Table of Contents


```
img0 = imread('Lena.bmp');
figure(1), imshow(img0)

%{
Low pass filtering withholds strong variations and passes the low
frequency components that form mild or no
variations.
%}
```

Q 2

```
filt_avg1 = ones(3,3)/9;

img1_LP = uint8(filter2( filt_avg1, img0 ));
```

Q 3

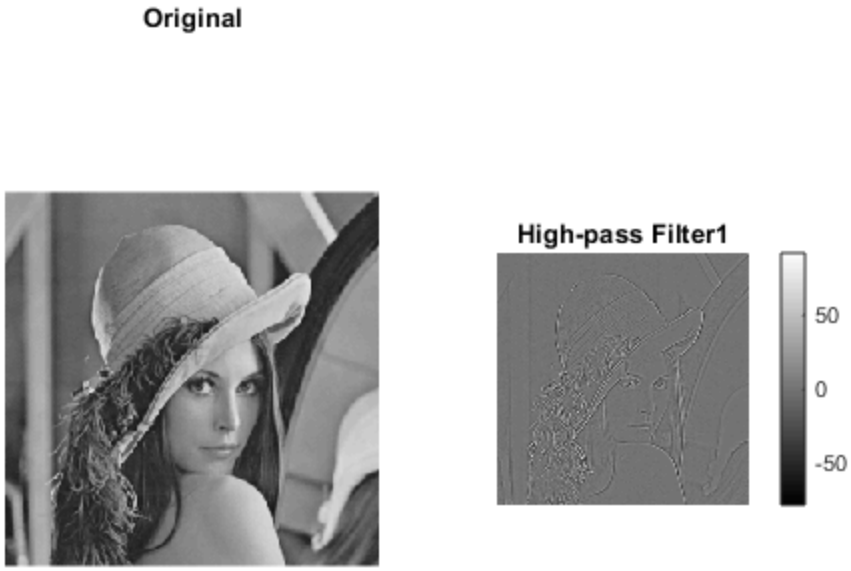
```
filt_avg2 = ones(3,3)/16;
filt_avg2(2,2) = 0.5;

img1_LP2 = uint8(filter2( filt_avg2, img0 ));
```

Q 4

```
figure(1), subplot(1,3,1),imshow(img0), title('Original')
subplot(1,3,2),imshow(img1_LP), title('Averaging Filter1')
subplot(1,3,3),imshow(img1_LP2), title('Averaging Filter2')
```

Part 2: Q 5

```
%{
It withholds smooth area by outputing zero or small
values, and passes the high frequency components that form
sharp variations.
%}

img1_HP = double(img0) - double(img1_LP);

figure(3), subplot(1,2,1), imshow(img0), title('Original'); axis equal
subplot(1,2,2), imshow(img1_HP, []), title('High-pass Filter1'),
 colorbar
```



PRT 3 Q 5

```
%The code clips the values outside [0,255] range and convert uint8

SP_factor = 2; % parameter to control the degree of sharpening
temp = double(img0) + double(img1_HP) * SP_factor;
img1_SP = uint8( (temp > 255) * 255 + (temp >= 0 & temp <= 255) .*
 temp );
% clip the values outside [0, 255] range and convert to uint8

subplot(1, 2, 1), imshow(img0), title('Original')
subplot(1, 2, 2), imshow(img1_SP), title('Sharpened')
```

Q 7

```
%When you average the values around that pixel, the photo will come
 out
%smoother becasue they will all have the same average value
```

Q 8

```
%It would accentuate the variations.
```

Q 9

```
% the second finds vertical edges and the first finds horizontal edges
```

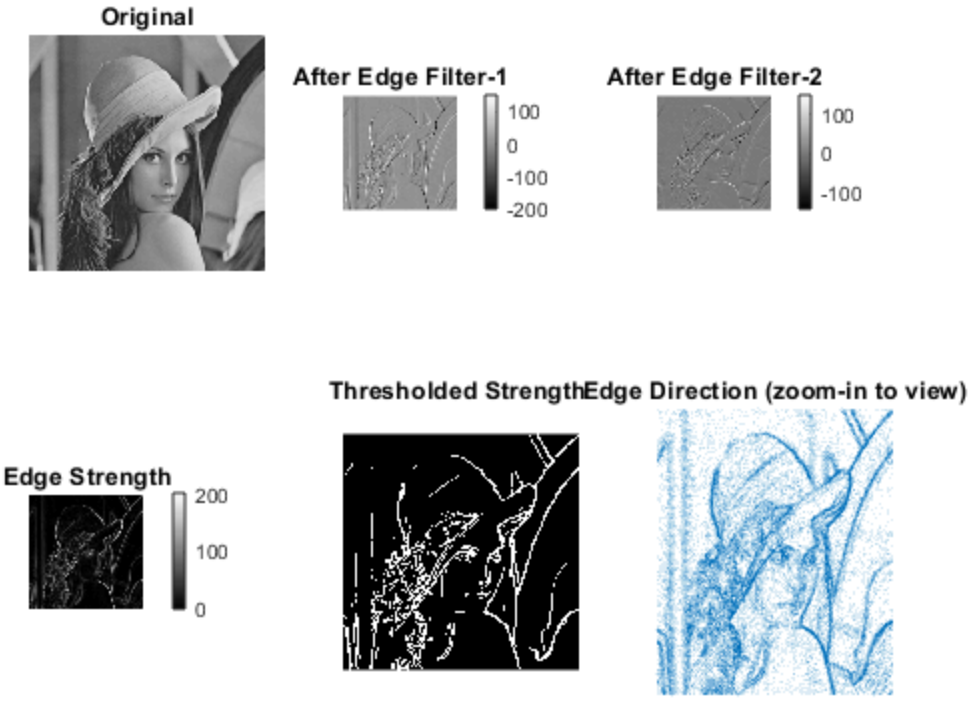
Q 10

```
% Depending on strong/weak ,an edge os where a pixel will have varying
% values in any direction of it. But there must be a consectutive
 numbers
% of these types of pixels to make a cohesive edge
```

Q 11

```
edge_filt1_1 = [-1, 0, 1; -1, 0, 1; -1, 0, 1], % "Prewitt" edge
 filters
edge_filt1_2 = [-1, -1, -1; 0, 0, 0; 1, 1, 1],

img1_edge1_1 = filter2( edge_filt1_1 / 3, img0 ); % apply edge filters
img1_edge1_2 = filter2( edge_filt1_2 / 3, img0 );

img1_edge1_2(1,:) = 0; % handle cases on image border
img1_edge1_1(:, 1) = 0;
img1_edge1_2(size(img0,1),:) = 0;
img1_edge1_1(:, size(img0,2)) = 0;

img1_edge1 = sqrt( double(img1_edge1_1) .^2 +
 double(img1_edge1_2) .^2 ); % edge strength
edge_th = 40; % threshold of edge strength

figure(4), subplot(2, 3, 1), imshow(img0), title('Original')
subplot(2, 3, 2), imshow(img1_edge1_1, []), title('After Edge
 Filter-1'), colorbar
subplot(2, 3, 3), imshow(img1_edge1_2, []), title('After Edge
 Filter-2'), colorbar
subplot(2, 3, 4), imshow(img1_edge1, []), title('Edge Strength'),
 colorbar
subplot(2, 3, 5), imshow(img1_edge1 > edge_th, []), title('Thresholded
 Strength ')
axis equal
subplot(2, 3, 6), st=1; %% visualize edge directions at a selected
 resolution
quiver( img1_edge1_2( size(img0,1):(-st):1, 1:st:size(img0,2) ), ...
        img1_edge1_1( size(img0,1):(-st):1, 1:st:size(img0,2) ) ),
axis([1, size(img0,2), 1, size(img0,1) ]), axis off
title('Edge Direction (zoom-in to view)')
```

```
edge_filt1_1 =

    -1     0     1
    -1     0     1
    -1     0     1


edge_filt1_2 =

    -1    -1    -1
     0     0     0
     1     1     1
```



Q 12

```
%Edge filter 1 points out the dominant vertical edges and FIlter edge
 2
%points out the dominant horizontal edges in the picture
```

Q 13

```
figure(6); st = 1;
quiver( img1_edge1_2( size(img0,1):(-st):1, 1:st:size(img0,2) ), ...
        img1_edge1_1( size(img0,1):(-st):1, 1:st:size(img0,2) ) ),
axis([1, size(img0,2), 1, size(img0,1) ]), axis off,
title('Edge Direction (zoom-in to view)')

%{
The arrows are pointing through or toward the edges of the image.
The edges that the arrows point to vary thoughout the figure since
 there
are many edges.


%}


edge_filt2_1 = [-1, 0, 1; -2, 0, 2; -1, 0, 1], % "Sobel" edge filters
```

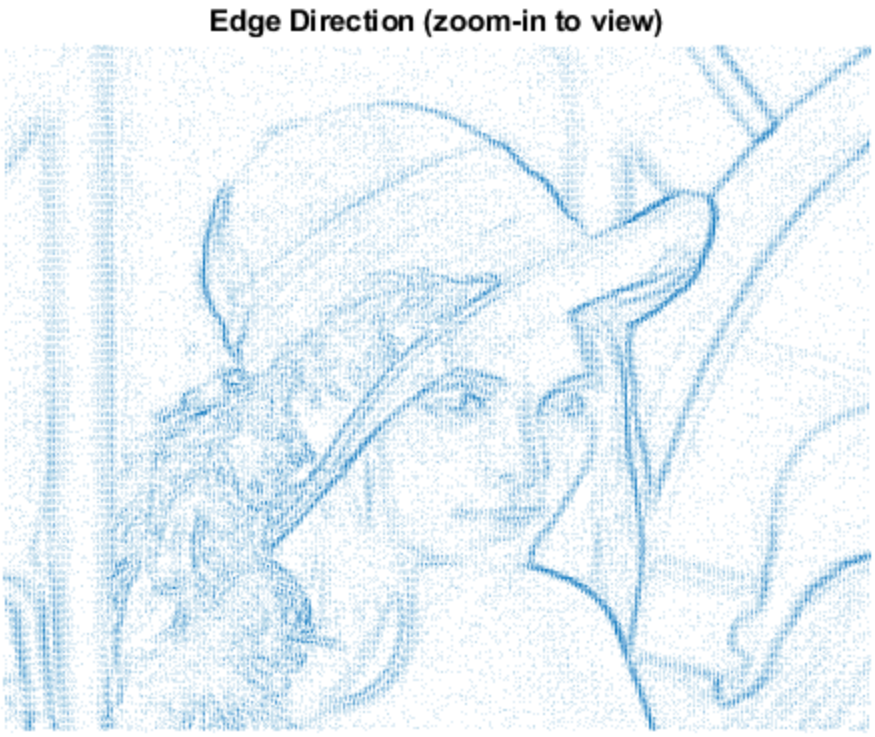
```
edge_filt2_2 = [-1, -2, -1; 0, 0, 0; 1, 2, 1],

img1_edge2_1 = filter2( edge_filt2_1 / 4, img0 ); % apply edge filters
img1_edge2_2 = filter2( edge_filt2_2 / 4, img0 );

%... continue with the visualization of edge detection results
% as shown above

img1_edge3 = edge(img0,'sobel');
img1_edge4 = edge(img0,'canny');
figure(7), subplot(1, 3, 1), imshow(img0), title('Original')
subplot(1, 3, 2), imshow(img1_edge3), title('Sobel Edge Detector')
subplot(1, 3, 3), imshow(img1_edge4), title('Canny Edge Detector')
```

edge_filt2_1 =

```
    -1     0     1
    -2     0     2
    -1     0     1
```

edge_filt2_2 =

```
    -1    -2    -1
     0     0     0
     1     2     1
```


Edge Direction (zoom-in to view)


Original
Sobel Edge Detector
Canny Edge Detector

Blg Part 3

Q 14

Lighter Circles can be dealt with be adjusting the detection function's sensitivity.

Q 15

No because the filters searched for edges with vertical and horizontal components for an image but a circle is a rounded shape , meaning it is not entirly vertical or horizontal.

Q 16

Gradient values specifcy the average value of a pixel based on the surrounding pixel values.

Q 17

```
rgb = imread('coloredChips.png');
gray_image = rgb2gray(rgb);

[centers,radii] = imfindcircles(rgb,[20
 25],'ObjectPolarity','dark', ...
          'Sensitivity',0.95);
[centersBright,radiiBright,metricBright] = imfindcircles(rgb,[20
 25], ...
    'ObjectPolarity','bright','Sensitivity',0.92,'EdgeThreshold',0.1);
imshow(rgb)
hBright = viscircles(centersBright, radiiBright,'Color','b');
h = viscircles(centers,radii);
```

Q 18

```
% Load images.
buildingDir =
 fullfile(toolboxdir('vision'), 'visiondata', 'building');
%buildingDir = fullfile('.\Tahoe');
buildingScene = imageDatastore(buildingDir);

% Display images to be stitched
figure(10);montage(buildingScene.Files)

% Read the first image from the image set.
I = readimage(buildingScene, 1);

% Initialize features for I(1)
grayImage = rgb2gray(I);
points = detectSURFFeatures(grayImage);
[features, points] = extractFeatures(grayImage, points);

% Initialize all the transforms to the identity matrix. Note that the
% projective transform is used here because the building images are
 fairly
% close to the camera. Had the scene been captured from a further
 distance,
% an affine transform would suffice.
```

```
numImages = numel(buildingScene.Files);
tforms(numImages) = projective2d(eye(3));

% Initialize variable to hold image sizes.
imageSize = zeros(numImages,2);

% Iterate over remaining image pairs
for n = 2:numImages

% Store points and features for I(n-1).
    pointsPrevious = points;
    featuresPrevious = features;

% Read I(n).
    I = readimage(buildingScene, n);

% Convert image to grayscale.
    grayImage = rgb2gray(I);

% Save image size.
    imageSize(n,:) = size(grayImage);

% Detect and extract SURF features for I(n).
    points = detectSURFFeatures(grayImage);
    [features, points] = extractFeatures(grayImage, points);

% Find correspondences between I(n) and I(n-1).
    indexPairs = matchFeatures(features, featuresPrevious, 'Unique',
 true);

    matchedPoints = points(indexPairs(:,1), :);
    matchedPointsPrev = pointsPrevious(indexPairs(:,2), :);

% Estimate the transformation between I(n) and I(n-1).
    tforms(n) = estimateGeometricTransform(matchedPoints,
 matchedPointsPrev,...
        'projective', 'Confidence', 99.9, 'MaxNumTrials', 2000);

% Compute T(n) * T(n-1) * ... * T(1)
    tforms(n).T = tforms.T * tforms(n-1).T;
end


% Compute the output limits for each transform
for i = 1:numel(tforms)
    [xlim(i,:), ylim(i,:)] = outputLimits(tforms(i), [1
 imageSize(i,2)], [1 imageSize(i,1)]);
end

avgXLim = mean(xlim, 2);
[~, idx] = sort(avgXLim);
centerIdx = floor((numel(tforms)+1)/2);

centerImageIdx = idx(centerIdx);
```

```
Tinv = invert(tforms(centerImageIdx));

for i = 1:numel(tforms)
    tforms(i).T = tforms(i).T * Tinv.T;
end

for i = 1:numel(tforms)
    [xlim(i,:), ylim(i,:)] = outputLimits(tforms(i), [1
 imageSize(i,2)], [1 imageSize(i,1)]);
end

maxImageSize = max(imageSize);

% Find the minimum and maximum output limits
xMin = min([1; xlim(:)]);
xMax = max([maxImageSize(2); xlim(:)]);

yMin = min([1; ylim(:)]);
yMax = max([maxImageSize(1); ylim(:)]);

% Width and height of panorama.
width = round(xMax - xMin);
height = round(yMax - yMin);

% Initialize the "empty" panorama.
panorama = zeros([height width 3], 'like', I);

blender = vision.AlphaBlender('Operation', 'Binary mask', ...
    'MaskSource', 'Input port');

% Create a 2-D spatial reference object defining the size of the
 panorama.
xLimits = [xMin xMax];
yLimits = [yMin yMax];
panoramaView = imref2d([height width], xLimits, yLimits);

% Create the panorama.
for i = 1:numImages

    I = readimage(buildingScene, i);

% Transform I into the panorama.
    warpedImage = imwarp(I, tforms(i), 'OutputView', panoramaView);

% Generate a binary mask.
    mask = imwarp(true(size(I,1),size(I,2)), tforms(i), 'OutputView',
 panoramaView);

% Overlay the warpedImage onto the panorama.
    panorama = step(blender, panorama, warpedImage, mask);
end
```

```
figure(27);
imshow(panorama)
```

```
Error using *
Too many input arguments.

Error in mod5lab2 (line 222)
    tforms(n).T = tforms.T * tforms(n-1).T;
```

Published with MATLAB® R2019a