Lab 1 Report

1. Introduction

In this lab, several image processing techniques have been explored using MATLAB. Firstly, point processing operations including contrast stretching and histogram equalization have been examined to enhance the overall quality of images. Next, the efficiency of spatial filtering techniques such as Gaussian filters and median filters for noise removal have been investigated. Besides, Fourier transform have been experimented to remove noise interference patterns which are not able to be removed by normal spatial filtering techniques. Finally, an experiment on undoing perspective distortion have been carried out for better understanding of imaging geometry.

2. Experiments

2.1 Contrast Stretching

1. Input the image into a MATLAB matrix variable, then convert the RGB image to grayscale and check the minimum and maximum intensities.

```
img = imread("images/mrttrainbland.jpg");
whos img
gray = rgb2gray(img);
min(gray(:))
max(gray(:))
```

   The minimum intensity is 13 and the maximum intensity is 204

2. Implement contrast stretching and check again the minimum and maximum intensities.

```
function new_img = contrast_stretching(img)
    img = im2double(img);
    min_val = min(img(:));
    max_val = max(img(:));
    new_img = (img - min_val) / max(max_val - min_val, eps);
    new_img = im2uint8(new_img);
end

new_gray = contrast_stretching(gray);
min(new_gray(:))
max(new_gray(:))
```

   The minimum intensity is 0 and the maximum intensity is 255

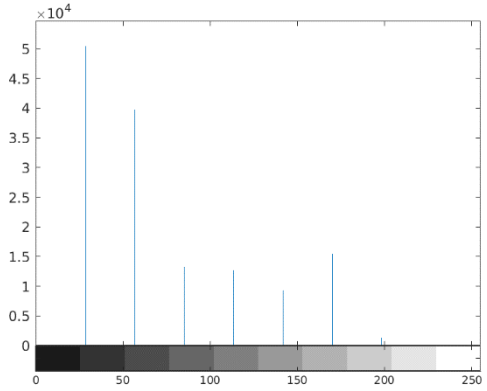
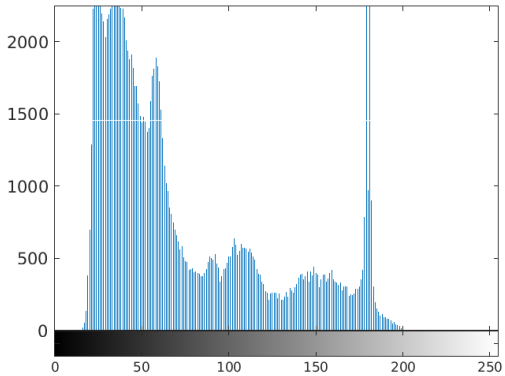
3. Compare the image before and after contrast stretching.

Original Image	After Contrast Stretching
	

2.2 Histogram Equalization

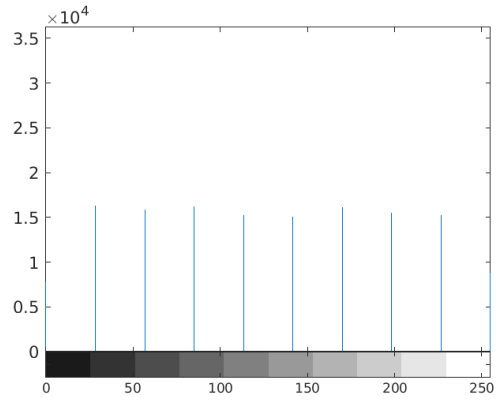
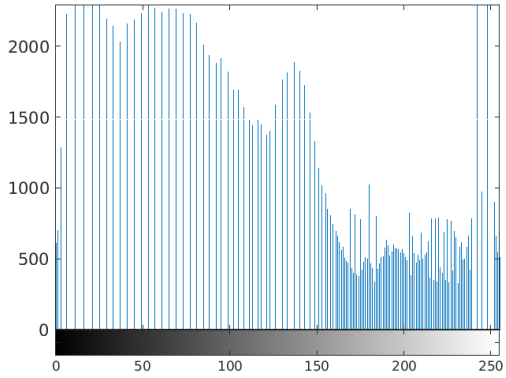
1. Display the histogram of P using 10 bins and 256 bins, what are the differences?

   In the histogram with 256 bins, the number of pixels for every gray level can be obtained, which gave us a more detailed distribution of the intensity of pixels across the whole image. Whereas in the 10-bin histogram, gray levels were grouped into bins and thus, the number of pixels for every bin was higher. The peak in between 150 and 200 gray level could only be observed in the 256-bin histogram, because the 10-bin histogram grouped the peak count with its neighboring intensities which were having much lower counts. Both histograms suggested that the image comprised mainly darker pixels as majority of the pixels lied below 100 gray level.

10 bins	256 bins
	

2. Carry out histogram equalization and redisplay the histograms with 10 and 256 bins. Are the histogram equalized? What are the similarities and differences between the latter 2 histograms?

   After histogram equalization, the range of intensity covered by both 10-bin and 256-bin histogram became wider, spanned across the whole range of intensity level from 0 – 255. The 10-bin histogram became more uniformly distributed as compared to before which was heavily right-skewed. The 256-bin histogram looked rugged due to the nature of histogram equalization which tried to move every pixel in an entire bin to the next bin when the total number of pixels up to that bin originally exceeded the total number of pixels that can be hold up to that bin after histogram equalization, thus leaving empty bins in between bins with large number of pixels.

10 bins	256 bins
	

3. Rerun the histogram equalization, does the histogram become more uniform? Give suggestions as to why this occurred.

   There were neither changes to the histogram, nor changes to the image. The result after the first histogram equalization was guaranteed to be optimum, rerun the algorithm would not create better results.

2.3 Linear Spatial Filtering

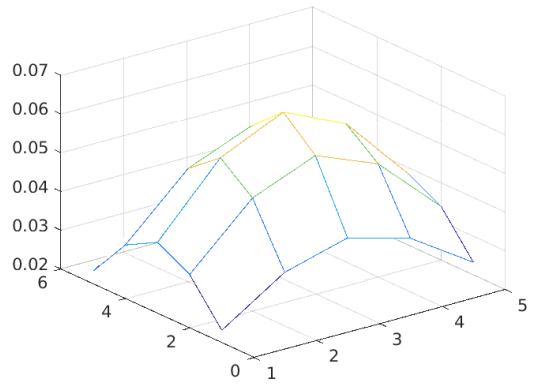
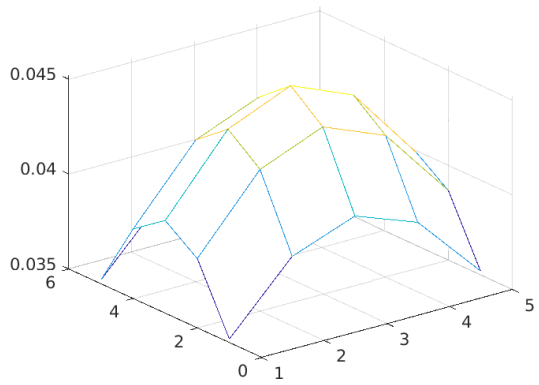
1. Generate the following filters and normalize the filters such that the sum of all elements equals to 1.0. View the filters as 3D graphs using **mesh** function.

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

   a. y and x dimensions are 5, and $\sigma = 1.0$
   b. y and x dimensions are 5, and $\sigma = 2.0$

```
function h = gaussian_filter(width, height, sigma)
    assert(mod(width, 2) == 1);
    assert(mod(height, 2) == 1);
    assert(sigma > 0);
    x = -1:2 / (width - 1):1;
    y = -1:2 / (height - 1):1;
    [X, Y] = meshgrid(x, y);
    h = (X .^ 2 + Y .^ 2) / (2 * sigma ^ 2);
    h = exp(-h) / (2 * sigma ^ 2 * pi);
    h = h / sum(h(:));
end

 h1 = gaussian_filter(5, 5, 1);
 h2 = gaussian_filter(5, 5, 2);
 figure("Name", "filt1");
 mesh(h1);
 figure("Name", "filt2");
 mesh(h2);
```

$\sigma = 1.0$	$\sigma = 2.0$
	

2. Read the image “ntu-gn.jpg” which consists of additive Gaussian noise. Filter the image using filters created above and display the resultant images. How effective are the filters in removing noise? What are the trade-offs between using either of the 2 filters, or not filtering the image at all?

```
% gaussian noise image
img = imread("images/ntugn.jpg");
img = im2double(img);

filt1 = conv2(img, h1, "same");
filt2 = conv2(img, h2, "same");
```

Original Image (ntu-gn.jpg):

$\sigma = 1.0$	$\sigma = 2.0$
	

After filtering the original image with Gaussian averaging filters, the Gaussian noise in the images became less obvious. However, the images became blurrier and the edges became less distinct. The larger the sigma of Gaussian averaging filter, the more effective the removal of Gaussian noise and the blurrier the image. Hence, Gaussian filter worked best for images with little edges. Since the above image has many edges, applying Gaussian filter would reduce the Gaussian noise, but with the trade-off of many blurry edges, thus might not be yielding a better result compared to the original image depending on individual's preferences.

3. Read the image "ntu-sp.jpg" which consists of additive speckle noise. Repeat the step above. Are the filters better at handling Gaussian noise or speckle noise?

```
% speckle noise image
img = imread("images/ntusp.jpg");
img = im2double(img);

filt1 = conv2(img, h1, "same");
filt2 = conv2(img, h2, "same");
```

Original image (ntu-sp.jpg):

$\sigma = 1.0$	$\sigma = 2.0$
	

After applying Gaussian averaging filter on the image with speckle noise, the intensity of white speckles reduced but the radius of speckles increased. The speckle noise can still be seen even though the images have been blurred. Hence, we can conclude that Gaussian averaging filters can handle Gaussian noise better than speckle noise, with the trade-off of blurring edges.

2.4 Median Filtering

1. Repeat steps 2 and 3 in Section 2.3 but using median filtering with different neighborhood sizes of 3×3 and 5×5. How does Gaussian filtering compare with median filtering in handling different types of noise? What are the tradeoffs?

```
% gaussian noise image
img = imread("images/ntugn.jpg");
img = im2double(img);

filt1 = medfilt2(img, [3, 3], "symmetric");
filt2 = medfilt2(img, [5, 5], "symmetric");
```

```
% speckle noise image
img = imread("images/ntusp.jpg");
img = im2double(img);

filt1 = medfilt2(img, [3, 3], "symmetric");
filt2 = medfilt2(img, [5, 5], "symmetric");
```

Filter Size	3×3	5×5
Gaussian Noise		
Speckle Noise		

Applying median filter could also help in removing Gaussian Noise but is not as effective as Gaussian average filtering. The larger median filter was slightly more effective in removing Gaussian noise compared to the smaller median filter. Whereas for speckle noise, both median filters managed to remove most of the speckle noise in the image.

In terms of blurriness of edges, using median filter could help to preserve edges better compared to Gaussian averaging filter. As the median filter became larger, the edges became blurrier, which can be clearly seen by comparing the edges of the trees.

Hence, we can conclude that Gaussian averaging filter is more effective in removing Gaussian noise, but with the trade off of blurrier edges. Median filters can effectively remove speckle noise and also perform better in preserving the edges in the image.

2.5 Suppressing Noise Interference Patterns

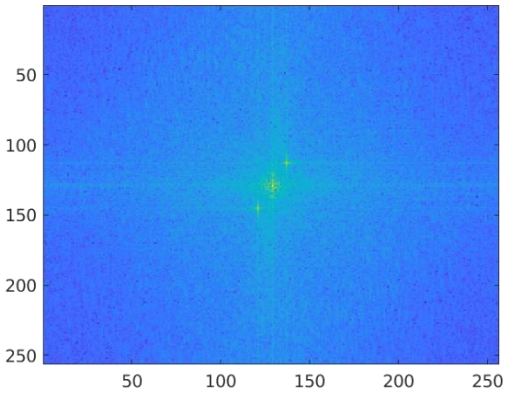
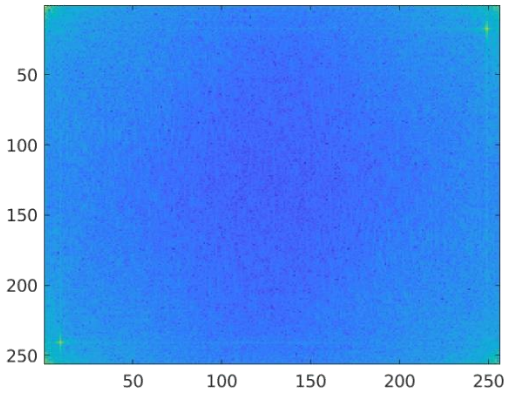
1. Read in the image “pck-int.jpg”

2. Obtain the Fourier transform of the image and display the spectrum.

```
F = fft2(img);
S = abs(F);

% figure("Name", "fourier-fftshift");
imagesc(fftshift(S .^ 0.1));
colormap("default");

% figure("Name", "fourier");
imagesc(S .^ 0.1);
colormap("default");
```

With **fftshift**	Without **fftshift**
	

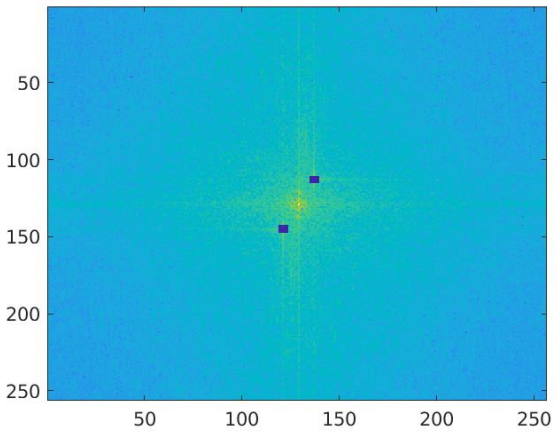
3. Notice that there are 2 distinct and symmetric frequency peaks that are isolated from the central mass, these frequency components correspond to the interference pattern of the image. Set to zero the 5×5 neighborhood elements at the locations of the peaks, recompute the power spectrum and display it.

```
function F = mask_points(F, neigh, x, y, mul)
    assert(length(x) == length(y));
    height = size(F, 1);
    width = size(F, 2);
    offset = floor(neigh / 2);
    for i = 1:length(x)
        x1 = max(1, round(x(i)) - offset);
        x2 = min(width, round(x(i)) + offset);
        y1 = max(1, round(y(i)) - offset);
        y2 = min(height, round(y(i)) + offset);
        F(y1:y2, x1:x2) = mul * F(y1:y2, x1:x2);
    end
end

x = [9, 249];
y = [241, 17];
new_F = F * 1;
new_F = mask_points(new_F, 5, x, y, 0);
new_S = abs(new_F);

% figure("Name", "fourier-modified");
imagesc(fftshift(new_S .^ 0.1));
colormap("default");
```

The peaks are identified at coordinates (9, 241) and (249, 17) in power spectrum of the image.



4. Compute the inverse Fourier transform and display the resultant image after removing the peaks. Comment on the result.

```
new_img = ifft2(new_F);
new_img = im2uint8(new_img);
figure("Name", "pckint-modified");
imshow(new_img);
```

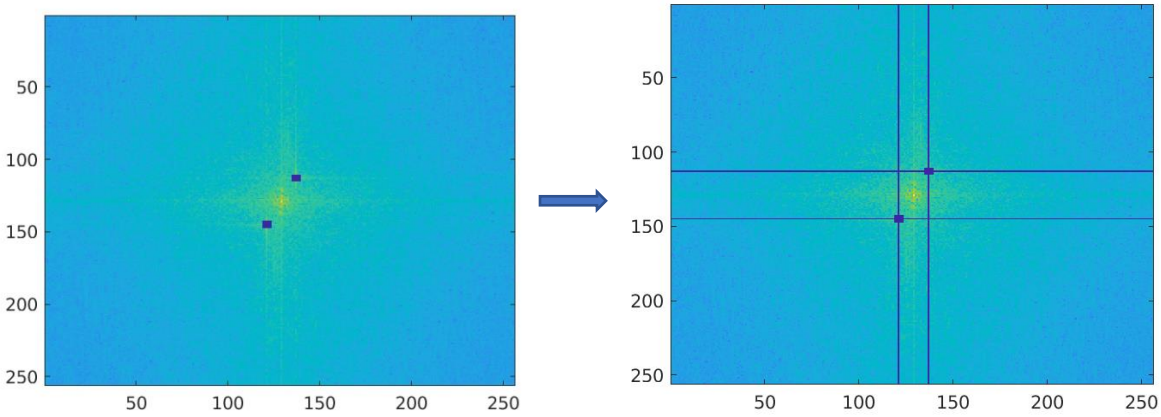
After removing the peaks of the Fourier transform, the interference patterns were reduced. The 2 peaks in the power spectrum of the original image implied that there were some high frequency components in the image, in this case would be the dominant diagonal parallel lines. Setting them to zero would therefore remove the high frequency components, thus reducing the intensity of those parallel lines.

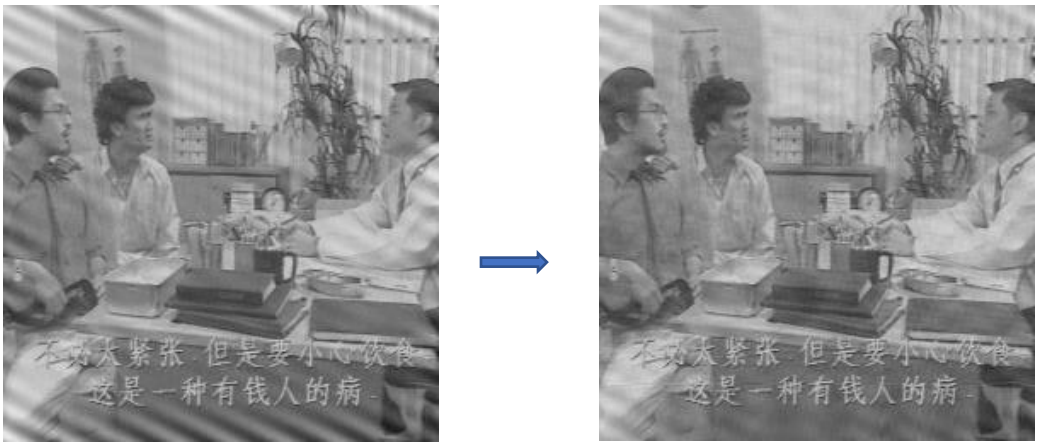
5. Can you suggest any way to improve this?

From the power spectrum of the image, we can see that there were horizontal and vertical lines which implied the existence of other high frequency components as well. Thus, these lines were set to zero and the intensity of the parallel diagonal lines were further reduced.

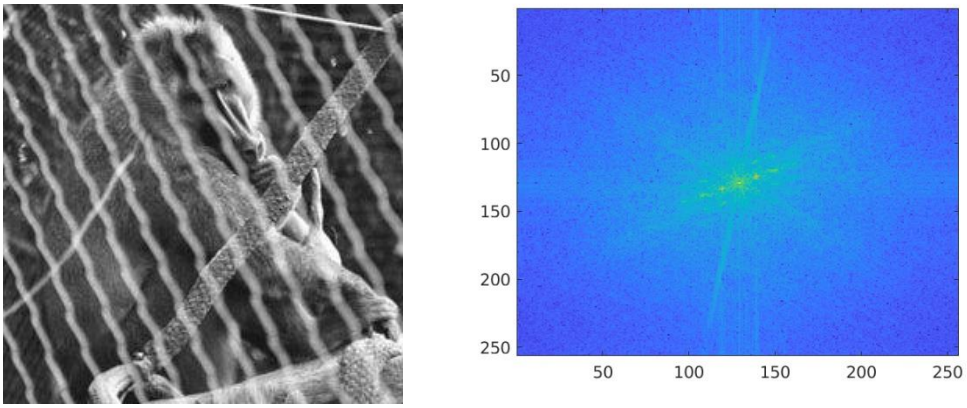
```
function F = mask_vhlines(F, x, y, mul)
    for i = 1:length(x)
        F(:, x(i)) = F(:, x(i)) * mul;
    end
    for i = 1:length(y)
        F(y(i), :) = F(y(i), :) * mul;
    end
end
```

```
new_F2 = new_F * 1;
new_F2 = mask_vhlines(new_F2, x, y, 0);
new_S2 = abs(new_F2);
```



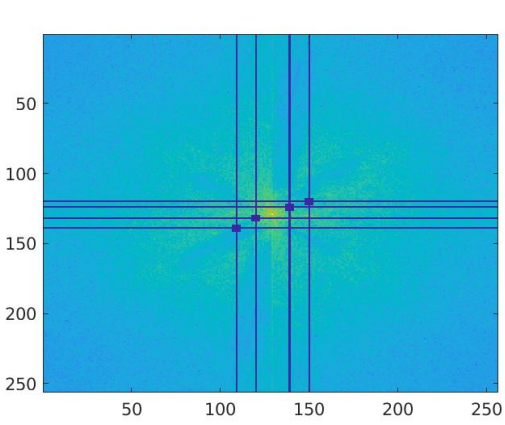


6. Download the image “primate-caged.jpg” and try to filter out the fence.



The left image is the original image and the right image is its power spectrum

After setting the peaks and some vertical and horizontal lines in the power spectrum to zero, the resultant image after inverse Fourier transform is displayed as below.

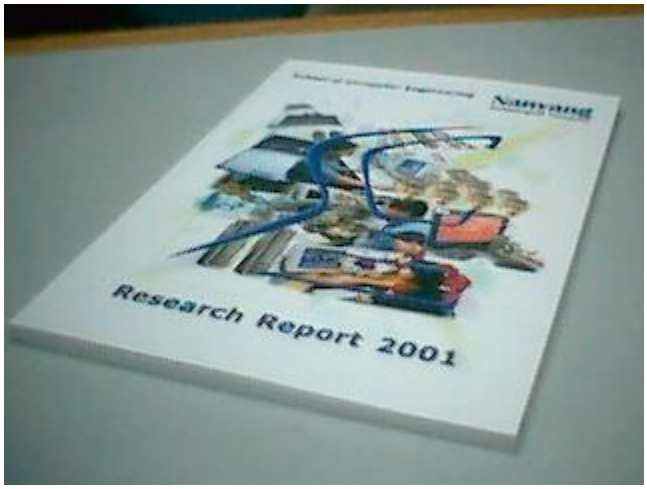


The left image is the modified power spectrum and the right image is the resultant image after inverse transform.

2.6 Undoing Perspective Distortion of Planar Surface

1. Read and display the image “book.jpg”, obtain the image coordinates of the 4 corners of the book.

   The coordinates of the 4 corners are (143, 28), (6, 159), (257, 214) and (308, 47).



2. Setup the matrices required to estimate the projective transformation.
   The given matrix equation is as follow:

$$\begin{bmatrix} X_w^1 & Y_w^1 & 1 & 0 & 0 & 0 & -x_{im}^1 X_w^1 & -x_{im}^1 Y_w^1 \\ 0 & 0 & 0 & X_w^1 & Y_w^1 & 1 & -y_{im}^1 X_w^1 & -y_{im}^1 Y_w^1 \\ X_w^2 & Y_w^2 & 1 & 0 & 0 & 0 & -x_{im}^2 X_w^2 & -x_{im}^2 Y_w^2 \\ 0 & 0 & 0 & X_w^2 & Y_w^2 & 1 & -y_{im}^2 X_w^2 & -y_{im}^2 Y_w^2 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ X_w^n & Y_w^n & 1 & 0 & 0 & 0 & -x_{im}^n X_w^n & -x_{im}^n Y_w^n \\ 0 & 0 & 0 & X_w^n & Y_w^n & 1 & -y_{im}^n X_w^n & -y_{im}^n Y_w^n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{31} \\ m_{32} \end{bmatrix} = \begin{bmatrix} x_{im}^1 \\ y_{im}^1 \\ x_{im}^2 \\ y_{im}^2 \\ \cdot \\ \cdot \\ x_{im}^n \\ y_{im}^n \end{bmatrix}$$

$$Au = V$$

where X_w^i and Y_w^i are the input coordinates and x_{im}^i and y_{im}^i are the target coordinates

```
x = [143, 6, 257, 308];
y = [28, 159, 214, 47];
xn=[0, 0, 210, 210];
yn=[0, 297, 297, 0];
```

Setup the matrix A:

```
A = [
    x(1), y(1), 1, 0, 0, 0, -xn(1) * x(1), -xn(1) * y(1);
    0, 0, 0, x(1), y(1), 1, -yn(1) * x(1), -yn(1) * y(1);
    x(2), y(2), 1, 0, 0, 0, -xn(2) * x(2), -xn(2) * y(2);
    0, 0, 0, x(2), y(2), 1, -yn(2) * x(2), -yn(2) * y(2);
    x(3), y(3), 1, 0, 0, 0, -xn(3) * x(3), -xn(3) * y(3);
    0, 0, 0, x(3), y(3), 1, -yn(3) * x(3), -yn(3) * y(3);
    x(4), y(4), 1, 0, 0, 0, -xn(4) * x(4), -xn(4) * y(4);
    0, 0, 0, x(4), y(4), 1, -yn(4) * x(4), -yn(4) * y(4);
];
```

Setup the matrix V, which consists of the target coordinates after transformation.

```
V = [0, 0, 0, 297, 210, 297, 210, 0]';
```

3. Compute the matrix $u = A^{-1}V$ and convert the projective transformation parameters m_{ij} to the normal matrix $U = \{m_{ij}\}$

```
u = A \ V;
U = reshape([u; 1], 3, 3)';
```

4. Verify if the calculated projective transformation parameters are correct by projecting the original coordinates to the targeted coordinates, then check if the targeted coordinates are as expected.

$$\begin{bmatrix} kx_{im} \\ ky_{im} \\ k \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

```
w = U * [x; y; ones(1, 4)];
w = w ./ (ones(3, 1) * w(3, :));
assert(isequal(round(w(1, :)), xn));
assert(isequal(round(w(2, :)), yn));
```

5. Warp the image and display the result. Comment on the quality of transformation and suggest reasons.

```
T= maketform("projective", U');
new_img = imtransform(img, T, 'XData', [0, 210], 'YData', [0, 297]);
figure("Name", "book-projected");
imshow(new_img);
```

The transformation managed to transform the book from the original slanted view to a frontal view. Some parts of the image became blurred which is due to the stretching of image. Stretching caused a pixel in the old image to be mapped to several pixels in the new image, this caused a lack of resolution as the same pixel is being repeated, thus causing the stretched part to be blurred.