 9 coefficients of the filter, normalize the filter coefficients by dividing each of them with the sum of all the nine coefficients. After normalization, the new coefficients will sum to 1.0 so that the average brightness of a smoothed image remains the same as the original image. Smooth only the interior pixels between 5th row, 5th column, (N-5)th row and (N-5)th column. Leave the other pixels as they were in the original image.

   First smooth the image along rows with the 1D Gaussian filter. The result of this step should then be smoothed along columns to obtain the final result. After this, compute the gradient magnitude and threshold it with a value in the range 10 to 40. If the gradient magnitude is higher than the threshold at a pixel (i,j), then set the value of the pixel in the output image to be 255 (maximum brightness). Compute edges in interior pixels between rows/columns 2 to 99 only, and set the other border pixels to be 0.

4. **Corner detection and local feature descriptor** (2+2 points):
   (a) **Corner detection:** Modify the program for edge detection above to implement the Harris-corner detector. Take the window size for computing M to be 11x11 with w(x,y) to be a Gaussian with sigma=5.5 pixels. Find a suitable threshold for R by trying different values. Your output should be the original image superimposed with the detected corner points shown with a 3x3 region shown with pixel values 255.

   For corner detection, try the following algorithm:

   1. Smooth the image with a 9x9 separable gaussian with sigma=2 pixels.
   2. Compute the gradient vector (Ix,Iy) at each pixel.
   3. Compute A=Ix^2, B=Iy^2, and C= Ix * Iy at each pixel.

4. Smooth A, B, and C, with an 11x11 gaussian with sigma= 11/2= 5.5 pixels, to get A', B', and C'.
5. At each pixel, M = ( A' , C' ,

C' B' ).

6. At each pixel, Compute R= det(M)-0.04* (trace(M))^2
7. At each pixel, if R>threshold, mark it as a corner point. You may have to

try different values of threshold before you find a good threshold. If one of you

find a good threshold value, post it on piazza.com. Also, note that, the threshold

may be large if your original image pixel values are in the range 0 to 255, and the threshold will be small if the pixel values or initially scaled to be in the range 0.0 to 1.0 double.

8. Non-maxima suppression: After thresholding R, at each pixel, compare the value of R at that pixel with the value of R at all the 8 neighboring pixels. If the current pixel has a value of R that is higher than all its' 8 neighbors, then mark it as a corner pixel. Otherwise, mark it as a non-corner pixel.

(b) **Local feature descriptor:** At each corner pixel, consider a 9x9 subimage with the corner pixel at the center. Compute a "normalized-gradient-direction histogram" as follows: Let the histogram's bins represent gradient directions quantized to 0,45,90,135,180,225, 270, and 315, degrees. For each pixel, find the direction of its gradient vector, quantize it to one of the 8 angles above, and increment the count of the corresponding histogram entry by the amount equal to the magnitude of the gradient. After this, normalize the histogram by "rotating the histogram modulo 360 degrees", so that the histogram entry with the highest value corresponds to 180 degrees. If the m-th entry is the maximum, for some m in 1 to 8, then you can obtain a normalized histogram hn(n) with the m-th entry brought to position k=5 (corresponding to 180 degrees) from the original histogram h(m) as

```
hn=zeros(1,8); %normalized histogram
h=zeros(1,8);   %original histogram
% compute original histogram  h()
% Rotate h()circularly to obtain hn() so that
% the maximum of h() at h(m) moves to hn(5)
k=5;
for i= 1 : 8
  hn(mod(k-2+i,8)+1) = h(mod(m-2+i,8)+1);
end
```

(c) Print the pixel coordinates of all the corner pixels followed by its associated local histogram (its local feature descriptor).

5. **Image matching**: Take two gray-level images of an object with an overlapping region, but one of them is translated, rotated, and scaled (affine transformed) with respect to the other. Apply the corner detector and local feature descriptor to both of them. Suppose there are M corner points P1 to Pm in the first image, and N corner points Q1 to Qn in the second image. Find at least 15 matching points either manually, or by matching the local feature descriptors in one image to the closest ones in the other. Given K matching points (K>=15), use one the linear least squares algorithm to find the affine transformation matrices.
6. **Panorama stitching** : Optional: Use the transformation parameters found in step 5 to stitch the two images together.

5. Linear Least Squares Fit

- closed form solution
- robust to noise
- not robust to outliers

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Identify the input, output, and the computational steps.

Suppose that $(x_i, y_i)$ are corner points in the first image that respectively match $(x_i', y_i')$ in the second image for i= 1 2, . . . N. Then formulate the linear system of equations:

A$\mathbf{x}$=$\mathbf{b}$

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ & & \vdots & & & \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ \vdots \end{bmatrix}$$

And obtain the solutions for the parameters a,b, . . . , f, as:

$$\mathbf{x} = [\mathbf{A}^{\mathrm{T}}\mathbf{A}]^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$$ (linear least squares solution)

6. Panorama stitching: (optional): Transform the corners of the second image to the first image using:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This gives the coordinates (x′,y′) of pixels of the second image with respect to the first image. Extend the size of the first image to include all the 4 corner points of the second image. Then, Copy the pixels of the second image to the extended first image to achieve panorama stitching of the two views.

This is the scheme for stitching 2D images. A similar method can be used for stitching 3D images.