Task 6: Image Filter and Edge Detection

Name: Aleksandar Jakovljevic
Degree: BA
ID: 20613063

Image

The following image, photographed by me, is used for Problems 1 to 3:

Metadata	**Riomaggiore**
Image	
Make	Nokia
Model	Nokia 8
Date and time	2018-08-14 14:01
Image file type	JPEG
Resolution	3840 x 2160
Focal length	4 mm
Exposure	1/2196
ISO	101
F-number	2
GPS	Off
Flash	Off

Problem 1

Parts (a) and (b)

```
% === 1a ================================================================ %

% Load image
imgTownFile = 'imgInputs\Riomaggiore.jpg';
imgTown = imread(imgTownFile);
imgTown = im2double(imgTown);
infoTown = imfinfo(imgTownFile);
channels = infoTown.NumberOfSamples;
sizeTown = [infoTown.Width infoTown.Height];

% Resize image
dim_resized = 200; % shortest length in pixels
imgTownResized = imresize(imgTown, dim_resized/min(sizeTown));

% Define kernel
H = [0, -1, 0; -1, 4, -1; 0, -1, 0];

% Perform convolution using conv2
for a = 1:channels
    imgTown_conv2(:,:,a) = conv2(imgTownResized(:,:,a), H, 'same');
end
figure(1); imshow(imgTown_conv2);
imwrite(imgTown_conv2, 'imgOutputs\Prob1\imgTown_conv2.jpg');

% === 1b ================================================================ %

% Set convolution parameters
k = (length(H)-1)/2;
sizeTownResized = size(imgTownResized);
imgTownH = sizeTownResized(1);
imgTownW = sizeTownResized(2);
imgTownPadded = zeros(imgTownH+2, imgTownW+2, channels);
imgTownPadded(2:end-1,2:end-1,:) = imgTownResized(:,:,:);
% the edges are zero-padded to allow for the convolved image to be the
% same size as the original image
imgTown_loop = zeros(imgTownH, imgTownW, channels);

% Perform convolution in a loop
for a = 1:channels
    for ii = 1:imgTownH
        for jj = 1:imgTownW
            for u = -k:k
                for v = -k:k
                    imgTown_loop(ii,jj,a) = imgTown_loop(ii,jj,a) + ...
                        H((2*k)+u,(2*k)+v)*imgTownPadded(ii-u+1,jj-v+1,a);
                end
            end
        end
    end
end
figure(2); imshow(imgTown_loop);
```

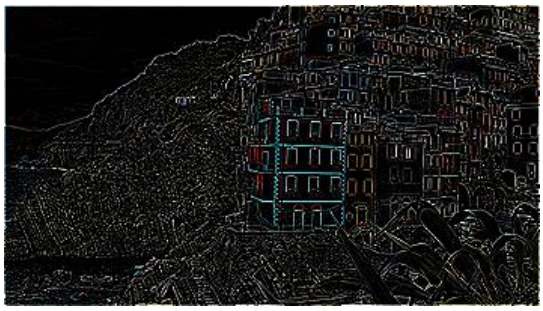
```
imwrite(imgTown_loop, 'imgOutputs\Prob1\imgTown_loop.jpg');

% ========================================================================= %
```

imgTown_conv2	imgTown_loop
	

The above comparison shows that both methods yield the same image.

Problem 2

```
% === 2 ================================================================== %

% Define kernels
H1 = (1/9)*ones(3,3);
H2 = [0, -1, 0; -1, 4, -1; 0, -1, 0];
H3 = (-0.55)*ones(3,3); H3(2,2) = 5.40;
H4 = [0.0030, 0.0133, 0.0219, 0.0133, 0.0030; 0.0133, 0.0596, 0.0983, ...
        0.0596, 0.0133; 0.0219, 0.0983, 0.1621, 0.0983, 0.0219; ...
        0.0133, 0.0596, 0.0983, 0.0596, 0.0133; 0.0030, 0.0133, ...
        0.0219, 0.0133, 0.0030];

% Perform convolutions using conv2
for a = 1:channels
    imgTown_H1(:,:,a) = conv2(imgTownResized(:,:,a), H1, 'same');
end
figure(3); imshow(imgTown_H1);
imwrite(imgTown_H1, 'imgOutputs\Prob2\imgTown_H1.jpg');
for a = 1:channels
    imgTown_H2(:,:,a) = conv2(imgTownResized(:,:,a), H2, 'same');
end
figure(4); imshow(imgTown_H2);
imwrite(imgTown_H2, 'imgOutputs\Prob2\imgTown_H2.jpg');
for a = 1:channels
    imgTown_H3(:,:,a) = conv2(imgTownResized(:,:,a), H3, 'same');
end
figure(5); imshow(imgTown_H3);
imwrite(imgTown_H3, 'imgOutputs\Prob2\imgTown_H3.jpg');
for a = 1:channels
    imgTown_H4(:,:,a) = conv2(imgTownResized(:,:,a), H4, 'same');
end
figure(6); imshow(imgTown_H4);
imwrite(imgTown_H4, 'imgOutputs\Prob2\imgTown_H4.jpg');

% ========================================================================= %
```

imgTown_H1	imgTown_H2
	
imgTown_H3	**imgTown_H4**
	

The filter H_1 applies a uniform blur across the image by averaging the value of all of the pixels in a 3x3 kernel and applying this value to the output pixel.

The filter H_2 highlights the edges in an image by comparing the pixel value at the centre of a 3x3 kernel to the four adjacent pixels. Since the kernel sums to zero, any area where these pixels all have the same colour appears as black in the output image, whereas any areas where there is a difference between adjacent pixels will become highlighted.

The filter H_3 sharpens the image by increasing the contrast in areas where adjacent pixels have different values. It does this by increasing the value at the centre pixel of a 3x3 kernel and equally subtracting all other values in the kernel, summing to one. In areas where neighbouring pixels are the same value there is no discernible change, but at the edges of objects the colour discontinuity is exaggerated which leads to object edges looking sharper.

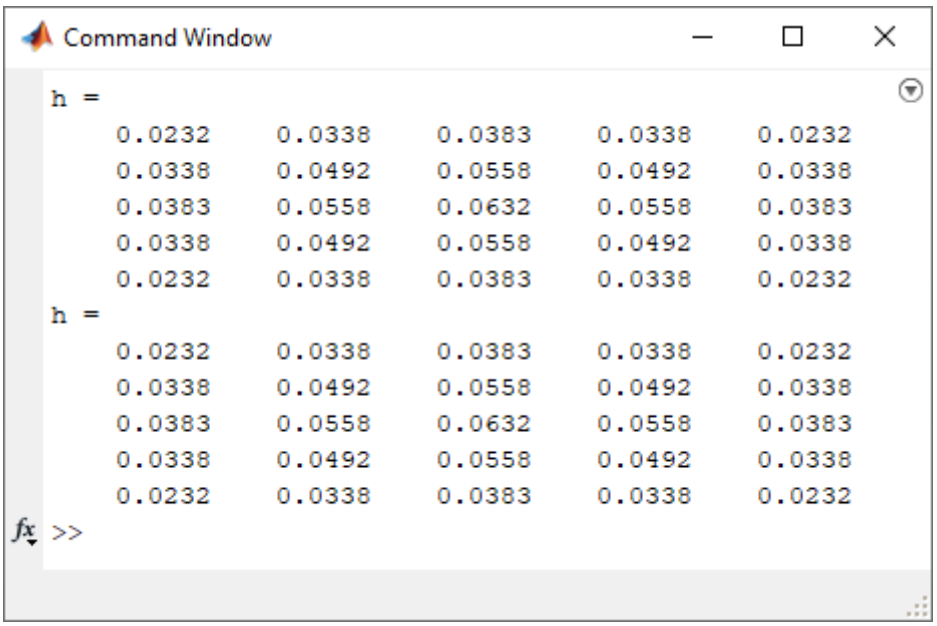
The filter H_4 applies a Gaussian blur to the image by weighting values using a 5x5 kernel containing values from a Gaussian distribution function. The end result is similar to the image obtained from H_1 as they are both blurring the image, however because of the larger size and gradual change in values in the Gaussian kernel, the image produced by H_4 appears slightly smoother.

Problem 3

Part (a)

```
% === 3a ================================================================ %

% Compute kernel to be matched
h = fspecial('gaussian', 5, 2)

% Define kernel parameters
sizeGaussian = 5;
sigmaGaussian = 2;
```

```
h = zeros(sizeGaussian, sizeGaussian, 3); % two matrices for temp storage

% Loop to populate matrix of x^2+y^2
for a = 1:sizeGaussian
    h(a,:,1) = (a-ceil(sizeGaussian/2))^2;
    h(:,a,2) = (a-ceil(sizeGaussian/2))^2;
end
h(:,:,3) = h(:,:,1)+h(:,:,2); h = h(:,:,3);

% Compute final matrix h
h = ((1/(2*pi*(sigmaGaussian^2)))*exp(-h/(2*(sigmaGaussian^2))));
h = h./sum(sum(h))

% ========================================================================= %
```



The command window output shows that the above two methods both yield the same kernel.

Part (b)

```
% === 3b ================================================================= %

% Define kernels
h1 = fspecial('gaussian', 20, 2);
h2 = fspecial('gaussian', 20, 3);

% Perform convolutions using conv2
for a = 1:channels
    imgTown_h1(:,:,a) = conv2(imgTownResized(:,:,a), h1, 'same');
end
figure(7); imshow(imgTown_h1);
imwrite(imgTown_h1, 'imgOutputs\Prob3\imgTown_h1.jpg');
for a = 1:channels
    imgTown_h2(:,:,a) = conv2(imgTownResized(:,:,a), h2, 'same');
end
figure(8); imshow(imgTown_h2);
imwrite(imgTown_h2, 'imgOutputs\Prob3\imgTown_h2.jpg');

% ========================================================================= %
```

imgTown_h1	imgTown_h2
	

The difference between the kernels H_1 and H_2 is their σ (standard deviation) value. The kernel H_2 has a higher standard deviation, which results in a "wider" Gaussian function and as a result larger weight values for pixels farther from the centre of the kernel. Because neighbouring pixels have greater weight on the output pixel, the result is a blurrier output image.

Part (c)

```
% === 3c ================================================================ %

% Define kernels
h2 = fspecial('gaussian', 20, 3);
h3 = fspecial('gaussian', 50, 3);

% Perform convolutions using conv2
for a = 1:channels
    imgTown_h2(:,:,a) = conv2(imgTownResized(:,:,a), h2, 'same');
end
figure(8); imshow(imgTown_h2);
imwrite(imgTown_h2, 'imgOutputs\Prob3\imgTown_h2.jpg');
for a = 1:channels
    imgTown_h3(:,:,a) = conv2(imgTownResized(:,:,a), h3, 'same');
end
figure(9); imshow(imgTown_h3);
imwrite(imgTown_h3, 'imgOutputs\Prob3\imgTown_h3.jpg');

% ======================================================================== %
```

imgTown_h2	imgTown_h3
	

The difference between the kernels H_2 and H_3 is their size. The kernel H_3 is larger, which means that the kernel is sampling from a larger number of pixels. This results in a blurrier output image.

Part (d)

```
% === 3d ================================================================ %

% Define kernels
h2 = fspecial('gaussian', 20, 3);
h4 = fspecial('gaussian', 20, 20);

% Perform convolutions using conv2
for a = 1:channels
    imgTown_h2(:,:,a) = conv2(imgTownResized(:,:,a), h2, 'same');
end
figure(8); imshow(imgTown_h2);
imwrite(imgTown_h2, 'imgOutputs\Prob3\imgTown_h2.jpg');
for a = 1:channels
    imgTown_h4(:,:,a) = conv2(imgTownResized(:,:,a), h4, 'same');
end
figure(10); imshow(imgTown_h4);
imwrite(imgTown_h4, 'imgOutputs\Prob3\imgTown_h4.jpg');

% ======================================================================== %
```

imgTown_h2	imgTown_h4
	

The difference between the kernels H_2 and H_4 is their σ (standard deviation) value. The kernel H_4 has a significantly higher standard deviation, which results in a much "wider" Gaussian function and as a result the Gaussian kernel becomes somewhat similar to a uniform blur kernel. The combination of the large size and large standard deviation of this kernel results in significant blurring over large portions of the image, and the end result is an extremely blurry image where individual objects have become difficult to distinguish.

Problem 4

```
% === 4 ================================================================ %

% Parameters
bookSize = [24, 31.5]; % cm
bookImgSize = bookSize*50;
bookVertices = [bookImgSize(1), 0; 0, 0; bookImgSize(1), ...
        bookImgSize(2); 0, bookImgSize(2)];
dir4in = 'imgInputs\Prob4';
dir4out = 'imgOutputs\Prob4';
img4list = dir('imgInputs\Prob4\*.jpg');
nImg4 = numel(img4list);
```

```
% Process images
for a = 1:nImg4
    % Load image
    imgBook = imread(fullfile(dir4in, img4list(a).name));
    imgBW = edge(imgaussfilt(rgb2gray(imgBook)), 'Canny', [0.10, 0.30]);

    % Perform Hough transform and determine peaks
    [HT, theta, rho] = hough(imgBW, 'RhoResolution', 0.5, ...
            'Theta', -90:0.5:89.5);
    HTpeaks = houghpeaks(HT, 4, 'Threshold', 0.2*max(HT(:)));
    HTlines = houghlines(imgBW, theta, rho, HTpeaks);

    % Remove repeat lines (may arise from discontinuities in imgBW
    % depedent on Canny thresholds and image quality)
    templines = [HTlines.theta; HTlines.rho]';
    [templines, index] = unique(templines, 'rows');
    HTpts = horzcat([reshape([HTlines.point1], 2, size(HTlines,2))]', ...
            [reshape([HTlines.point2], 2, size(HTlines,2))]');
    HTpts = HTpts(index,:);

    % Find equations of all four lines in homogenous coordinates
    intlines = zeros(4,3);
    for ii = 1:4
        intlines(ii,:) = cross([HTpts(ii,1), HTpts(ii,2), 1], ...
                [HTpts(ii,3), HTpts(ii,4), 1]);
    end

    % Calculate 6 points of intersection and convert to Cartesian coords
    PoI = zeros(6,2);
    count = 1;
    for ii = 1:4
        for jj = ii+1:4
            PoI(count,:) = hom2cart(cross(intlines(ii,:), intlines(jj,:)));
            count = count + 1;
        end
    end

    % Remove values that are out of frame due to parallel edges
    PoI(PoI(:,1) < 0 | PoI(:,2) < 0, :) = [];

    % Check that long/short edges are in same order as bookVertices
    % If not then rearrange the indices so that they match
    % (this may occur depending on the rotation of the book in the image)
    shortDist = sqrt((((PoI(2,1))-(PoI(1,1)))^2)+...
            (((PoI(2,2))-(PoI(1,2)))^2));
    longDist = sqrt((((PoI(2,1))-(PoI(4,1)))^2)+...
            (((PoI(2,2))-(PoI(4,2)))^2));
    if shortDist > longDist
        PoI([1 2 3 4],:) = PoI([2 4 1 3],:);
    end

    % Perform transform and crop image
    tform = fitgeotrans(PoI, bookVertices, 'projective');
    [imgBookMod, RBook] = imwarp(imgBook, tform);
    imgBookCrop = imcrop(imgBookMod, [-RBook.XWorldLimits(1), ...
            -RBook.YWorldLimits(1), bookImgSize]);
    figure(10+a); imshow(imgBookCrop);
```

```
    filename = strcat('imgOutputs\Prob4\imgBook_', num2str(a), '.jpg');
    imwrite(imgBookCrop, filename);
end

% ========================================================================= %
```

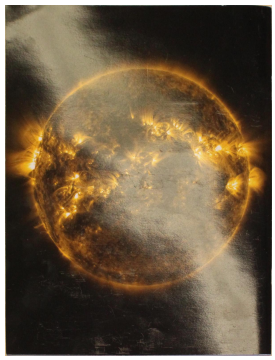
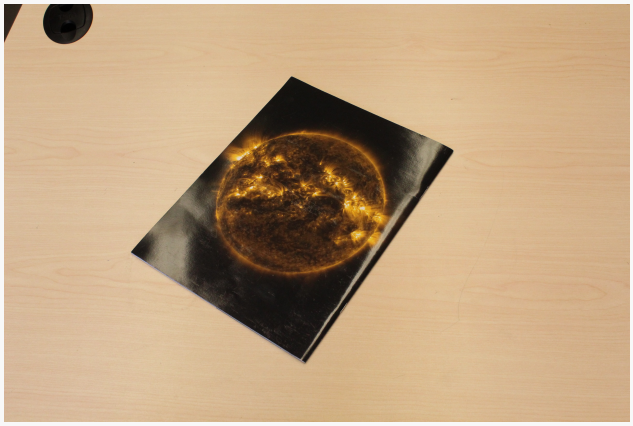
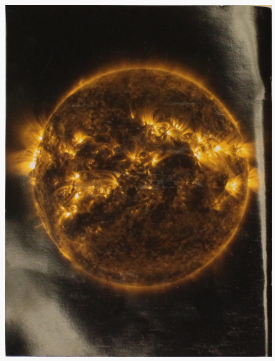
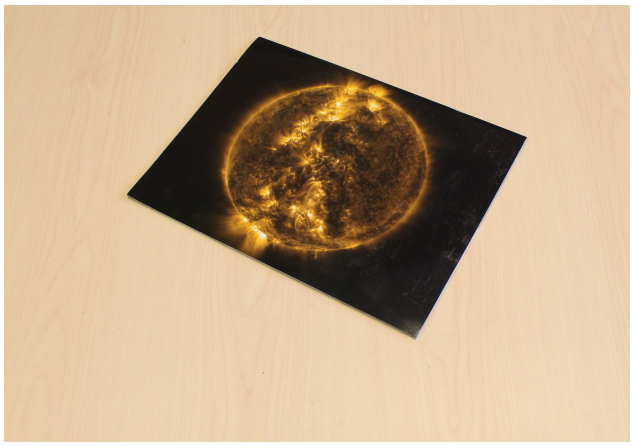
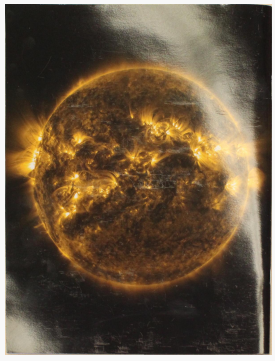
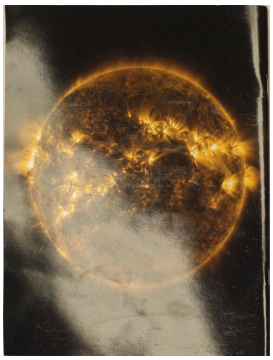
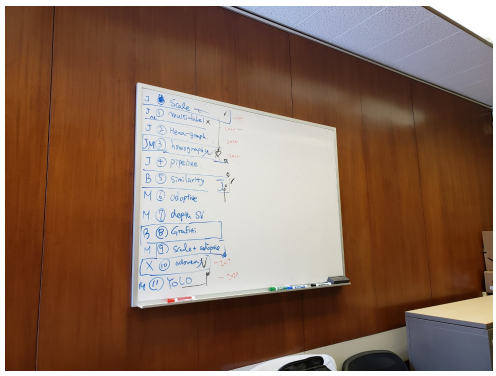
Image	Input	Output
IMG_0080		
IMG_0081		
IMG_0082		
IMG_0084		

Image	Input	Output
IMG_0085		

Problem 5

```
% === 5 ========================================================================= %

% Load images
imgBookFile = 'imgInputs\Prob5\IMG_0080.jpg';
imgBook = imread(imgBookFile);
imgBoardFile = 'imgInputs\Prob5\board.jpg';
imgBoard = imread(imgBoardFile);

% Parameters
bookSize = [24, 31.5]; % cm
bookImgSize = bookSize*50;
bookVertices = [bookImgSize(1), 0; 0, 0; bookImgSize(1), ...
        bookImgSize(2); 0, bookImgSize(2)];

% Find vertices of book, transform, and crop (same code as before)
imgBW = edge(imgaussfilt(rgb2gray(imgBook)), 'Canny', [0.10, 0.30]);
[HT, theta, rho] = hough(imgBW, 'RhoResolution', 0.5, ...
        'Theta', -90:0.5:89.5);
HTpeaks = houghpeaks(HT, 4, 'Threshold', 0.2*max(HT(:)));
HTlines = houghlines(imgBW, theta, rho, HTpeaks);
templines = [HTlines.theta; HTlines.rho]';
[templines, index] = unique(templines, 'rows');
HTpts = horzcat([reshape([HTlines.point1], 2, size(HTlines,2))]', ...
        [reshape([HTlines.point2], 2, size(HTlines,2))]');
HTpts = HTpts(index,:);
intlines = zeros(4,3);
for ii = 1:4
    intlines(ii,:) = cross([HTpts(ii,1), HTpts(ii,2), 1], ...
        [HTpts(ii,3), HTpts(ii,4), 1]);
end
PoI = zeros(6,2);
count = 1;
for ii = 1:4
    for jj = ii+1:4
        PoI(count,:) = hom2cart(cross(intlines(ii,:), intlines(jj,:)));
        count = count + 1;
    end
```

```
end
PoI(PoI(:,1) < 0 | PoI(:,2) < 0, :) = [];
shortDist = sqrt((((PoI(2,1))-(PoI(1,1)))^2)+ ...
        (((PoI(2,2))-(PoI(1,2)))^2));
longDist = sqrt((((PoI(2,1))-(PoI(4,1)))^2)+ ...
        (((PoI(2,2))-(PoI(4,2)))^2));
if shortDist > longDist
    PoI([1 2 3 4],:) = PoI([2 4 1 3],:);
end
tform = fitgeotrans(PoI, bookVertices, 'projective');
[imgBookMod, RBook] = imwarp(imgBook, tform);
imgBookCrop = imcrop(imgBookMod, [-RBook.XWorldLimits(1), ...
        -RBook.YWorldLimits(1), bookImgSize]);

% Select board vertices
figure(16); imshow(imgBoard);
clc
fprintf('Select vertices of board.\n')
polyBoard = impoly;
coordBoard = getPosition(polyBoard);
fprintf('Board vertices set.\n\n')
delete(polyBoard);

% Check if the user closed the polygon going clockwise or counter-clockwise
% (the default assumption is start at top-left corner and go clockwise) -
% the coords are rearranged if they were selected counter-clockwise
if coordBoard(2,1) < coordBoard(4,1) % compare x-values of p2 and p4
    coordBoard([1 2 3 4],:) = coordBoard([1 4 3 2],:);
end

% Perform transform and paste onto board
bookVertices = [0, size(imgBookCrop,1); 0, 0; size(imgBookCrop,2), 0;
        size(imgBookCrop,2), size(imgBookCrop,1)];
tform = fitgeotrans(bookVertices, coordBoard, 'projective');
[imgBookCropMod, RBookCrop] = imwarp(imgBookCrop, tform);
bwBook = roipoly(imgBookCropMod, coordBoard(:,1) ...
        -RBookCrop.XWorldLimits(1), coordBoard(:,2) ...
        -RBookCrop.YWorldLimits(1));
bwBoard = ~roipoly(imgBoard, coordBoard(:,1), coordBoard(:,2));
RBoard = imref2d(size(bwBoard));
imgBoardMask = bsxfun(@times, imgBoard, cast(bwBoard, 'like', imgBoard));
imgBookModMask = bsxfun(@times, imgBookCropMod, ...
        cast(bwBook, 'like', imgBookCropMod));
imgBoardFinal(:,:,1) = imfuse(imgBoardMask(:,:,1), RBoard, ...
        imgBookModMask(:,:,1), RBookCrop, 'diff');
imgBoardFinal(:,:,2) = imfuse(imgBoardMask(:,:,2), RBoard, ...
        imgBookModMask(:,:,2), RBookCrop, 'diff');
imgBoardFinal(:,:,3) = imfuse(imgBoardMask(:,:,3), RBoard, ...
        imgBookModMask(:,:,3), RBookCrop, 'diff');
imshow(imgBoardFinal);
imwrite(imgBoardFinal, 'imgOutputs\Prob5\imgBookBoard.jpg');

% ======================================================================= %
```

IMG_0080	board
	

BookBoard