NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

COLLEGE OF COMPUTING AND DATA SCIENCE

SC4061 – Computer Vision

Laboratory 1

Ahmet Buğra Kuş

N2503674F

College of Computing and Data Science

October 7, 2025

Contents

1 Introduction

In this laboratory session, I implemented several foundational techniques in image processing using MATLAB. The experiments were structured according to the Lab 1 Manual, starting from basic point processing operations and moving toward spatial and frequency domain methods.

The session began with contrast stretching, where I observed how linear scaling of pixel intensities could significantly enhance image visibility. This was followed by histogram equalization, which improved contrast by redistributing intensity values across the image.

Next, I applied spatial filtering techniques such as averaging and Gaussian filters for smoothing, and Laplacian filtering for sharpening. I also explored median filtering to reduce salt-and-pepper noise without blurring the edges.

Another key part of the lab was working in the Fourier domain to suppress periodic noise. I used notch filters to selectively remove interference patterns based on their frequency characteristics.

In the later sections, I performed a perspective transformation using homography to correct distortion in a planar surface, simulating document rectification. Finally, I implemented two versions of the perceptron learning algorithm — one based on rule-based updates, and the other using a unified update formulation — to classify linearly separable data.

This lab provided me with practical exposure to core concepts in digital image processing and introduced me to basic machine learning principles in classification.

2 Experiments

The following sections detail the experiments conducted during the lab session. Each experiment focuses on a specific image processing concept, implemented using MATLAB. The results and observations from each part are presented alongside the relevant code and explanations.

2.1 Contrast Stretching

In this experiment, we explored the concept of contrast stretching, a basic point processing method used to enhance the visual quality of grayscale images. By linearly scaling the intensity values from their original range to the full 0–255 range, we aimed to improve the visibility of image details in a low-contrast input.

(a) Original grayscale image

(b) After contrast stretching

Figure 1: Contrast Stretching applied to `mrt-train.jpg`

2.1.1 Appendix

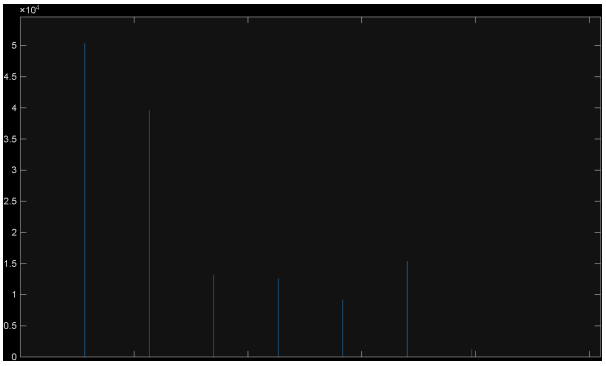
```
Pc = imread('mrt-train.jpg');
whos Pc
P = rgb2gray(Pc);
figure; imshow(P);

min_P = min(P(:));     % min_P = 13
max_P = max(P(:));     % max_P = 204

P2 = imsubtract(double(P), double(min_P));
P2 = uint8(immultiply(P2, 255 / double((max_P - min_P))));
% as learnt in lecture

figure; imshow(P2);

max_P2 = max(P2(:));   % max_P2 = 255
min_P2 = min(P2(:));   % min_P2 = 0
```

Listing 1: Contrast Stretching in MATLAB

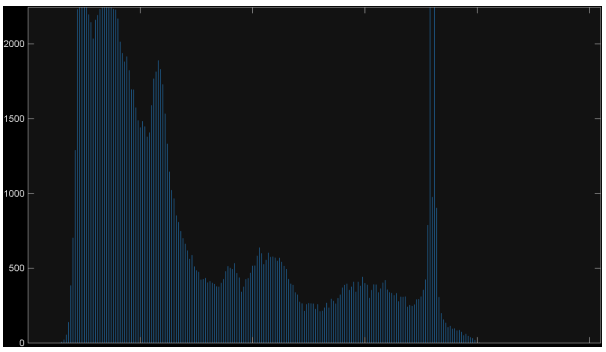
2.2 Histogram Equalization

2.2.1 (a) Histogram of Original Image

In this part, the histogram of the grayscale image `P` is visualized using two different bin sizes: 10 and 256.



(a) Histogram with 10 bins



(b) Histogram with 256 bins

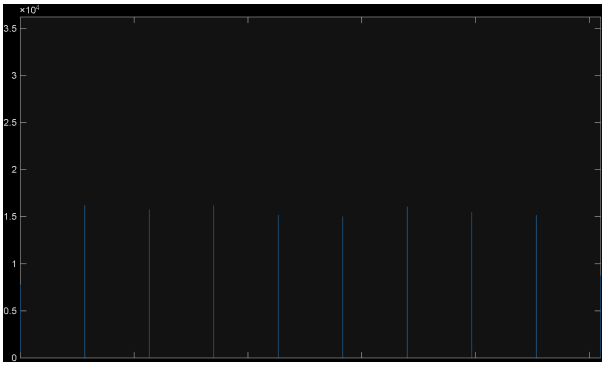
Figure 2: Initial histograms of grayscale image

When using only 10 bins, the histogram provides a coarse overview of the image's intensity distribution, grouping many pixel values into broad ranges. In contrast, the 256-bin histogram offers a much more detailed representation, clearly showing the exact frequency of each gray level. This finer resolution helps better understand the image's contrast characteristics and guides enhancement methods more effectively.

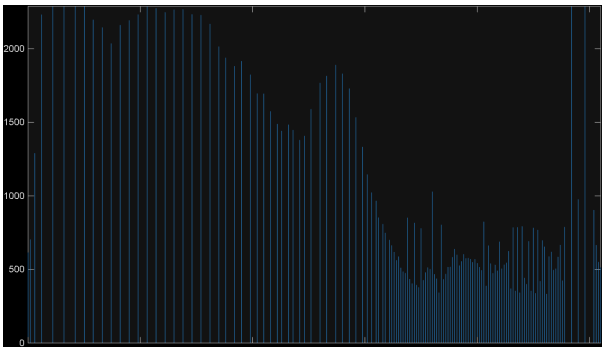
2.2.2 (b) After Histogram Equalization

Histogram equalization was applied using the `histeq` command. The resulting image and its histograms were analyzed.

After applying histogram equalization, the 10-bin histogram appears almost flat, showing that the pixel intensities are more evenly distributed across broad ranges. The 256-bin histogram, however, still reveals some local variations but with a much more uniform spread than before equalization. This indicates that the overall contrast has been enhanced, but at higher resolution the intensity levels are not perfectly uniform due to the image content.



(a) Equalized Histogram (10 bins)

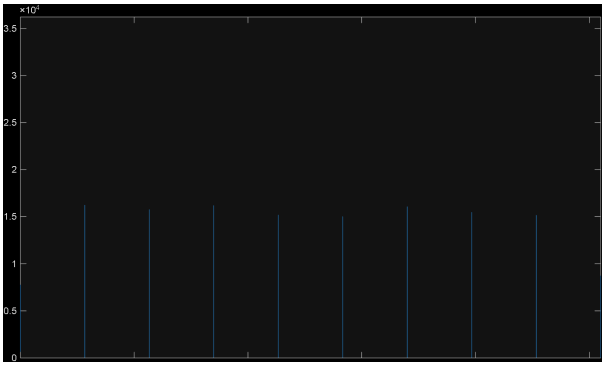


(b) Equalized Histogram (256 bins)

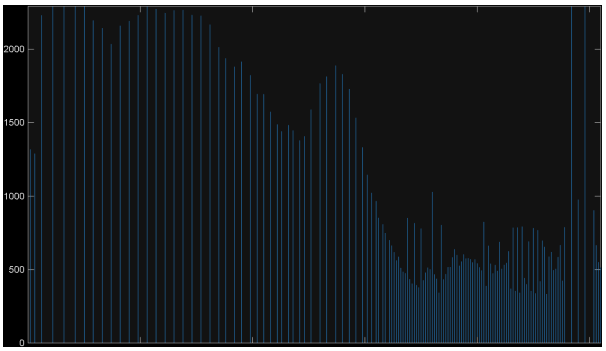
Figure 3: Histograms after histogram equalization

2.2.3 (c) Applying Equalization Again

After reapplying histogram equalization on the already equalized image *P3*, the histogram does **not** become significantly more uniform. This is because the first equalization step has already redistributed the intensity values to approximate a uniform histogram. Since histogram equalization is a nonlinear transformation intended to flatten the histogram, repeating the process on an already equalized image results in minimal changes. The mapping has already saturated the intensity range, so the second application has limited effect.



(a) Equalized Histogram (10 bins)



(b) Equalized Histogram (256 bins)

Figure 4: Histograms after histogram re-equalization

(a) Original mrt-train image

(b) Histogram equalization

(c) Histogram re-equalization

Figure 5: Histograms after histogram re-equalization

2.2.4 Appendix

```
figure; imhist(P,10);
figure;imhist(P,256);

P3 = histeq(P,255);
figure;imhist(P3,10);
figure;imhist(P3,256);
%imshow(P3);

P4 = histeq(P3,255);
figure;imhist(P4,10);
figure;imhist(P4,256);
% Display the original and processed images for comparison
figure;
subplot(1, 3, 1); imshow(P);
subplot(1, 3, 2); imshow(P3);
subplot(1, 3, 3); imshow(P4);
```

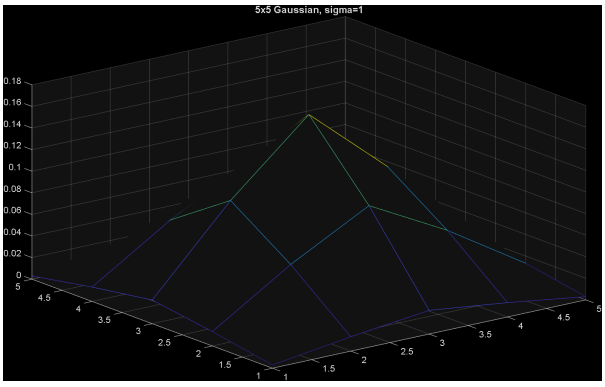
Listing 2: Histogram Equalization in MATLAB

2.3 Linear Spatial Filtering

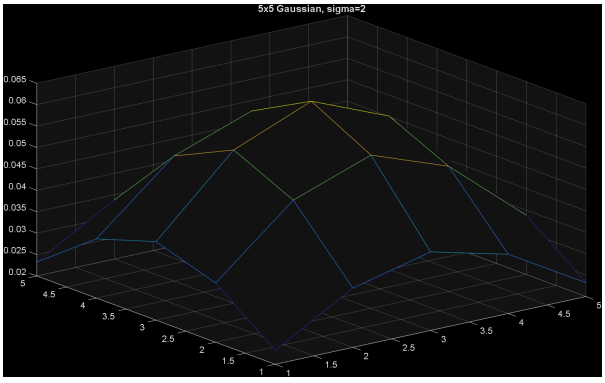
In this experiment, we explored linear spatial filtering by applying Gaussian filters to images with different types of noise. The filtering operation is based on the convolution of the image with a kernel function, commonly referred to as the point spread function (PSF).

2.3.1 (a) Generating Gaussian Filters

Two 2D Gaussian filters were generated with size 5×5, using standard deviations $\sigma = 1.0$ and $\sigma = 2.0$, respectively. Each filter was normalized such that the sum of its elements is 1. These filters were visualized using the `mesh()` function in MATLAB.



(a) Gaussian filter, $\sigma = 1.0$



(b) Gaussian filter, $\sigma = 2.0$

Figure 6: Generated Gaussian Filters visualized using `mesh()`

2.3.2 (b) Viewing Image with Gaussian Noise

The image `lib-gn.jpg`, which contains additive Gaussian noise, was loaded and visualized.

Figure 7: Original image with Gaussian noise

2.3.3 (c) Filtering the Noisy Image

The image was convolved with both filters using `conv2()`. The results show how filtering reduces noise but may also blur the image.

(a) Filtered with $\sigma = 1.0$

(b) Filtered with $\sigma = 2.0$

Figure 8: Filtered images with Gaussian filters

Filtering with a smaller σ preserves edges better but removes less noise. A larger σ smooths the image more aggressively, which can reduce noise more effectively but also causes more blurring.

2.3.4 (d) Viewing Image with Speckle Noise

The image `lib-sp.jpg`, which contains speckle noise, was viewed next.

Figure 9: Original image with Speckle noise

2.3.5 (e) Filtering Speckle Noise

The same filters were applied to the speckle-noised image. The results were compared against those for Gaussian noise.

(a) Filtered with $\sigma = 1.0$

(b) Filtered with $\sigma = 2.0$

Figure 10: Filtered images with speckle noise

Gaussian filters are generally more effective on Gaussian noise compared to speckle noise. While some reduction is observed in speckle noise, specialized filters such as median filtering tend to perform better for this type of noise.

2.3.6 Appendix

```
size = 5;
[x,y] = meshgrid(-(size-1)/2:(size-1)/2, -(size-1)/2:(size-1)/2);

sigma1 = 1.0;
h1 = (1/(2*pi/(sigma1.^2)))*exp(-(x.^2 + y.^2) / (2*sigma1.^2));
h1 = h1 / sum(h1(:));   % normalize

sigma2 = 2.0;
h2 = (1/(2*pi*(sigma2.^2)))*exp(-(x.^2 + y.^2) / (2*sigma2.^2));
h2 = h2 / sum(h2(:));

figure; mesh(h1); title('5x5 Gaussian, sigma=1');
figure; mesh(h2); title('5x5 Gaussian, sigma=2');


Ig = imread('lib-gn.jpg');

Ig_h1 = uint8(conv2(double(Ig), h1, 'same'));
Ig_h2 = uint8(conv2(double(Ig), h2, 'same'));

%figure; montage({Ig, Ig_h1, Ig_h2}, 'Size',[1 3]);
%title('Gaussian noise: Original | a=1 | a=2');
figure; imshow(Ig_h1);
figure; imshow(Ig_h2);

Ig_sp = imread('lib-sp.jpg');

Igsp_h1 = uint8(conv2(double(Ig_sp), h1, 'same'));
Igsp_h2 = uint8(conv2(double(Ig_sp), h2, 'same'));

%figure; montage({Ig_sp, Igsp_h1, Igsp_h2}, 'Size',[1 3]);
%title('speckle noise: Original | a=1 | a=2');

figure; imshow(Igsp_h1);
figure; imshow(Igsp_h2);
```

Listing 3: Linear Spatial Filtering in MATLAB

2.4 Median Filtering

This experiment explores median filtering, a non-linear technique effective in removing salt-and-pepper and speckle noise while preserving edges better than linear filters.

2.4.1 Image with Gaussian Noise (lib-gn.jpg)

Median filtering was applied to the Gaussian noise-corrupted image using two different neighborhood sizes: 3×3 and 5×5. The performance was compared with Gaussian filtering.

(a) Original (Gaussian noise) (b) Median Filter 3x3 (c) Median Filter 5x5

Figure 11: Median filtering on Gaussian noise

Gaussian noise is better handled by Gaussian filtering, as median filtering may leave some residual noise. However, increasing the neighborhood size in median filtering does smooth the noise more at the cost of edge clarity.

2.4.2 Image with Speckle Noise (lib-sp.jpg)

The same median filters were applied to the speckle noise-corrupted image. The results are shown below.

(a) Original (Speckle noise) (b) Median Filter 3x3 (c) Median Filter 5x5

Figure 12: Median filtering on Speckle noise

Median filtering shows more effective suppression of speckle noise compared to Gaussian filtering. The 5x5 filter yields better noise reduction but introduces slight blurring.

2.4.3 Comparison with Gaussian Filtering

Median filtering preserves edges better than Gaussian filtering, especially in cases of salt-and-pepper or speckle noise. However, it may be less effective than Gaussian filtering in smoothing out Gaussian-distributed noise. Gaussian filtering is linear and smooths uniformly, while median filtering selectively removes outliers, making it more robust for non-Gaussian noise patterns.

2.4.4 Appendix

```
1
2  Ig_mf = imread('lib-gn.jpg');
3
4  Ig_mf_3=uint8(medfilt2(Ig_mf,[3 3]));
5  Ig_mf_5=uint8(medfilt2(Ig_mf,[5 5]));
6
7  %figure; montage({Ig_mf, Ig_mf_3, Ig_mf_5}, 'Size',[1 3]);
8  %title('Gaussian noise: Original | 3x3 | 5x5');
9
10 figure;imshow(Ig_mf_3);
11 figure; imshow(Ig_mf_5);
12 Ig_sp_mf = imread('lib-sp.jpg');
13 Ig_sp_mf_3=uint8(medfilt2(Ig_sp_mf,[3 3]));
14 Ig_sp_mf_5=uint8(medfilt2(Ig_sp_mf,[5 5]));
15
16 %figure; montage({Ig_sp_mf, Ig_sp_mf_3, Ig_sp_mf_5}, 'Size',[1 3]);
17 %title('spectle noise: Original | 3x3 | 5x5');
18
19 figure;imshow(Ig_sp_mf_3);
20 figure; imshow(Ig_sp_mf_5);
```

Listing 4: Median Filtering in MATLAB

2.5 Suppressing Noise Interference Patterns

This experiment explores the suppression of periodic interference patterns in images using frequency domain filtering via the Fourier transform. The method is particularly useful in removing structured noise, such as diagonal or fence-like patterns that manifest as strong spectral components in the frequency domain.

(a) Image and Power Spectrum Visualization

The corrupted image `pck-int.jpg` was loaded and transformed into the frequency domain using the `fft2` command. The power spectrum was computed and visualized, with and without `fftshift`, to detect prominent frequency peaks associated with the interference.

```
or_im = imread("pck-int.jpg");
orim_ft = fft2(double(or_im));     % complex-valued spectrum
orim_ft2 = abs(orim_ft).^2;        % power spectrum
imagesc(orim_ft2.^0.1); colormap default;
title('Power spectrum WITHOUT fftshift (S^{0.1})');
```

Listing 5: Power spectrum visualization

Figure 13: Original corrupted image `pck-int.jpg`

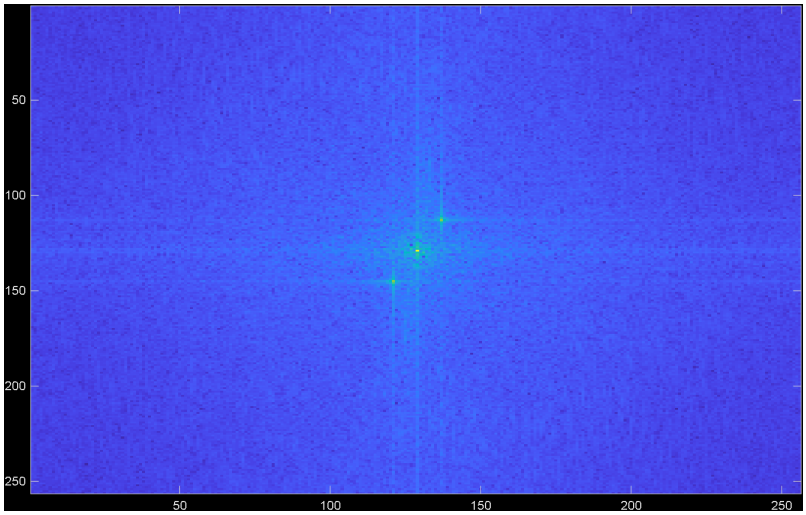


Figure 14: Unshifted power spectrum $S^{0.1}$ (no fftshift)

(b) Locating Peaks in Frequency Domain

Two distinct symmetric peaks were identified in the unshifted power spectrum using `ginput`. These peaks correspond to interference components in the image:

- Peak #1: (row = 9, col = 241)
- Peak #2: (row = 249, col = 17)

(c) Notch Filtering via Zeroing

To suppress the unwanted frequencies, small 5x5 neighborhoods centered on the detected peaks in the Fourier domain were set to zero. This removes the narrow-band components causing the interference.

```
pe_po1 = [241, 9];
pe_po2 = [17, 249];
rad = 2;
[H, W] = size(orim_ft);

r1_a = max(1, pe_po1(1)-rad); r1_b = min(H, pe_po1(1)+rad);
c1_a = max(1, pe_po1(2)-rad); c1_b = min(W, pe_po1(2)+rad);

r2_a = max(1, pe_po2(1)-rad); r2_b = min(H, pe_po2(1)+rad);
c2_a = max(1, pe_po2(2)-rad); c2_b = min(W, pe_po2(2)+rad);

orim_ft(r1_a:r1_b, c1_a:c1_b) = 0;
orim_ft(r2_a:r2_b, c2_a:c2_b) = 0;
```

Listing 6: Zeroing 5x5 neighborhoods around peaks

(d) Spectrum and Image Reconstruction

After zeroing the peaks, the modified spectrum was visualized using `fftshift`. The image was then reconstructed using the inverse Fourier transform `ifft2`.

```
S2 = abs(orim_ft).^2;
imagesc(fftshift(S2.^0.1)); colormap default;

Irec = real(ifft2(orim_ft));
Irec2 = uint8(Irec);
montage({or_im, Irec2}, 'Size', [1 2]);
title('Original | Notch-filtered');
```

Listing 7: Reconstruction after notch filtering

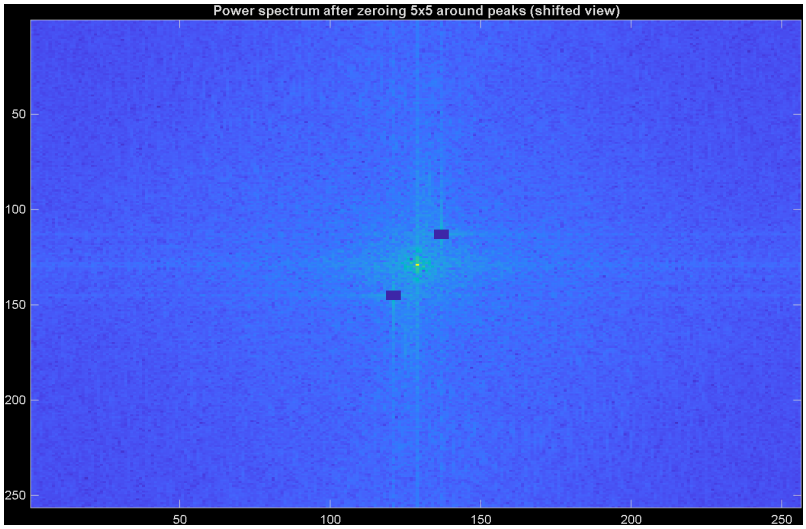


Figure 15: Shifted spectrum after notch filtering (highlighting removed peaks)

Figure 16: Image comparison: Left — original, Right — filtered

(e) Results and Observations

The notch filtering effectively suppressed the dominant diagonal interference pattern in the image. While the reconstructed image retains overall structure and detail, some minor distortions may be introduced due to the loss of nearby frequency information. Nevertheless, the removal of structured noise significantly enhances the image quality.

(f) Fence Removal in `primate-caged.jpg`

The image `primate-caged.jpg` contains a visual obstruction in the form of a fence, forming periodic vertical patterns. In this part, we attempt to remove the fence interference using frequency domain notch filtering, following the same steps used previously.

Step 1: Load and Visualize the Power Spectrum The image was converted to grayscale and transformed into the frequency domain using `fft2`. The power spectrum was computed and visualized without `fftshift`, revealing bright peaks corresponding to the periodic fence structure.

```
prim = imread("primate-caged.jpg");
prim = rgb2gray(prim);
A = fft2(double(prim));           % Frequency domain (complex-valued)
B = abs(A).^2;                    % Power spectrum

imagesc(B.^0.1); colormap default;
title('Power spectrum WITHOUT fftshift (S^{0.1})');
```

Listing 8: Power spectrum computation for primate image

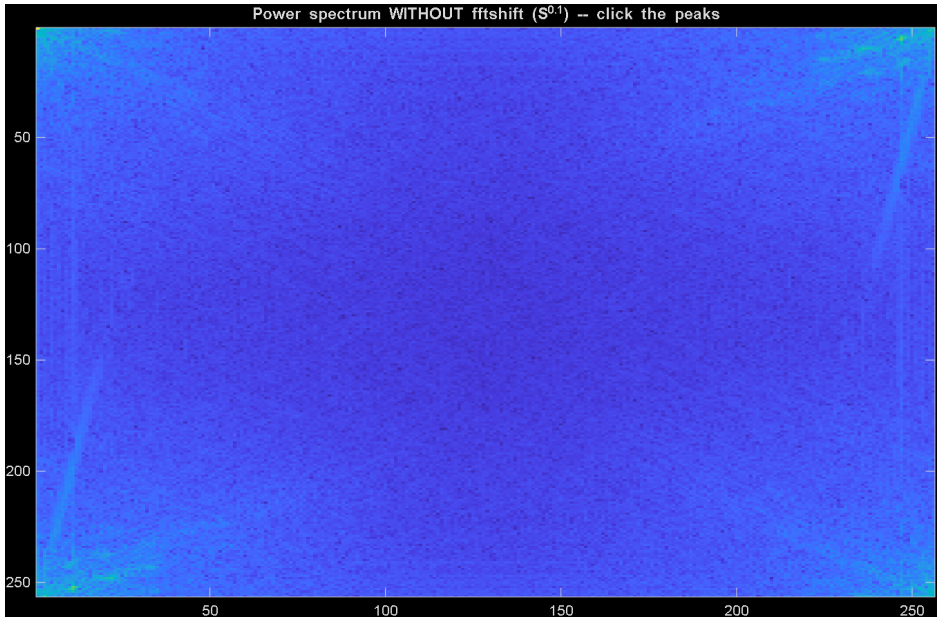


Figure 17: Power spectrum of `primate-caged.jpg` without `fftshift`. Peaks caused by fence pattern are visible.

Step 2: Identify and Suppress the Peaks Two prominent peaks were manually identified from the spectrum:

- Peak #1: (row = 252, col = 11)
- Peak #2: (row = 10, col = 236)

Small 5×5 neighborhoods around these peaks were zeroed in the frequency domain to suppress the interference.

```
peak1 = [252, 11];
peak2 = [10, 236];
rad = 2;
[H, W] = size(A);

% Safe index boundaries
r1_a = max(1, peak1(1)-rad); r1_b = min(H, peak1(1)+rad);
c1_a = max(1, peak1(2)-rad); c1_b = min(W, peak1(2)+rad);

r2_a = max(1, peak2(1)-rad); r2_b = min(H, peak2(1)+rad);
c2_a = max(1, peak2(2)-rad); c2_b = min(W, peak2(2)+rad);

% Apply notch filtering
A(r1_a:r1_b, c1_a:c1_b) = 0;
A(r2_a:r2_b, c2_a:c2_b) = 0;
```

Listing 9: Zeroing around frequency peaks

Step 3: Visualize Filtered Spectrum and Reconstruct the Image The power spectrum after filtering was recalculated and displayed. The image was then reconstructed using the inverse Fourier transform `ifft2`.

```
B2 = abs(A).^2;
imagesc(fftshift(B2.^0.1)); colormap default;
title('Power spectrum after notch filtering');

% Reconstruct image
Irec = real(ifft2(A));
Irec2 = uint8(Irec);
montage({prim, Irec2}, 'Size', [1 2]);
title('Fence Removal Attempt: Original | Filtered');
```

Listing 10: Reconstruction after filtering

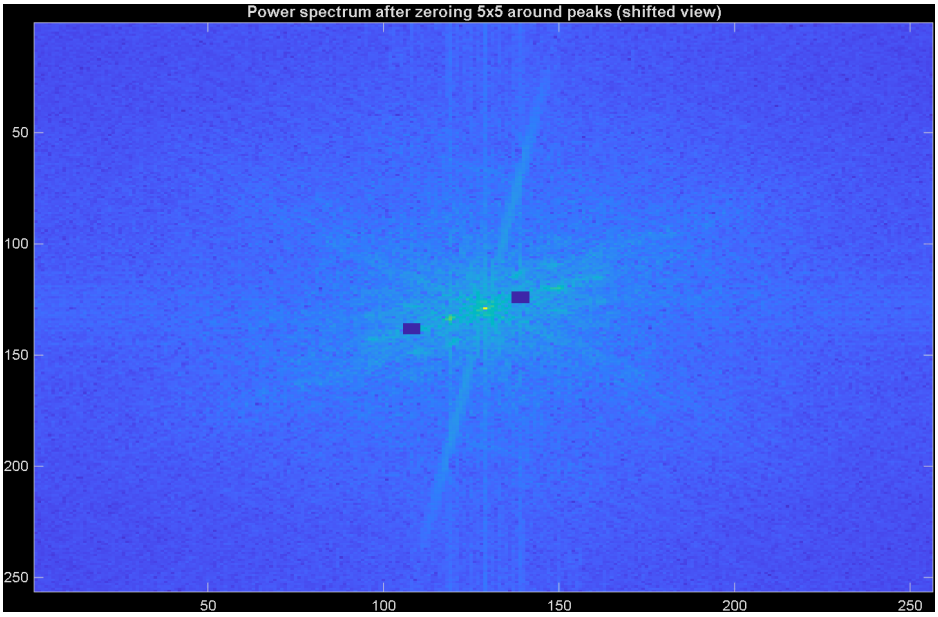


Figure 18: Power spectrum after notch filtering on `primate-caged.jpg`. Symmetric peaks removed.

Figure 19: Fence removal result: Original image (left) and filtered image (right) using notch filtering.

Observations The notch filtering effectively suppressed the dominant frequency components caused by the fence. While not perfectly clean, the visual obstruction is significantly reduced, making the object behind the fence more visible. Some minor distortions remain due to the proximity of removed frequencies to actual image content. However, this method demonstrates the usefulness of frequency domain filtering in handling periodic noise patterns.

2.6 Undoing Perspective Distortion of Planar Surface

This section focuses on removing perspective distortion from an image of a book and detecting a prominent pink-colored region, potentially corresponding to a screen or highlighted area. The task involves two main steps: applying a projective transformation to obtain a frontal view of the book, and analyzing the resulting image to isolate the pink region using color-based segmentation techniques.

Perspective Rectification We began by selecting four corner points on the slanted image using MATLAB's `ginput` function. These correspond to the corners of the book in the input image.

```
I = imread('book.jpg');
imshow(I);
title('Click 4 corners');
[x, y] = ginput(4);
```

Listing 11: User selection of 4 corners from the slanted image

Knowing the actual physical dimensions of an A4 sheet (210 mm × 297 mm), we mapped these points to their corresponding locations in the desired fronto-parallel view:

```
X = [0 210 210 0];
Y = [0 0 297 297];
```

Listing 12: Setting the real-world coordinates for A4 paper

To compute the projective transformation, we formulated a linear system $Au = v$ using the relationship between input and output coordinates. Solving this system yielded the transformation parameters, which were reshaped into a 3×3 homography matrix:

```
A = zeros(8, 8);
v = zeros(8, 1);
for i = 1:4
    A(2*i-1, :) = [x(i) y(i) 1 0 0 0 -X(i)*x(i) -X(i)*y(i)];
    v(2*i-1)    = X(i);
    A(2*i, :)   = [0 0 0 x(i) y(i) 1 -Y(i)*x(i) -Y(i)*y(i)];
    v(2*i)      = Y(i);
end
u = A \ v;
U = reshape([u;1], 3, 3)';
```

Listing 13: Computing the projective transformation matrix

We verified the transformation by applying it to the clicked points:

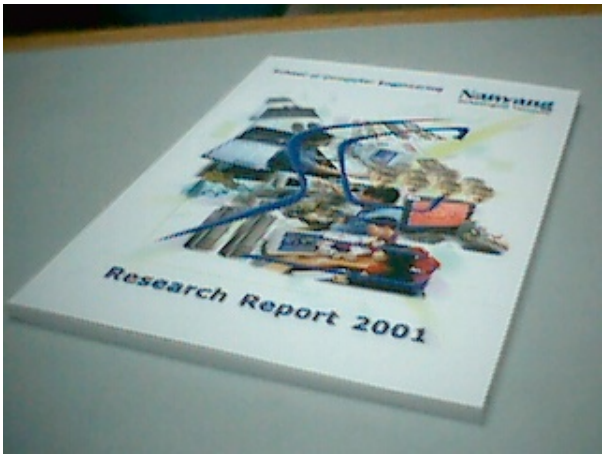
```
w = U * [x'; y'; ones(1,4)];
w = w ./ (ones(3,1) * w(3,:));
disp(w);
```

Listing 14: Verifying coordinate transformation

Then, the image was warped using the computed transformation:

```
T = maketform('projective', U');
I2 = imtransform(I, T, 'XData', [0 210], 'YData', [0 297]);
figure; imshow(I2);
title('Corrected Image');
```

Listing 15: Applying the perspective transformation to the image



(a) Original slanted book image

(b) Perspective-corrected (warped) image

Figure 20: Perspective correction using projective transform based on four manually selected corners.

Detection of Pink Region Following the rectification, the goal was to detect the large pink area in the center-right of the book image. This region may represent a digital screen or marked area and is characterized by its distinct hue.

To make the detection more robust to lighting, we first converted the image from RGB to HSV space:

```
img = imread('corrected_book.jpg');
hsv_img = rgb2hsv(img);
h = hsv_img(:,:,1);  % Hue
s = hsv_img(:,:,2);  % Saturation
v = hsv_img(:,:,3);  % Brightness
```

Listing 16: Conversion from RGB to HSV color space

Using the hue channel, we applied thresholds to isolate pink tones:

```
mask1 = h < 0.06;
mask2 = h > 0.85;
mask = (mask1 | mask2);
mask = mask & s > 0.32 & v > 0.32;
```

Listing 17: Creating binary mask for pink detection

To eliminate false positives around the edges, we removed the border pixels:

```
mask(1:20, :) = 0;
mask(end-20:end, :) = 0;
mask(:, 1:20) = 0;
mask(:, end-20:end) = 0;
```

Listing 18: Suppressing border noise

The mask was refined by retaining only the largest connected region:

```
[label_img, num_labels] = bwlabel(mask);
max_area = 0;
for i = 1:num_labels
    current_area = sum(label_img(:) == i);
    if current_area > max_area
        max_area = current_area;
        biggest_label = i;
    end
end
clean_mask = (label_img == biggest_label);
```

Listing 19: Identifying and isolating largest pink region

Finally, the bounding box of the detected region was visualized:

```
[rows, cols] = find(clean_mask);
xmin = min(cols); xmax = max(cols);
ymin = min(rows); ymax = max(rows);

figure;
imshow(img);
hold on;
rectangle('Position', [xmin ymin xmax - xmin ymax - ymin], ...
          'EdgeColor', 'g', 'LineWidth', 2);
title('Detected Pink Area');
hold off;
```

Listing 20: Bounding box visualization

(a) Binary mask showing detected pink pixels

(b) Detected pink region highlighted with bounding box

Figure 21: Detection and localization of prominent pink region using HSV-based segmentation and blob analysis.

Observations The projective transformation successfully removed the perspective distortion, yielding a frontal, properly scaled view of the book. This rectification was crucial, as it allowed the subsequent color segmentation step to operate on a geometrically normalized image.

The HSV-based pink detection algorithm proved effective in highlighting the target region. By leveraging the properties of hue, saturation, and brightness, the algorithm isolated the pink region with reasonable precision. Noise reduction through boundary clearing and connected component analysis further enhanced the robustness of the detection.

2.7 Code Two Perceptrons

In this section, we implemented two perceptron learning algorithms — the classic Perceptron Learning Algorithm (PLA) and a gradient-based update rule. Both algorithms were applied to a simple binary classification task to observe their behavior.

Problem Setup We used the following samples:

- $\mathbf{x}_1 = [3, 3, 1]$, label $r_1 = +1$
- $\mathbf{x}_2 = [1, 1, 1]$, label $r_2 = -1$

The learning rate was set to $\alpha = 1$. Both algorithms were run for 4 alternating steps.

Algorithm 1: Perceptron Learning Algorithm (PLA)

This algorithm updates the weights only when a sample is misclassified:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbf{x}_k \quad \text{if } r_k = +1 \text{ and misclassified}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{x}_k \quad \text{if } r_k = -1 \text{ and misclassified}$$

Starting from $\mathbf{w} = [0, 0, 0]$, the updates after each iteration were:

- Iteration 1: $\mathbf{w} = [3, 3, 1]$
- Iteration 2: $\mathbf{w} = [2, 2, 0]$
- Iteration 3: $\mathbf{w} = [2, 2, 0]$ (no change)
- Iteration 4: $\mathbf{w} = [1, 1, -1]$

Final weights: $\mathbf{w}_1 = [1, 1, -1]$.

Algorithm 2: Gradient-based Update

This algorithm updates the weights at every iteration based on the prediction error:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(r_k - \mathbf{w}^\top \mathbf{x}_k\right) \mathbf{x}_k$$

Starting from $\mathbf{w} = [0, 0, 0]$, the updates were:

- Iteration 1: $\mathbf{w} = [3, 3, 1]$
- Iteration 2: $\mathbf{w} = [-5, -5, -7]$
- Iteration 3: $\mathbf{w} = [109, 109, 31]$
- Iteration 4: $\mathbf{w} = [-141, -141, -219]$

Final weights: $\mathbf{w}_2 = [-141, -141, -219]$.

MATLAB Implementation The following MATLAB code implements both algorithms described above:

```
clc; clear;
% Learning rate
alpha = 1;
% Sample 1  belongs to Class +1
x1 = [3 3 1];    % Bias term added
r1 = 1;
% Sample 2  belongs to Class -1
x2 = [1 1 1];    % Bias term added
r2 = -1;

%% Algorithm 1: Perceptron Learning Rule
w1 = [0 0 0];    % Initial weights for Algorithm 1

for step = 1:4
    if mod(step, 2) == 1
        xk = x1; rk = r1;
    else
        xk = x2; rk = r2;
    end
    yk = dot(w1, xk);    % Prediction
    % Apply PLA rules
    if (rk == 1 && yk <= 0)
        w1 = w1 + alpha * xk;
    elseif (rk == -1 && yk >= 0)
        w1 = w1 - alpha * xk;
    end
end

%% Algorithm 2: Gradient-based Update (Slide 5.3)
w2 = [0 0 0];    % Initial weights for Algorithm 2

for step = 1:4
    if mod(step, 2) == 1
        xk = x1; rk = r1;
    else
        xk = x2; rk = r2;
    end
    yk = dot(w2, xk);    % Prediction
    % Update using error signal (r - y) * x
    w2 = w2 + alpha * (rk - yk) * xk;
end

%% Display Results
disp('Final weights using Algorithm 1 (PLA):');
disp(w1);

disp('Final weights using Algorithm 2 (Gradient-based from Slide):');
disp(w2);
```

Listing 21: Implementation of Algorithm 1 (PLA) and Algorithm 2 (Gradient-based update)

Comparison and Observations Algorithm 1 showed steady, controlled weight changes, converging to a reasonable separating hyperplane. Algorithm 2, however, produced very large weight values due to the unconstrained gradient-like updates. This highlights the difference between a fixed update rule (PLA) and an error-scaled update rule (Gradient-based), and shows why normalization or a smaller learning rate is often required for gradient-based methods.