Homework 3 solution template

CMPSCI 370 Spring 2019, UMass Amherst

Name: DongWon Park

Here is a template that your solutions should roughly follow. Include outputs as figures, and code should be included in the end.

1 Image filtering [10 points]

- Why is filtering with a Gaussian kernel preferable over a box kernel for denoising an image?
  Filtering with a box kernel produces edge effects. To smooth an image well, the filter should look like a 'fuzzy blob' so that takes an average from its neighbors more likely smoother and well rounded effect than using a box kernel. A box kernel is more likely a squared shape so that the result will looks like a square which shows edge effects.

- What is the effect of increasing the σ of the Gaussian kernel on the result of filtering?
  σ determines the extent of smoothing. Increasing σ of the Gaussian kernel produces wider kernel window so that it takes larger area of its neighborhood and as you can see the shape of the Gaussian probability distribution, the center value is getting lesser as the σ increases, and the result will contain less information at the center and will take more values from its wider neighborhood because the window size is getting bigger and distribution is much wider, so the final result will be smoother and more blurry image.

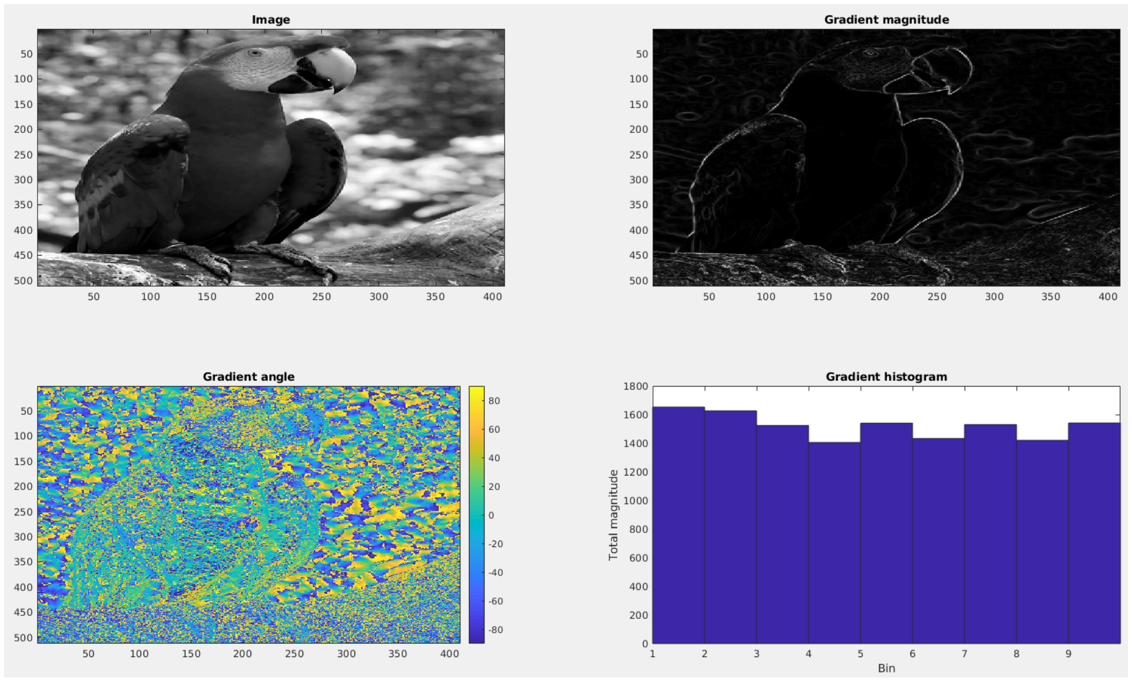
- When is median filtering preferable over Gaussian kernel filtering for denoising?
  Median filtering is preferable when the noise type is such as salt and pepper. Median filtering is a non-linear filter so it takes the median value from its neighbors and replace the pixel rather that taking the average from the neighbors. The result is great with salt and pepper noise type because the median filtering is edge preserving so that it doesn't blur images and replace the noise with the median value.

- Why is it a good idea to smooth an image before filtering with a derivative filter?
  If the image has noises, then filtering with a derivative filter will result into unexpected result because it will represent every difference from the noise. Then it would be impossible to detect edges from it. Instead, smoothing will make the image less noises so that taking a derivative will not affected by noises and it becomes more successful to detect edges from the signal.

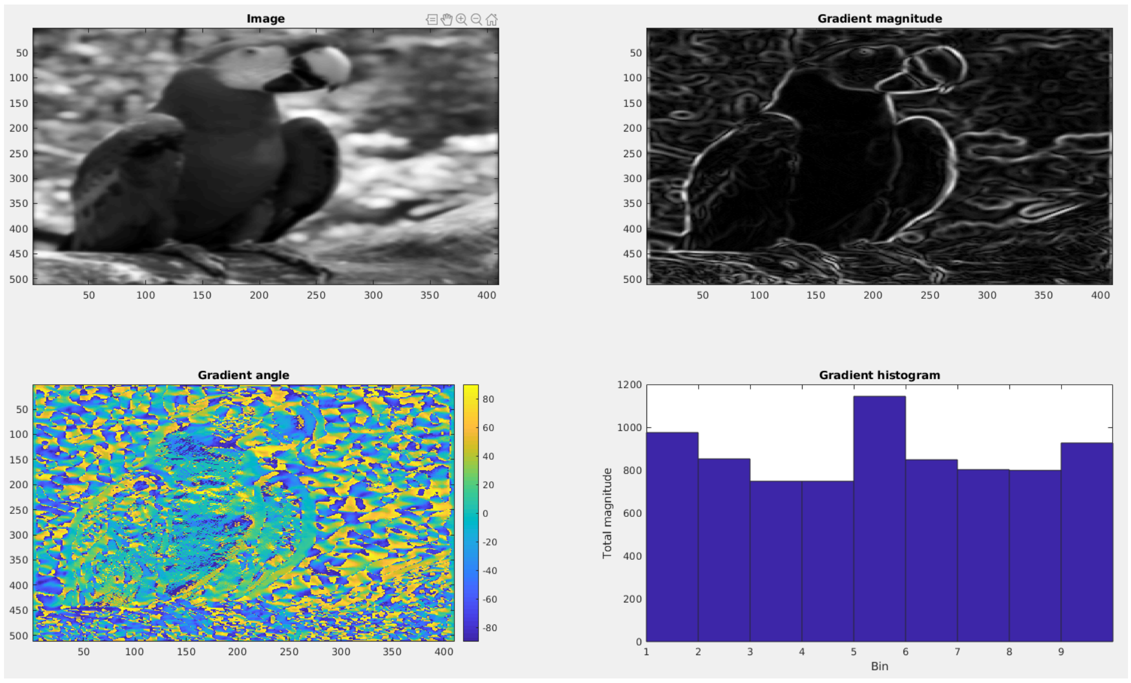
- What does filtering an image with a Laplacian of Gaussian filter do?
  Laplacaian of Gaussian is the second derivative of Gaussian which means it highlights regions of rapid intensity changes and it is used for edge detection. It can be used in a blob detector because the shape of Laplacian of Gaussian looks like a blob, circle.

2 Image Gradient and Orientation Histogram

1. Visualizing gradient magnitude, angle and histogram

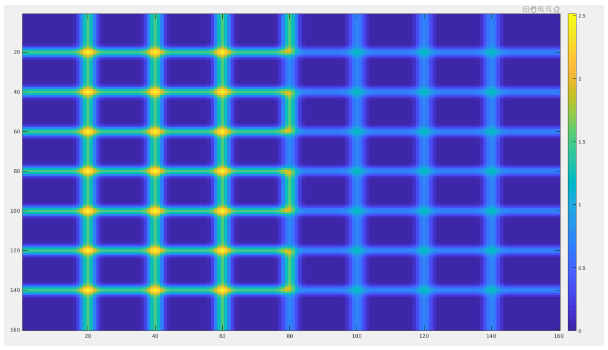


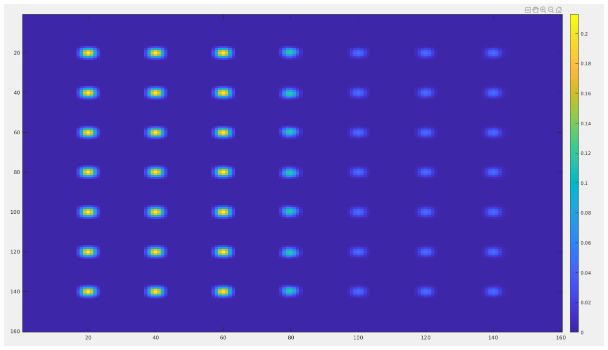
2. Visualizing gradient magnitude, angle and histogram for smoothed image

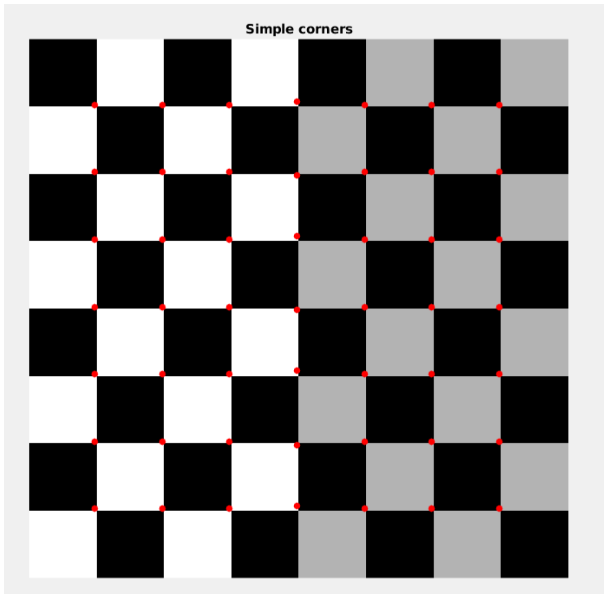


3 Corner detection

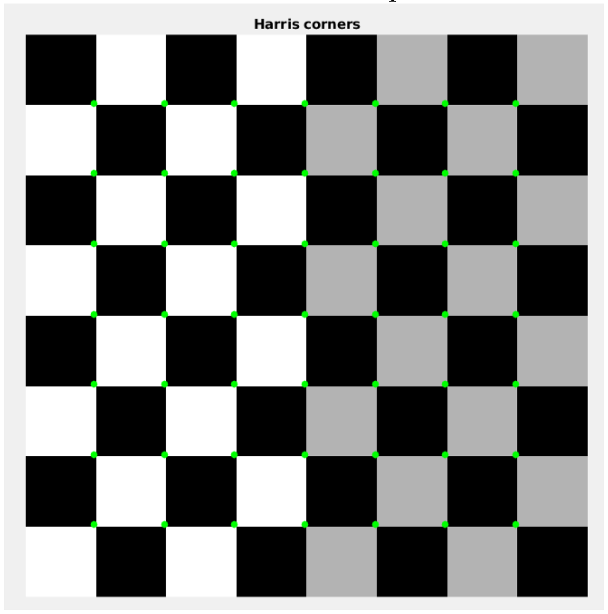
1. Output for the checkerboard image. When you get the correct output of corner detectors, the heatmaps of corner scores should have higher values (more yellowish) around corners and lower (more bluish) elsewhere.


Corner scores as heatmap (Simple)


Corner scores as heatmap (Harris)



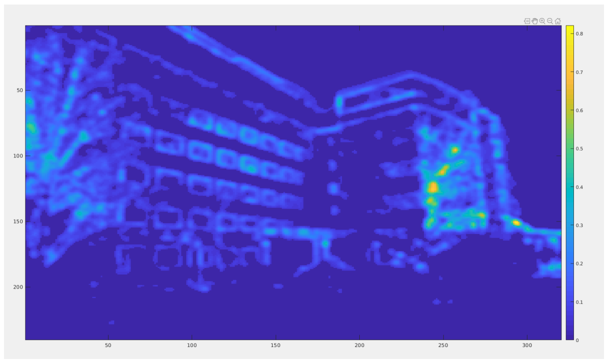
Simple corners



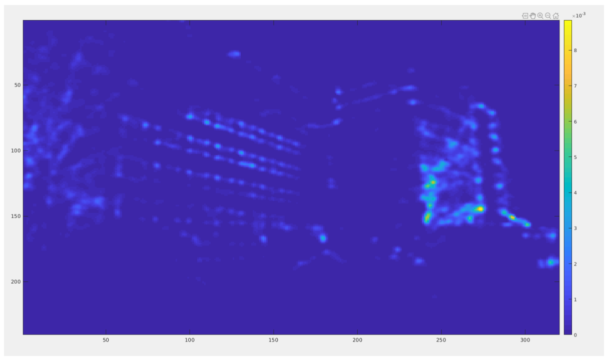
Harris corners

Figure 1: Results for the checkerboard image.

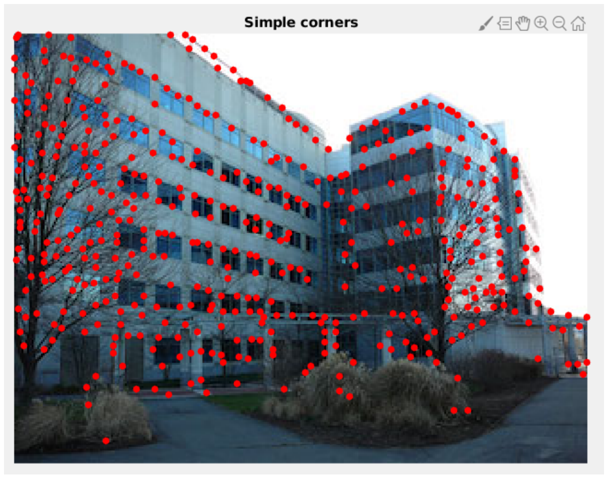
2. Output for the `'polymer-science-umass.jpg'` image



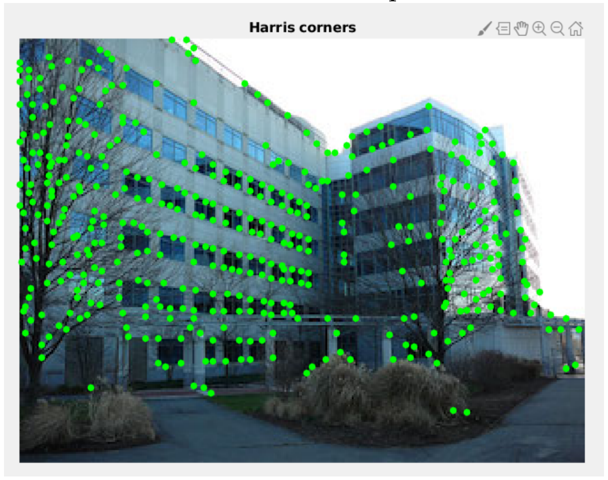
Corner scores as heatmap (Simple)



Corner scores as heatmap (Harris)



Simple corners

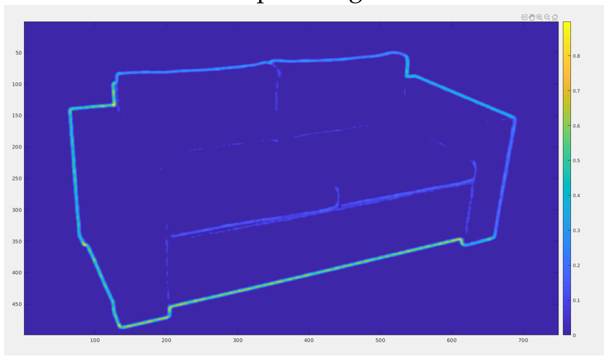


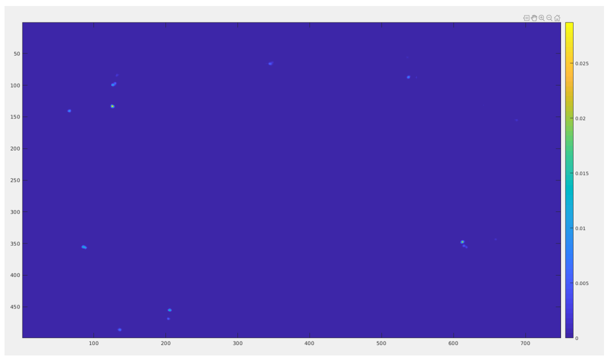
Harris corners

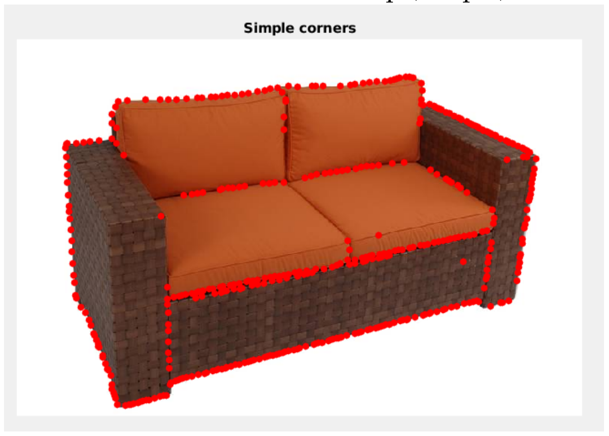
Figure 2: Results for the `'polymer-science-umass.jpg'` image.

3. Output for the image of your own choice

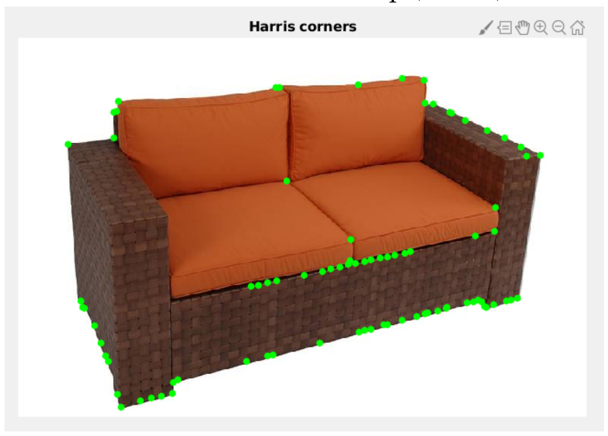
Input image


Corner scores as heatmap (Simple)


Corner scores as heatmap (Harris)



Simple corners



Harris corners

Figure 3: Results for the image of your own choice.

4 Solution code

Include the source code for your solutions as seen below (only the files you implemented are necessary). In latex the command `verbatiminput{alignChannels.m}` allows you to include the code verbatim as seen below. Regardless of how you do this the main requirement is that the included code is readable (use proper formatting, variable names, etc.) A screenshot of your code works to provided you include a link to source files.

4.a detectCorners.m

```
function [cx, cy, cs] = detectCorners(I, isSimple, w, th)
% This code is part of:
%
%   CMPSCI 370: Computer Vision, Spring 2016
%   University of Massachusetts, Amherst
%   Instructor: Subhransu Maji
%
%   Homework 3

% Convert to double format
I = im2double(I);

% Convert color to grayscale
if size(I, 3) > 1
    I = rgb2gray(I);
end

% Step 1: Compute corner score
if isSimple
    cornerScore = simpleScore(I, w);
else
    cornerScore = harrisScore(I, w);
end

% Step 2: Threshold corner score abd find peaks
cornerScore (cornerScore < th) = 0;
[cx, cy, cs] = nms(cornerScore);

%------------------------------------------------------------------------
%                                    Simple score function (Implement this)
%------------------------------------------------------------------------
function cornerScore = simpleScore(I, w)
cornerScore = zeros(size(I));

% Gaussian filter
Gsigma = fspecial('gaussian', 6*w+1, w);

for u = -1:1
    for v = -1:1
        if u == 0 && v == 0 % skip itself pixel
            continue;
        end
        imdiff = imfilter(I, filter_uv(u, v), 'replicate');
        cornerScore = cornerScore + imfilter(imdiff.^2, Gsigma, 'replicate');
    end
end



function fuv = filter_uv(u, v)
% Create a filter depends on the argument u, v shifts
  fuv = zeros([3 3]);
  fuv(v+2, u+2) = 1;
  fuv(2, 2) = -1;
  if u == 0 && v == 0
      fuv(2, 2) = 1;
  end

%--------------------------------------------------------------------------
%                                    Harris score function (Implement this)
%--------------------------------------------------------------------------
function cornerScore= harrisScore(I, w)
% cornerScore = 0.000102*rand(size(I)); % Replace this with your implementation

dx = [0 0 0;
      -1 0 1;
      0 0 0;];
dy = [0 1 0;
      0 0 0;
      0 -1 0;];

% Partial derivatives
Ix = imfilter(I, dx, 'replicate');
Iy = imfilter(I, dy, 'replicate');

% Gaussian filter
G = fspecial('gaussian', 6*w+1, w);

% Matrix M
Ix2 = imfilter(Ix.^2, G, 'replicate');
Iy2 = imfilter(Iy.^2, G, 'replicate');
IxIy = imfilter(Ix.*Iy, G, 'replicate');
IyIx = imfilter(Iy.*Ix, G, 'replicate');

k = 0.04;
detM = Ix2.*Iy2 - IxIy.*IyIx; % det(M) = ad - bc
traceM = Ix2 + Iy2; % trace(M) = a + d

cornerScore = detM - k*traceM.*traceM;
```

4.b imageGradient.m

```
function [mag, ang] = imageGradient(im)

    % Convert to grayscale, and convert to double format
    if size(im, 3) > 1
        imgd = double(rgb2gray(im))/255;
    else
        imgd = double(im)/255;
    end

    % Gradient filters
    fx = [-1 0 1];
    fy = fx';

    % Gradient image of each direction, x and y
    gx = imfilter(imgd, fx, 'replicate');
    gy = imfilter(imgd, fy, 'replicate');

    % Magnitude
    mag = sqrt(gx.^2 + gy.^2);
    % Angle
    ang = atand(gy./(gx+0.0000000000001)); % added non zero value to avoid divided by 0

    % Make bins for histogram
    [x, y] = size(imgd); % get image size
    bin = zeros([1 9]); % 9 bins
    for i = 1:x
        for j = 1:y
            % add 90 to each degree value to compute bins easier
            e = ang(i, j) + 90;
            % get corresponding bin numbers by modulo 20
            bin_num = idivide(e, uint8(20)) + 1;
            if e == 180
                bin_num = 9; % deal edge case which is 90 degree
            end
            % accumuldate corresponding magnitude per each angle degree values
            bin(1, bin_num) = bin(1, bin_num) + mag(i, j);
        end
    end

    figure
    subplot(2, 2, 1);
    imagesc(imgd); colormap(gca, 'gray');
    title('Image');
    subplot(2, 2, 2);
    imagesc(mag); colormap(gca, 'gray');
    title('Gradient magnitude');
    subplot(2, 2, 3);
    imagesc(ang); colormap(gca, 'parula'), colorbar;
    title('Gradient angle');
    subplot(2, 2, 4);
    axis = [1 2 3 4 5 6 7 8 9];
    bar(axis, bin, 'histc');
    xlabel('Bin'); ylabel('Total magnitude');
    title('Gradient histogram');

end
```