**Project 3: Image Processing, Local Feature Descriptor, and**

**Model Fitting**

**ESE 358 Computer Vision, SBU, ECE, SUNY at Stony Brook,**

**Prof. Murali Subbarao ©**

(35 points. Weight: 10%)

**Draft 1.0**

Two greyscale images are given as input (e.g. pic1grey300.jpg and pic2grey300.jpg). On these images, perform the following operations, and display the output.

All computations must be performed using the intensity data at the level of pixel values. Pixel values and positions (array indices) must be accessed and used in calculations. In particular, you are not permitted to use built in library functions of Python to perform filtering, edge detection, histogram computation, etc. Computations must be done explicitly by using gray-level values at each pixel. You must submit the input images, output images, source code, and also a project document (MS doc or pdf format) with input images, output images, and source code with captions and comments in the source code. No separate description is required.

1. (5 points**) Image Flitering/Convolution:** Convolve the two input images with a specified filter and display the outputs. To complete this step, implement convolution with an arbitrary MxM filter g(i,j) of an image f(k,l) of size NxN (e.g. N=300). Take M and N as given inputs where M is an odd number (e.g. M=9). Provide one option to read the filter coefficients of g(i,j) from a specified file, and another option to initialize g(i,j) with a Gaussian having the spread parameter sigma= M/4.0 . Use the formula for the 2D Gaussian to compute the coefficients. If the sum of all coefficients s, then normalize all the coefficients by dividing them by s so that their new sum total becomes 1.0. Compute the output image h(i,j) only for the interior pixels which are more than (M-1)/2 pixels inside of the border. Set the other pixels close to the border of h(i,j) to zero. For input image f(i,j) and filter g(k,l), the output image h(i,j) in the interior pixels is to be computed according to the following outline in pseudo-code:

for i= (M-1)/2 to N-1-(M-1)/2 // compute output for interior pixels only.

for j= (M-1)/2 to N-1-(M-1)/2

endfor

endfor

Display the output image h(i,j) for the option where g(k,l) is the Gaussian filter. Note: displaying the output image for other filters may require normalizing the range of pixels values which could be negative. You can try it for the case where g(k,l)=1.0/(M\*M) for all k,l, which is a mean-filter.

1. (10 points) **Edge detection.** First smooth the input image (N=300) 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, 2.0, and 3.0) and different thresholds (20, 30, 40, and 50). Make sure that the one-dimensional Gaussian filter is normalized so that the sum of the 9 coefficients is 1.0.

**Details:** 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 as in part 1 (index range from (M-1)/2 to N-1-(M-1)/2. Leave the other pixels as they were in the original image (instead of setting them to zero as in part 1).

First smooth the image along rows with the 1D Gaussian filter. The result of this step should then be smoothed along columns with the same 1D Gaussian filter to obtain the final result. After this, compute the gradient magnitude, and then threshold **Thresh** it with a value in the range 10 to 40. In computing the gradient magnitude, you may use the simple estimate of hx(i,j)=h(i+1,j)-h(i,j) for the partial-derivative along i, and hy(i,j)=h(i,j+1)-h(i,j) for the partial derivative along j. If the gradient magnitude is higher than the threshold **Thresh** at a pixel (i,j), then set the value of the pixel in the output image to be 255 (maximum brightness for unint8 type or 1.0 for float type). Compute edges in the interior pixels between rows/columns only, and set the other border pixels to be 0. (Note: if you use the image with pixel values from 0 to 255, you can start with sigma=1.0 and thresh=20, then try other combinations to see what will happen.)

Typically, the percentage of pixels at the border is small, e.g. 5% or less, and so any scheme which handles the border pixels in a reasonable way is acceptable.

Outline of filtering with a Gaussian filter.

Outline with explicit computations using For loops in pseudo-code format. You have to fill in the details and double check it, and then use it.

There may be minor errors in the outline, and you should fix them.

First compute the filter g(k) as:

sigma=2.0;

summ=0;

  for k = 0 to M-1

                g(k) =  exp ( - ( ( ( k - (M-1)/2 )^2 ) / (2\*sigma\*sigma) ) );

                summ += g(k);

            end

Then, normalize the filter coeffs as:

for k = 0 to M-1

                g(k) =  g(k)/summ;

  end

Now, use the above g(k) and filter the image f(i,j),  first along rows to get h1(i,j). Then the filter result h1(i,j) along columns with g(k) to get the final desired result h2(i,j).

For simplicity, initialize h1(i,j)=f(i,j) and h2(i,j)=h1(i,j) , for all i,j,  to take care of the border pixels.

%filter each row i

for i = 0 to N-1  
    for j = (M-1)/2 to N-1-(M-1)/2  
        summ = 0;  
        for k = 0 to M-1  
                summ = summ + g(k)\* f(i,j-(k-(M-1)/2));  
            end  
            h1(i,j) = summ;  
        end  
    end  
end

%filter each column j

for j = 0 to N-1  
    for i = (M-1)/2 to N-1-(M-1)/2  
        summ = 0;  
        for k = 0 to M-1  
              summ = summ + g(k)\* h1(i-(k- (M-1)/2),j);  
        end  
        h2(i,j) = summ;  
    end

end

Display the output h2(i,j).

1. **Corner detection and local feature descriptor** (10+10 points):
2. **Corner detection (10 points):** Modify the program for edge detection above to implement the Harris-corner detector. Take the window size for computing corner-metric-**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 (e.g. print the actual value of R at a known corner point, and use around half of that as threshold). Your output should be the original image superimposed with the detected corner points shown with a 3x3 region having 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) = (hx,hy) at each pixel.﻿﻿﻿﻿﻿﻿﻿﻿﻿ In order to keep the values small in the steps later, divide Ix by 10, and Iy by 10. (Note: if your image pixel values are from 0 to 255, you can try to divide Ix and Iy by 5 instead.)

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'. Use a separable filter.

5. At each pixel, let

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. Find a suitable threshold for R by trying different values (e.g. print the actual value of R at a known corner point, and use around half of that as threshold). Also, R depends on the input image characteristics. 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 are 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.

1. **Local feature descriptor (10 points):** 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 (use the equivalent of atan2(Ix,Iy) to get an angle in the range of 0 to 2pi radians; be alert about the units of measurement—degrees or radians), 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. For example, if the gradient magnitude is 7.3 at an angle of 125 degrees, then the histogram h(i) is updated something like here:

i= round\_off(125/45) % i=round\_off(2.78)=3

h(3) += 7.3

Let the histogram thus computed at this step be h(i) for i=0,1,2,…,7. 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.

For example, suppose the original histogram is:

i : 0 1 2 3 4 5 6 7

h(i) : 3.1 4.5 9.1 2.1 6.5 5.6 7.1 6.2

Then the maximum is at m=2 and therefore the rotation-normalized histogram will be

i : 0 1 2 3 4 5 6 7

hn(i) : 7.1 6.2 3.1 4.5 9.1 2.1 6.5 5.6

If the m-th entry is the maximum in h(i), for some m in the range 0,1, 2, …, to 7, then the normalized histogram hn(i) can be obtained with the original m-th entry brought to position 4 (corresponding to 180 degrees) from the original histogram h(i) as

% output hn(i)for i= 0 to 7 is the normalized histogram

%h(i) for i= 0 to 7 is the original histogram

**% compute original histogram h()**

**% find the location m of maximum**

**% such that h(m)>=h(i) for all i**

**% Rotate h(i)circularly to obtain hn(i) so that**

**% the maximum of h(i) at h(m) moves to hn(4)**

**% Note mod(k,8) gives the remainder of k when divided by 8**

for i= 0 : 7

hn(mod(4+i,8)) = h(mod(m+i,8));

end

Print the pixel coordinates of all the corner pixels followed by their associated local histogram (its local feature descriptor). For example, if there is a corner point at pixel position (i,j)=(10,20) with histogram 0.1, 0.2, 0.3, 0.4, 0.35, 0.25, 0.15, then the corresponding output will be one line as here:

pixel at (i,j)=(10,20) has histogram 0.1, 0.2, 0.3, 0.4, 0.35, 0.25, 0.15,

**END OF REQUIRED PARTS. SKIP THE REMAINING PARTS**

1. **SKIP THIS: Image matching**: **(5 points)** Take the two gray-level images provided as input. They are the images of a 2D object with an overlapping region, but one of them is translated, rotated, and scaled (affine transformed) with respect to the other in the 2D space. 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 manually (You may also find them automatically if you wish, by matching the local feature descriptors in one image to the closest ones in the other). Given K matching points (K>=15) that are spread-out in the image (i.e. they are not confined to a small region in the image), use one of the linear least-squares algorithm to find the 2D affine transformation matrix and the translation vector, and print the matrix and the vector.

**One possible method: Linear Least Squares Fit**

* + closed form solution
  + robust to noise
  + not robust to outliers



Identify the input, output, and the computational steps.

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



A**x**=**b**

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

A black text on a white background

Description automatically generated (linear least squares solution)

1. **Panorama stitching** : **Skip this.** **Optional:** Use the transformation parameters found above to stitch the two images together.

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



This gives the coordinates (x’,y’) in the first image of pixels (x,y) in the second image after the linear transformation with the parameters a,b,c,…,f. Extend the size of the first image to include all the 4 boundary corner points of the second image. Then, copy each and every pixels of the second image at (x,y) to the extended first image at location (x’,y’) where (x’,y’) is computed from (x,y) using the transformation parameters a,b,c,…, f. These steps achieve panorama stitching of the two views. Interpolation of gray-scale pixel values would be needed in this step. You may use a bilinear interpolation scheme.

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

APPENDIX

MATLAB LIBRARY GIVES THE FOLLOWING OUTPUTS.

THIS MAY BE USEFUL FOR COMPARISON.

% Program to check test data for Project 3

% Prof. M. Subbarao, SBU, ECE,

%Sample output of Canny's edge detector and Harris corner detector are

%shown.

p1g3 = imread('pic1grey300.jpg');

figure, imshow(p1g3) , title('pic1grey300')

%p1g3ep = edge(p1g3,'prewitt');

%figure, imshow(p1g3ep) , title('prewitt edges pic1grey300')

p1g3ec = edge(p1g3,'canny');

figure, imshow(p1g3ec) , title('Canny edges pic1grey300')

imwrite(p1g3ec,'pic1greyCannyEdges.jpg');

corners = detectHarrisFeatures(p1g3, 'MinQuality' , 0.0001 , 'FilterSize' , 7);

figure, imshow(p1g3); title('Harris Corners, pic1grey300'), hold on;

plot(corners.selectStrongest(200));

p2g3 = imread('pic2grey300.jpg');

figure, imshow(p2g3) , title('pic2grey300')

%p2g3ep = edge(p2g3,'prewitt');

%figure, imshow(p2g3ep) , title('prewitt edges pic1grey300')

p2g3ec = edge(p2g3,'canny');

figure, imshow(p2g3ec) , title('Canny edges pic2grey300')

imwrite(p2g3ec,'pic2greyCannyEdges.jpg');

corners = detectHarrisFeatures(p2g3, 'MinQuality' , 0.0001 , 'FilterSize' , 7);

figure, imshow(p2g3); title('Harris Corners, pic2grey300'), hold on;

plot(corners.selectStrongest(200));

A collage of different types of houses

Description automatically generated

Pic1grey300.jpg

A close-up of a house

Description automatically generated

Pic2grey300.jpg

A group of houses with different views

Description automatically generated with medium confidence

Pic1greyCanny.jpg

A black and white map of houses

Description automatically generated

Pic2greyCanny.jpg

![A screenshot of a computer

Description automatically generated]()

![A screenshot of a computer

Description automatically generated]()