Example Homework Solution

1. It is often useful to generate a synthetic image with known properties that can be used to test algorithms. Generate an image composed of two concentric circles as shown below. The inner circle should have a radius of 50 pixels and a mean value of 192. The outer circle should have a radius of 100 pixels and a mean value of 128. The background should have a mean value of 64. Add uniform random noise to each pixel in the range -16 .. +16 (see Matlab’s rand function). Save the image in “tif” format, and make sure the saved image looks correct. Turn in the Matlab program and the image that you generated. (Hint: recall the equation of a circle: $x^2 + y^2 = r^2$. These are the points on the circle border; to represent points inside the circle, you would you use an inequality.)

Solution:

```matlab
% Generate a synthetic image of concentric circles, and add noise.
clear all
close all
N = 400;
I = 64*ones(N,N); % An easy way to generate a constant value image
R1 = 100;
R2 = 50;

for r=1:N
    for c=1:N
        if (r-N/2)^2 + (c-N/2)^2 < R2^2
            I(r,c) = 192;
        elseif (r-N/2)^2 + (c-N/2)^2 < R1^2
            I(r,c) = 128;
        end
    end
end
I = I + 32*(rand(N)-0.5);
imshow(I,[0 255]), impixelinfo

% When writing to a "tif" file, make sure you convert to uint8
imwrite(uint8(I), 'test.tif');
```
2. The “coins.png” image (available in the Matlab example images directory) has light-colored coins against a dark background. As done in class, you can segment the coins from the background using a simple global thresholding technique, such as

\[ B = I > t; \]

where \( t \) is a value between 0 and 255. Write a Matlab program to calculate the maximum, mean and standard deviation of the pixels in the regions corresponding to the coins (you don’t need to do this for each coin; just the union of the coin regions). Turn in the Matlab program, the calculated values, and the image of the segmented coins.

Solution:

```matlab
% HW1, problem 5
% Segment coins (by thresholding) in the "coins.png" image;
% estimate maximum, mean, and standard deviation of the coin pixels.

% Good idea to do this at the beginning of all of your programs
clear all       % Clear out all old variables in the workspace
close all       % Close all open figures and images

I = imread('coins.png');    % Read image
imshow(I, []);

B = I > 80;                 % Threshold image, so that coin pixels = 1
figure, imshow(B);

% Go through image, and accumulate statistics.
% For the calculation of standard deviation, we will use the equation
%   Var(X) = mean(X^2) - mean(X)^2
maxval = 0;
sum = 0;        % Sum of coin values
sum2 = 0;       % Sum of values squared
nPts = 0;       % Number of pixels in coin regions

for r=1:size(I,1)
    for c=1:size(I,2)
        if B(r,c)       % if this is a coin pixel
            nPts = nPts + 1;
            if I(r,c)> maxval
                maxval = I(r,c);
            end

            % Accumulate sums.  Convert to double to get better precision.
            sum = sum + double(I(r,c));
            sum2 = sum2 + double(I(r,c))^2;
        end
    end
end

avg = sum/nPts;
var = sum2/nPts - avg^2;
```

1 There may be a couple of pixels that are not segmented correctly; don’t worry about that.
std = sqrt(var);

fprintf('Maximum value in coin regions = %d\n', maxval);
fprintf('Average value in coin regions = %f\n', avg);
fprintf('Standard deviation of pixels in coin regions = %f\n', std);

This prints out:
Maximum value in coin regions = 255
Average value in coin regions = 178.614854
Standard deviation of pixels in coin regions = 31.017945

If you use Matlab a lot, you will find that you can write programs that are much shorter and more efficient. Since Matlab is an interpreted language, “for” loops are generally pretty slow for doing matrix and array operations, so it is much better to write the operations using matrix operators. Below is an alternative solution that gives the same results:

% Alternative solution
I = imread('coins.png'); % Read image
B = I > 80; % Threshold image, so that coin pixels = 1
nPts = sum(sum(B)); % Sum entire image to count coin pixels
M = 255*uint8(B); % Make a mask image where coin pixels = 255
% Do a logical AND of each pixel in the mask image, with I.
% You could instead have used the command: I = I .* uint8(B).
I = bitand(I,M); % Force background pixels to 0
avg = sum(sum(I))/nPts;
var = sum(sum(double(I).^2))/nPts - avg^2;
std = sqrt(var);

Or, even simpler:

I = imread('coins.png'); % Read image
indices = I > 80; % Get indices of coin pixels
avg = mean(I(indices));
maxval = max(I(indices));
stdval = std(I(indices));

3. Develop a program to resize the example image “cameraman.tif” from its original size of 256x256 to an enlarged size of 400x400, using bilinear interpolation. For this problem, don’t use the Matlab functions “imresize”, “interp2” or the equivalent OpenCV function. Turn in your program and the resulting image.

Solution:

clear all
close all
% Source image
I1 = imread('cameraman.tif');
[M1,N1] = size(I1);

% Destination image
M2 = 400;
N2 = 400;
I2 = zeros(M2,N2);

% Scaling factors
sx = N2/N1;
sy = M2/M1;

% The transformation from I2 to I1 is
% x1 = x2/sx, y1 = y2/sy
% Scan through I2 and interpolate value from I1.
for y2 = 1:M2
  for x2 = 1:N2
    % Compute exact location of corresponding point in I1
    x1 = x2/sx;
    y1 = y2/sy;
    
    % Get closest integer coordinates less than x1,y1 (ie, round down)
    x0 = floor(x1);
    y0 = floor(y1);
    
    % Don't go outside bounds of the image
    if x0 < 1     x0 = 1;     end
    if y0 < 1     y0 = 1;     end
    if x0 > N1-1  x0 = N1-1;  end
    if y0 > M1-1  y0 = M1-1;  end
    
    % Get the offsets of (x1,y1) from (x0,y0)
    x = x1 - x0;
    y = y1 - y0;
    
    % Interpolate the value at (x1,y1), using the formula from the
    % lecture notes. Remember when we access a pixel, we have to use
    % (y,x) instead of (x,y).
    I2(y2,x2) = I1(y0,x0) * (1-x)*(1-y) + ...
               I1(y0,x0+1)*x*(1-y) + ...
               I1(y0+1,x0)*(1-x)*y + ...
               I1(y0+1,x0+1)*x*y;
  end
end
imshow(I2,[]);

The original (left) and resized (right) images: