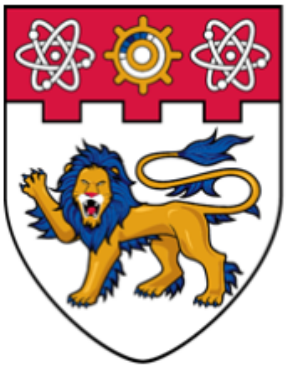

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4003 Computer Vision

School of Computer Science and Engineering

AY23/24 Semester 1

Lab 1

Lee Yew Chuan Michael
U2021372J
October 6, 2023

Contents

1 Contrast Stretching

Figure 1: Original mrt-train.jpg.

a Converting to Grayscale

Input:

```
Pc = imread('mrt-train.jpg');
whos Pc % 320x443x3 indicates 3 channels, meaning RGB image

P = rgb2gray(Pc); % convert to grayscale
whos P
```

Output:

```
Name         Size                 Bytes  Class    Attributes

Pc         320x443x3             425280  uint8

Name         Size                Bytes  Class    Attributes

P          320x443               141760  uint8
```

Lee Yew Chuan Michael's Comment:
Upon performing rgb2gray(Pc), the number of channels changed from 3 to 1. This indicates that the image has been converted to grayscale.

b View Image

Input:

```
imshow(P)
```

Output:

Figure 2: Grayscale mrt-train.jpg.

c Minimum and Maximum Intensities

Input:

```
min(P(:)), max(P(:)) % Check the min & max intensities
```

Output:

```
ans =

  uint8

   13

ans =

  uint8

   204
```

Lee Yew Chuan Michael's Comment:
The minimum intensity present in the image is 13 and the maximum intensity present in the image is 204.

d Contrast Stretching

Input:

```
[min_inten, max_inten] = deal(double(min(P(:))), double(max(P(:))));
P2 = (255/(max_inten-min_inten)) * imsubtract(P,min_inten);

min(P2(:)), max(P2(:)) % Check image P2 min & max intensities
```

Output:

```
ans =

  uint8

   0

ans =

  uint8

   255
```

Lee Yew Chuan Michael's Comment:
After performing contrast stretching, when the min and max commands is run again, we can observe that P2 does indeed have the correct minimum intensity of 0 and the correct maximum intensity of 255.

e Redisplay Image

Input:

```
imshow(uint8(P2))
```

Output:

Figure 3: Contrast Stretched mrt-train.jpg.

Lee Yew Chuan Michael's Comment:
After performing contrast stretching, it can be observed that the quality of the image obtained has been enhanced. The contrast of the image has improved.

2 Histogram Equalization

a Image Intensity Histogram

a.1 Histogram of 10 Bins and 256 bins

Input:

```
imhist(P,10); % histogram of P using 10 bins
imhist(P,256); % histogram of P using 256 bins
```

Output:

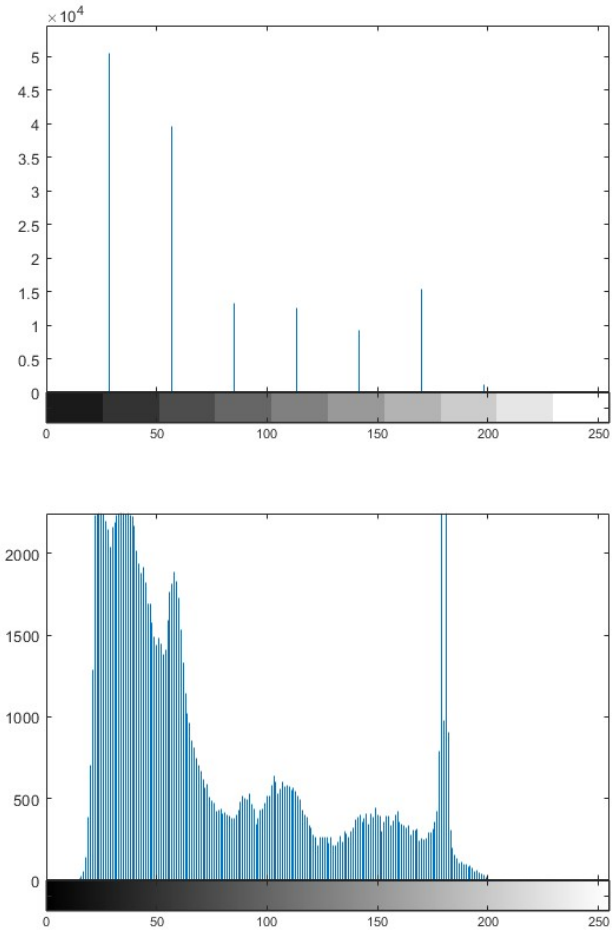

Figure 4: **Top:** Image intensity histogram of P using 10 bins. **Bottom:** Image intensity histogram of P using 256 bins.

a.2 Question

What are the differences?
The histogram using 10 bins contains more pixels in each bin as compared to the histogram using 256 bins. As a result, the histogram using 10 bins does not capture as much detail compared to the histogram using 256 bins. For example, the spike for gray level = 179 and gray level = 181 can only be seen in the histogram using 256 bins. The histogram with 10 bins can thus only display overall trend, and is less detailed compared to the histogram using 256 bins.

b Histogram Equalization

b.1 Re-display Image Intensity Histogram

Input:

```
P3 = histeq(P,255); % histogram equalization
imhist(P3,10); % histogram of P3 using 10 bins
imhist(P3,256); % histogram of P3 using 256 bins
```

Output:

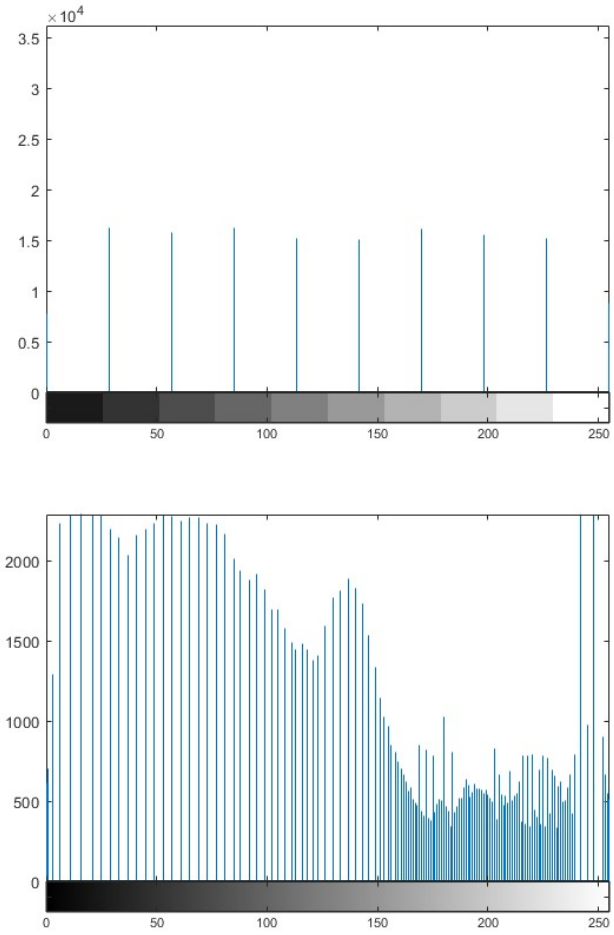

Figure 5: **Top:** Image intensity histogram of P3 using 10 bins. **Bottom:** Image intensity histogram of P3 using 256 bins.

b.2 Question

Are the histograms equalized?
The histograms for P3 using both 10 bins and 256 bins are equalized. However, the histogram with 256 bins is not as well equalized compared to the histogram with 10 bins.

Similarities:
From Figure. 5, we can see that for both histograms, the pixels are considerably more evenly distributed compared to the histograms corresponding to P; which was before equalization (Figure. 4). Additionally, for both histograms, bins which were previously empty now have pixels.

Differences:
While the histograms for P3 using 10 bins is flat (uniform) and well distributed, the histogram using 256 bins is still rather uneven and nonuniform. This indicates that the number of pixels in each bin when 10 bins were used were almost the same. On the other hand, when 256 bins were used, the number of pixels in each bin were uneven. The histogram using 10 bins was closer to an ideal case of equalization compared to the histogram using 256 bins.

The resultant image after histogram equalization is provided below.

Figure 6: Equalized mrt-train.jpg.

c Rerun Equalization

Input:

```
P3_rerun = histeq(P3,255); % histogram equalization
imhist(P3_rerun,10); % histogram of P3_rerun using 10 bins
imhist(P3_rerun,256); % histogram of P3_rerun using 256 bins
```

Output:

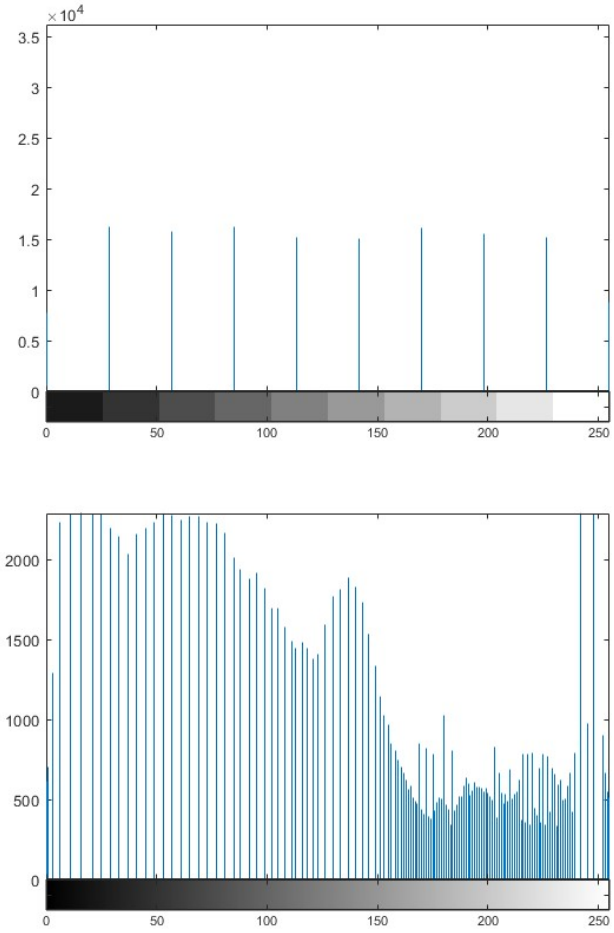

Figure 7: **Top:** Image intensity histogram of P3_rerun using 10 bins. **Bottom:** Image intensity histogram of P3_rerun using 256 bins.

c.1 Question

Does the histogram become more uniform?
There is no change observed. The histogram does not become more uniform.

Give suggestions as to why this occurs
In histogram equalization, repeating the algorithm again would not change results, as it is idempotent. After running histogram equalization once, the distribution is already optimized. Thus, running histogram equalization again will yield no change as it will just keep redistributing the pixel intensities to the same target distribution.

3 Linear Spatial Filtering

a Generate the following filters

Input:

```
x = -2:2; % [-2, -1, 0, 1, 2] 5x5 kernel
y = -2:2; % [-2, -1, 0, 1, 2] 5x5 kernel
[X,Y] = meshgrid(x,y);
% filter for 2.3.a.(i)
sigma_square = 1.0^2; % sigma square for i
e_term_one = exp(-1 * ((X.*X) + (Y.*Y)) / (2*(sigma_square)));
filter_one = (e_term_one) / (2*pi*sigma_square);
filter_one = filter_one ./ sum(filter_one(:));
sum(filter_one(:)) % sanity check to add up to 1
mesh(filter_one)

% filter for 2.3.a.(ii)
sigma_square = 2.0^2; % sigma square for ii
e_term_two = exp(-1 * ((X.*X) + (Y.*Y)) / (2*(sigma_square)));
filter_two = (e_term_two) / (2*pi*sigma_square);
filter_two = filter_two ./ sum(filter_two(:));
sum(filter_two(:)) % sanity check to add up to 1
mesh(filter_two)
```

Output:

```
ans =

     1

ans =

    1.0000
```

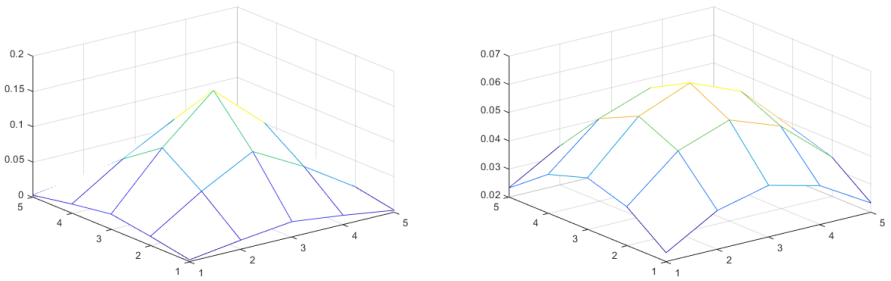

Figure 8: **Left:** Filter using $\sigma = 1.0$. **Right:** Filter using $\sigma = 2.0$.

b View Image

Input:

```
imshow('lib-gn.jpg');
```

Output:

Figure 9: Library with additive Gaussian noise.jpg.

c Filtering Gaussian Noise

Input:

```
original_img = imread('lib-gn.jpg');
convolved_one = uint8(conv2(original_img, filter_one));
imshow(convolved_one);

convolved_two = uint8(conv2(original_img, filter_two));
imshow(convolved_two);
```

Output:

Figure 10: **Top:** Filtered image using filter with $\sigma = 1.0$. **Bottom:** Filtered image using filter with $\sigma = 2.0$.

c.1 Question

How effective are the filters in removing noise?
While both filters do show their effectiveness in removing Gaussian noise, the filter using $\sigma = 2.0$ removes more noise compared to the filter using $\sigma = 1.0$.

What are the trade-offs between using either of the two filters, or not filtering the image at all?
The trade-offs between the 2 filters would be that as more noise is removed from the image (Higher σ), more details are lost. As in Figure. 10, less noise is present in the image when $\sigma = 2.0$ is used compared to when $\sigma = 1{,}0$. However, at the same time, the image using $\sigma = 2.0$ appears more blurred compared to the image using $\sigma = 1.0$. More details such as edges are lost. Thus, the higher the σ value, the more noise is filtered, but the blurrier the image is and the more details are lost.

In the case when no filters are used at all, while we would not lose details and the image would not get blurry, we would still have a noisy image, which is clearly undesirable.

d View Image

Input:

```
imshow('lib-gn.jpg');
```

Output:

Figure 11: Library with additive speckle noise.jpg.

e Filtering Speckle Noise

Input:

```
original_img_sp = imread('lib-sp.jpg');
convolved_one = uint8(conv2(original_img_sp, filter_one));
imshow(convolved_one);

convolved_two = uint8(conv2(original_img_sp, filter_two));
imshow(convolved_two);
```

Output:

Figure 12: **Top:** Filtered image using filter with $\sigma = 1.0$. **Bottom:** Filtered image using filter with $\sigma = 2.0$.

e.1 Question

How effective are the filters in removing noise?
Both filters do show their capability of filtering speckle noise to a certain extent. However, speckle noise after filtering can still be seen in both images. This leftover noise is more evident in the image filtered with the filter using $\sigma = 1.0$ compared to the image filtered with the filter using $\sigma = 2.0$.

What are the trade-offs between using either of the two filters, or not filtering the image at all?
Similar to the case with filtering Gaussian noise, the trade-off is that as more noise is removed from the image, more details are lost. As in Figure. 12, the image using $\sigma = 2.0$ appears more blurred compared to the image using $\sigma = 1.0$. Despite the presence of less noise, more details such as edges are lost. Thus, the higher the σ value, the more noise is filtered, but the more details are lost.
Despite this, the images obtained after processing are more desirable compared to the unfiltered noisy image as in Figure. 11.

Are the filters better at handling Gaussian noise or speckle noise?
As seen from Figure. 12, given that speckle noise is still clearly observed after applying the filters on the original image (Figure. 11), we can conclude that the filters are better at handling Gaussian noise. This can be attributed to the fact that the neighbouring noise in speckle noise are considerably different from the centre pixel. Thus, making Gaussian filters, which apply non-zero weights to surrounding pixels ineffective when dealing with speckle noise.

4 Median Filtering

a Filtering Gaussian Noise

Input:

```
original_img_gn = imread('lib-gn.jpg');
median_filtered_3_gn =  medfilt2(original_img_gn, [3,3]); % 3x3 kernel
imshow(median_filtered_3_gn);

median_filtered_5_gn =  medfilt2(original_img_gn, [5,5]);  % 5x5 kernel
imshow(median_filtered_5_gn);
```

Output:

Figure 13: **Top:** Median filtered image using filter with [3,3]. **Bottom:** Median filtered image using filter with [5,5]

a.1 Question

How effective are the filters in removing noise?
The filter with [3x3] kernel does not appear to be very effective in removing Gaussian noise. This is compared to the [5x5] kernel which removes considerably more Gaussian noise.

What are the trade-offs between using either of the two filters, or not filtering the image at all?
The trade-off would also be that as more noise is removed from the image, the image gets blurrier, and the more details are lost.
Despite this, the images obtained after processing are more desirable compared to the unfiltered noisy image as in Figure. 9.

b Filtering Speckle Noise

Input:

```
original_img_sp = imread('lib-sp.jpg');
median_filtered_3_sp =  medfilt2(original_img_sp, [3,3]); % 3x3 kernel
imshow(median_filtered_3_sp);

median_filtered_5_sp =  medfilt2(original_img_sp, [5,5]); % 5x5 kernel
imshow(median_filtered_5_sp);
```

Output:

Figure 14: **Top:** Median filtered image using filter with [3x3]. **Bottom:** Median filtered image using filter with [5x5]

b.1 Question

How effective are the filters in removing noise?
Both filters appear to be rather effective in filtering speckle noise. As seen in Figure. 14, both images using kernel size [3x3] and [5x5] have a large part of the noise removed.

What are the trade-offs between using either of the two filters, or not filtering the image at all?
The trade-off would also be that as kernel size increases, more noise is removed from the image. However, the increase in kernel size also results in the image getting blurrier and losing more details. However, compared to when Gaussian filters were used, edge information is now better preserved.

Are the filters better at handling Gaussian noise or speckle noise?
Median filters are considerably better at handling speckle noise. Even with kernel size of [3x3], results obtained upon filtering were better than when Gaussian filter of $\sigma = 2.0$ was used. Additionally, the poor results obtained in Figure. 13 indicates that median filters are not very suited for filtering Gaussian noise.
While Gaussian filters are better at filtering Gaussian noise, Median filters are better at filtering speckle noise.

c Comparing Gaussian filters and Median filters

c.1 Question

How does Gaussian filtering compare with median filtering in handling the different types of noise?
Gaussian filtering is better than Median filtering at handling Gaussian noise in images. However, median filtering is better than Gaussian filtering at handling speckle noise in images.

What are the tradeoffs?
While Gaussian filters are more effective at removing Gaussian noise, it introduces blurring to the image. Median filters on the other hand are better at removing speckle noise, and can preserve edge information better. However, it be noted that as kernel size increases, while more noise is removed, more edge information gets lost.

5 Suppressing Noise Interference Patterns

a View Image

Input:

```
imshow('pck-int.jpg')
```

Output:

Figure 15: pck-int.jpg.

b Display power spectrum with fftshift

Input:

```
original_img_pck = imread('pck-int.jpg');
F = fft2(original_img_pck); % fast FT; gets kernel coefficients
S = abs(F).^2; %calculate power spectrum
imagesc(fftshift(S.^0.1));
colormap('default');
```

Output:

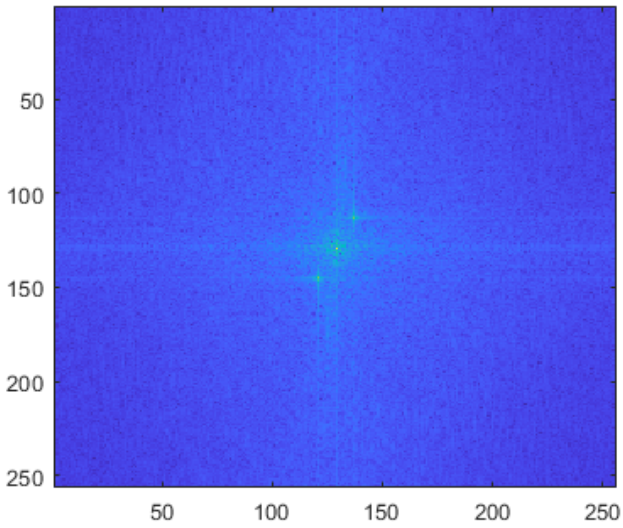

Figure 16: Power spectrum with fftshift

c Display power spectrum without fftshift

Input:

```
imagesc(S.^0.1);
colormap('default');
```

Output:

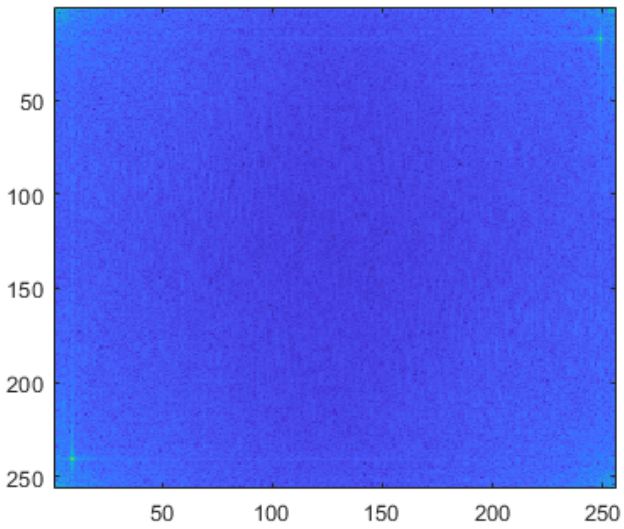

Figure 17: Power spectrum without fftshift

Lee Yew Chuan Michael's Comment: The location of the peaks are found at (9,241) and (249,17).

d Zeroing High Frequencies

Input:

```
x_1 = 9;
y_1 = 241;
x_2 = 249;
y_2 = 17;

F(y_1-2:y_1+2, x_1-2:x_1+2) = 0; % +-2 cos 5x5 kernel
F(y_2-2:y_2+2, x_2-2:x_2+2) = 0; % +-2 cos 5x5 kernel
S = abs(F).^2; %calculate power spectrum
imagesc(fftshift(S.^0.1));
colormap('default');
```

Output:

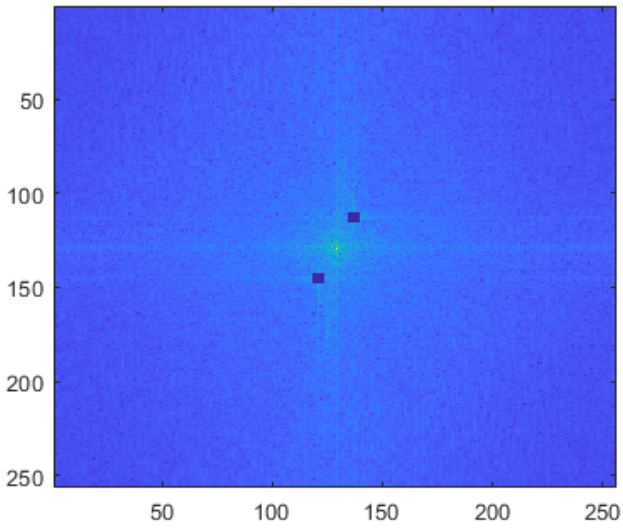

Figure 18: Power spectrum with zeroed peaks

e Compute the inverse Fourier transform

Input:

```
imshow(uint8(ifft2(F)));
```

Output:

Figure 19: Resultant image from inverse Fourier transform

e.1 Question

Comment on the result and how this relates to step (c)
The resultant image is much clearer than the original image. Most of the parallel lines in the middle of the image are removed. Much more details such as the human faces and the Chinese characters can be clearly identified from the image now.
Such a phenomenon is not unexpected. In part (c), we marked out coordinates which corresponded to high frequencies in the image. These peaks corresponds to regions that caused such parallel lines to appear in our image. By setting the contribution of these points to 0, we were able to remove the interference (parallel lines) when conversion back to the spatial domain was

done.
We also want to note that the zeroed out points were somewhat perpendicular to the direction of the parallel lines in Figure 15. This is consistent with the theory of Fourier transform, and contributed to the removal of some of the noise in the direction of the parallel lines.
However, noise near the top and bottom of the images are still quite clearly present. The parallel in the centre of the image can also still be faintly seen. These could be attributed to the fact that besides the 2 peaks that were zeroed out, there are still indications of high frequency areas isolated from the central mass. In particular, we can observe from the power spectrum plotted in Figure. 18 that there are light lines extending in both the horizontal and vertical directions from the 2 peaks.
These could be removed by introducing additional filtering. In particular, by zeroing out these horizontal and vertical lines.

e.2 Improve the Result

Here, we present our attempts at improving the results. As mentioned, we pad the horizontal and vertical lines extending from the 2 peaks. Two variants of this experiments were performed. One where the lines were only 1 pixel thick, and another where the lines were 3 pixels thick. It was reasoned that just as how increasing kernel size can improve noise removal, increasing the thickness of lines that were zeroed out could potentially give better solutions.

Input:

```
original_img_pck = imread('pck-int.jpg');
F = fft2(original_img_pck); % fast FT; gets kernel coefficients
x_1 = 9;
y_1 = 241;
x_2 = 249;
y_2 = 17;

% zero out line extending from peaks (Single pixel thick line)
F(:,x_1) = 0;
F(y_1,:) = 0;
F(:,x_2) = 0;
F(y_2,:) = 0;

S = abs(F).^2; %calculate power spectrum
imagesc(fftshift(S.^0.1));
colormap('default');
```

```
imshow(uint8(ifft2(F)));

%reread image: Now, increase the thickness of lines
original_img_pck = imread('pck-int.jpg');
F = fft2(original_img_pck); % fast FT; gets kernel coefficients

% zero out line extending from peaks (3 pixels thick line)
[w, h]=size(F);
F(:,x_1-1:x_1+1) = zeros(w,3);
F(y_1-1:y_1+1,:) = zeros(3,h);
F(:,x_2-1:x_2+1) = zeros(w,3);
F(y_2-1:y_2+1,:) = zeros(3,h);

S = abs(F).^2; %calculate power spectrum
imagesc(fftshift(S.^0.1));
colormap('default');
imshow(uint8(ifft2(F)))
```

Output:

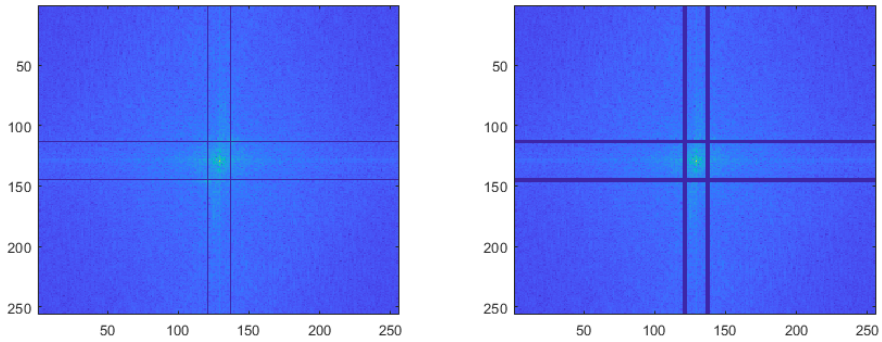

Figure 20:
Left: Power spectrum of image where lines zeroed out is 1 pixel thick.
Right: Power spectrum of image where lines zeroed out is 3 pixels thick

Figure 21: **Left:** Resultant image when lines zeroed out were 1 pixel thick. **Right:** Resultant image when lines zeroed out were 3 pixels thick

Both settings showed improvement compared to the image where only the 2 peaks were zeroed out. Based on the resultant images, the setting where lines zeroed out were 3 pixels thick does give better performance compared to when lines were only 1 pixel thick. Less parallels are observed when lines zeroed out were 3 pixels thick.

f Free the Primate

Figure 22: Original image of caged primate

As seen in the above image, the primate is being fenced in. Our goal is to clear as much of the fence away as possible. To perform this, we apply what we used earlier in part (d). That is, we display the power spectrum of the original image, and identify the coordinates of the peaks. In this image, 6 points were identified. Experiments using various setting when zeroing these point out were performed. In particular, when setting to zero, the size of the neighbourhood elements that were also set to zero at locations corresponding to the peaks were varied. Experiments using a 3x3, 5x5, 7x7, 9x9, and 11x11 sized neighbourhood were run.

Input:

```
original_primate = imread('primate-caged.jpg');
original_primate_gray = rgb2gray(original_primate);

F_primate = fft2(original_primate_gray); % fast FT

S = abs(F_primate).^2; % calculate power spectrum
imagesc(fftshift(S.^0.1));
colormap('default');
```

```
imagesc(S.^0.1);
colormap('default');

% zeroed coordinates
x1 = 11; y1 = 252;
x2 = 22; y2 = 248;
x3 = 246; y3 = 8;
x4 = 234; y4 = 12;
x5 = 235; y5 = 10;
x6 = 32; y6 = 244;

a = 4; % controls kernel size
F_primate(y1-a : y1+a, x1-a : x1+a) = 0;
F_primate(y2-a : y2+a, x2-a : x2+a) = 0;
F_primate(y3-a : y3+a, x3-a : x3+a) = 0;
F_primate(y4-a : y4+a, x4-a : x4+a) = 0;
F_primate(y5-a : y5+a, x5-a : x5+a) = 0;
F_primate(y6-a : y6+a, x6-a : x6+a) = 0;

S = abs(F_primate).^2; %calculate power spectrum
imagesc(S.^0.1);
colormap('default');

imshow(real(uint8(ifft2(F_primate))));
```

Output:

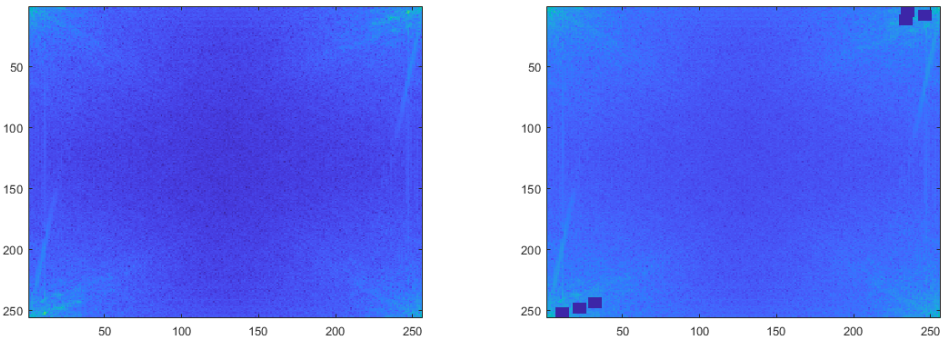

Figure 23: **Left:** Power spectrum of original image; **Right:** Power spectrum after zeroing points (points in this image were zeroed out in a 9x9 neighbourhood)

(a) Zero the 3x3 neighbourhood (b) Zero the 5x5 neighbourhood

(c) Zero the 7x7 neighbourhood (d) Zero the 9x9 neighbourhood

(e) Zero the 11x11 neighbourhood

Figure 24: Resultant images when varying size of neighbourhood elements to be zeroed out.

Upon varying the size of neighbouring elements to be zeroed out, we can see that the resultant image under the 9x9 setting performed the best. The setting using 11x11 showed to be over excessive, causing the image to lose too much detail. This is evidenced by the blurring of the image in (e) in Fig. 24. On the other hand, using too low settings like 3x3 and 5x5 resulted in subpar performance, the fence could still be clearly seen in (a) and (b) in Fig. 24. While using 7x7 does show reasonable results, the resultant image using 9x9 proves to be better.

6 Undoing Perspective Distortion of Planar Surface

a Display the Image

Input:

```
P = imread('book.jpg');
imshow(P);
```

Output:

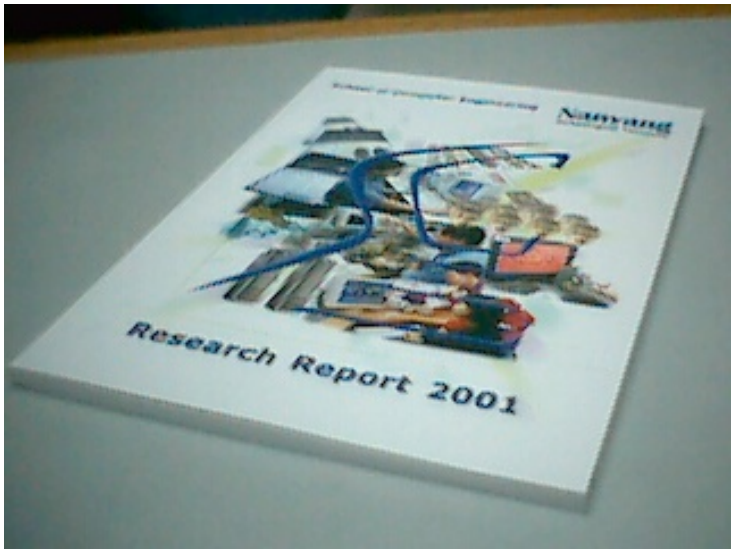

Figure 25: book.jpg.

b Find out the Location of 4 Corners of the Book

Input:

```
[X Y] = ginput(4)
X_A4 = [0; 210; 210; 0];
Y_A4 = [0; 0; 297; 297];
```

Output:

```
X =

  144.0000
  308.0000
  257.0000
    7.0000


Y =

   28.0000
   48.0000
  216.0000
  158.0000
```

Order which corners are measured:
Top Left: [144.0000, 28.0000], Top Right: [308.0000, 48.0000]
Bottom Right: [257.0000, 216.0000], Bottom Left: [7.0000, 158.0000]

c Setup the matrices

Input:

```
% Setup the matrix
A = [[X(1), Y(1), 1, 0, 0, 0, -X_A4(1)*X(1), -X_A4(1)*Y(1)];
    [0, 0, 0, X(1), Y(1), 1, -Y_A4(1)*X(1), -Y_A4(1)*Y(1)];
    [X(2), Y(2), 1, 0, 0, 0, -X_A4(2)*X(2), -X_A4(2)*Y(2)];
    [0, 0, 0, X(2), Y(2), 1, -Y_A4(2)*X(2), -Y_A4(2)*Y(2)];
    [X(3), Y(3), 1, 0, 0, 0, -X_A4(3)*X(3), -X_A4(3)*Y(3)];
    [0, 0, 0, X(3), Y(3), 1, -Y_A4(3)*X(3), -Y_A4(3)*Y(3)];
    [X(4), Y(4), 1, 0, 0, 0, -X_A4(4)*X(4), -X_A4(4)*Y(4)];
    [0, 0, 0, X(4), Y(4), 1, -Y_A4(4)*X(4), -Y_A4(4)*Y(4)];];

v = [X_A4(1); Y_A4(1); X_A4(2); Y_A4(2);
    X_A4(3); Y_A4(3); X_A4(4); Y_A4(4)];

u = A \ v; %computes u = A-1v
U = reshape([u;1], 3, 3)' % convert to the normal matrix form

% Verify correctness by transforming the original coordinates
w = U*[X'; Y'; ones(1,4)];
w = w ./ (ones(3,1) * w(3,:));
```

Output:

```
U =

    1.4758    1.5553 -256.0645
   -0.4546    3.7274  -38.9102
    0.0001    0.0053    1.0000

w =

         0  210.0000  210.0000         0
   -0.0000   -0.0000  297.0000  297.0000
    1.0000    1.0000    1.0000    1.0000
```

d Warp the image

Input:

```
T = maketform('projective', U');
P2 = imtransform(P, T, 'XData', [0 210], 'YData', [0 297]);
```

e Display the image

```
imshow(P2);
```

Output:

Figure 26: warped book.jpg.

e.1 Question

Is this what you expect?
The result obtained is as expected. The image is no longer distorted. We have successfully gotten from an originally slanted view of a book to one with a frontal view.
Comment on the quality of the transformation and suggest reasons.
However, the image quality across the image is not uniform. As can be seen from the image in Figure. 26, the image quality at the bottom right is rather sharp and good. However, as it moves towards the top left and top right, image quality decreases. The image quality at the top left is significantly worse compared to the rest of the image. These observations

can be attributed to the fact that to begin with, the top left corner of the book in the original image is already not of good quality. This was due to it being at an angle and positioned further away from the camera when the picture was taken. Further reasons for the poor quality could be due to the low resolution of the original image.

f Display the image

To identify the pink rectangle, a few steps were taken.

First, it was noticed that a large part of this area to be identified was orange in colour. Hence, a mask to extract varying shades of orange pixels from the image was generated. This mask is controlled by the range of values indicated in lowerOrange and UpperOrange. After careful tuning, the values as provided in the code snippet below were obtained.

```
originalImage = imread('book.jpg');

% Convert from RGB to HSV color space
hsvRightHalf = rgb2hsv(originalImage);

% Define the HSV range for the orange color
% Adjusted HSV range as needed to captures more orange shades
lowerOrange = [0.01, 0.1, 0.1];
upperOrange = [0.06, 0.8, 1.0];

% mask out the orange
orangeMask = (hsvRightHalf(:,:,1) >= lowerOrange(1) ...
    & hsvRightHalf(:,:,1) <= upperOrange(1)) ...
    & (hsvRightHalf(:,:,2) >= lowerOrange(2) ...
    & hsvRightHalf(:,:,2) <= upperOrange(2)) ...
    & (hsvRightHalf(:,:,3) >= lowerOrange(3) ...
    & hsvRightHalf(:,:,3) <= upperOrange(3));

filteredImage = originalImage;
filteredImage(repmat(~orangeMask, [1, 1, 3])) = 0;  % perform masking

imshow(filteredImage);
```

Figure 27: Partially filtered orange pixels.jpg.

Upon applying the mask 'OrangeMask' to the original image, the image above is obtained. As can be seen from the above image, there is 'noise' around. This noise corresponds to other sections of the image that had orange pixels too. As these areas do not correspond to the screen we are interested in, we term them as noise. We can observe that the screen (our main object of interest) takes up the largest area of orange pixels.

Based on this observation, another mask was applied to the above image. But this time round, putting a restriction such that only sections of orange pixels that are more than a certain area are taken into account. This threshold ensures that we would retain only the screen, and not the other 'noisy' orange pixels that do not correspond to the screen.

```
% Filter out small noise regions in the mask using bwareaopen
minOrangeRegionArea = 21;  % Adjust as needed
filteredMask = bwareaopen(orangeMask, minOrangeRegionArea);

% Calculate the area of orange regions in the filtered mask
orangeRegionStats = regionprops(filteredMask, 'Area');

% Initialize the output image with the original image
outputImage = originalImage;

% Create a binary mask to keep only the dense orange regions
denseOrangeMask = false(size(filteredMask));
for i = 1:numel(orangeRegionStats)
    if orangeRegionStats(i).Area >= minOrangeRegionArea
```

```
        % Mark the region as dense in the dense orange mask
        denseOrangeMask = denseOrangeMask | (filteredMask == i);
    end
end

outputImage(repmat(~denseOrangeMask, [1, 1, 3])) = 0;  % apply threshold
imshow(outputImage);
```

Figure 28: Partially filtered orange pixels with threshold on area.jpg.

As can be seen from the above image, much noise is removed. However, small areas of noise (larger area) that do not correspond to the screen still exist. Additionally, a part of the screen has been separated. We need to join the separated parts of the screen together. This was done by dilating (expanding) the mask obtained in the previous section. In doing so, we can get a bigger surrounding area which would hopefully fill up the parts of the screen that were previously eliminated by our masking.

```
% Use dilation to expand the mask across neighboring pixels
se = strel('rectangle', [5, 8]);  % Adjust the disk size as needed
dilatedMask = imdilate(filteredMask, se);

combinedMask = denseOrangeMask | dilatedMask;
% Apply the dense orange mask to the output image
% Set non-orange pixels to black
outputImage(repmat(~combinedMask, [1, 1, 3])) = 0;

imshow(outputImage);
```

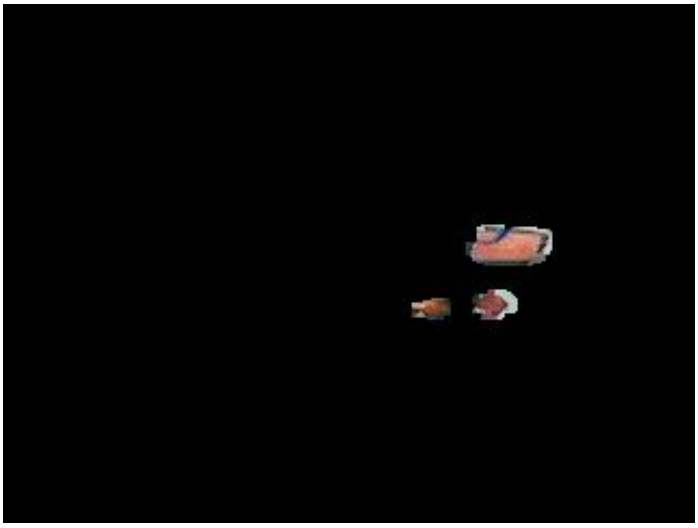

Figure 29: Partially filtered orange pixels dilated.jpg.

As can be seen from the above image, after dilation, the parts of the screen that were missing are now regained. The whole screen is now visible. However, as a result of this dilation, the noisy areas are also now enlarged. To remove these noisy areas and retain just the screen, a similar threshold on the area of pixels that was done previously can be applied too.

```
% Label connected components in the binary image
cc = bwconncomp(outputImage);

% Define the minimum required pixel count threshold
minPixelCount = 1000;  % Adjust as needed

% Mask to keep only connected components meeting the threshold
filteredComponentMask = false(size(outputImage));

% Iterate through connected components
for i = 1:cc.NumObjects
    % Calculate the size (number of pixels) of the connected component
    componentSize = numel(cc.PixelIdxList{i});

    % Check if the component size meets the threshold
    if componentSize >= minPixelCount
        % If it meets the threshold, mark it in the mask
        filteredComponentMask(cc.PixelIdxList{i}) = true;
    end
end

% Apply the filtered component mask to the binary image
```

```
filteredBinaryImage = outputImage;
filteredBinaryImage(~filteredComponentMask) = 0;

imshow(filteredBinaryImage);
```

Output:

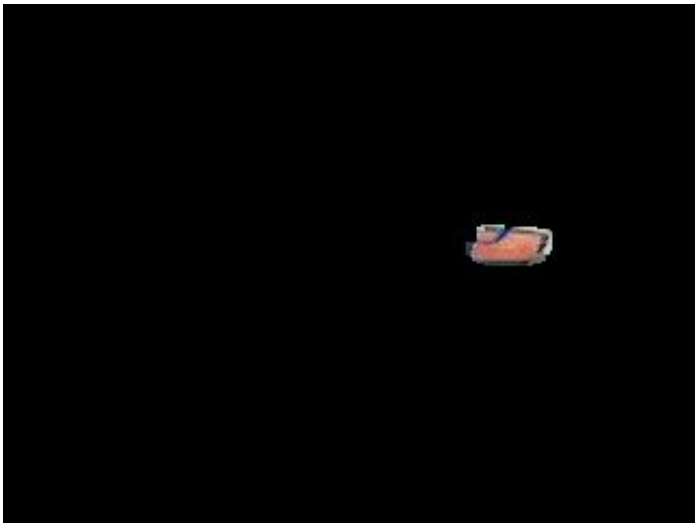

Figure 30: Identified pink screen.jpg.

Using the methods described above, we can obtain and identify just the big rectangular pink area.

Figure 31: Closer view of identified pink screen.jpg.

A closer zoomed in view of the image is provided for better clarity. As the screen in the original image was of poor resolution and small in size, poor quality of image can be observed in the image above.