SC4061 – Computer Vision

Laboratory 1 Report

Muhammed Ikbal Özbey

N2504208D

College of Computing and Data Science

October 7, 2025

Contents

1 Introduction

This laboratory aims to introduce fundamental image processing concepts using MATLAB. Through a series of experiments, techniques such as contrast stretching, histogram equalization, spatial and frequency filtering, and geometric transformations are explored. These exercises help develop a practical understanding of how digital images can be enhanced, filtered, and analyzed.

2 Environment Setup

2.1 Contrast Stretching

2.1.d - Calculation Codes for Contrast Stretching

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

min_value=min(P(:));
max_value=max(P(:));

new_image =  imsubtract(double(P),double(min_value));
ratio= (255/double(max_value-min_value));
new_image = uint8(immultiply(new_image, ratio));

enhance_pic_max=max(new_image(:));
enhance_pic_min=min(new_image(:));
```

2.1.e - New Image

Figure 1: Original Image

Figure 2: After Contrast Stretching

As shown in Figures 1 and 2, contrast stretching enhances the image clarity and reveals more details in shadowed areas.

2.2 Histogram Equalization

2.2.a - Compare histograms and image appearance

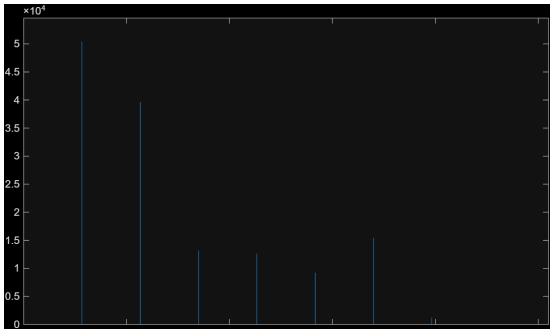

Figure 3: 10 Bins

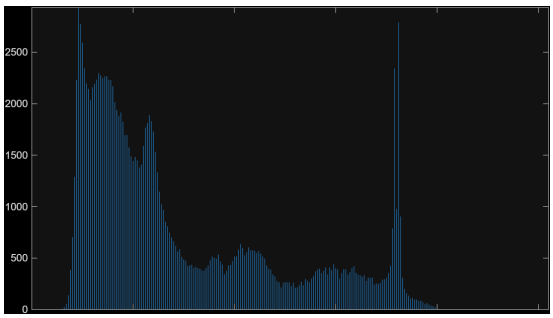

Figure 4: 256 Bins

The **256-bin histogram** provides a very detailed distribution of gray levels. However, it also shows sharp spikes at specific values (e.g., around x = 170–180), which indicate outliers caused by certain pixel intensities occurring very frequently. These extreme peaks can distort the interpretation of the overall contrast. Histogram smoothing or using fewer bins can help suppress these artifacts.

The **10-bin histogram** groups gray levels into wider ranges, so the distribution looks smoother. While some details are lost compared to the 256-bin version, the overall intensity distribution is easier to interpret and less sensitive to sharp outliers.

2.2.b - Answer

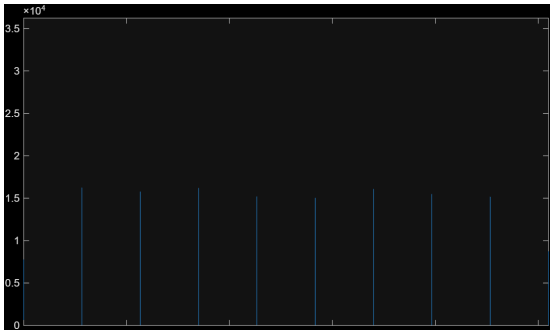

Figure 5: 10 Bins

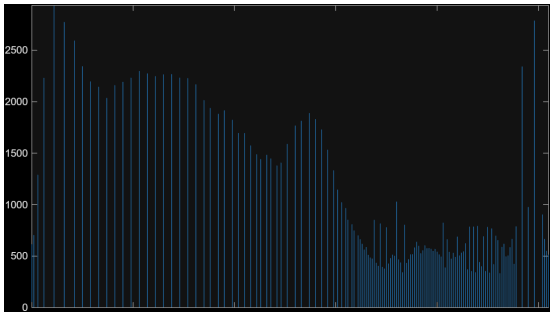

Figure 6: 256 Bins

The 10-bin histogram of P3 is smooth and nearly flat – the histogram is coarse, appearing almost flat and uniform because many gray levels are merged. The 256-bin histogram is more irregular and detailed, but both similarly equalized the histogram overall.

2.2.c - Answer The histogram equalized twice is almost the same as the one equalized once. Since the first equalization already redistributed intensities well, the second one did not significantly affect the image.

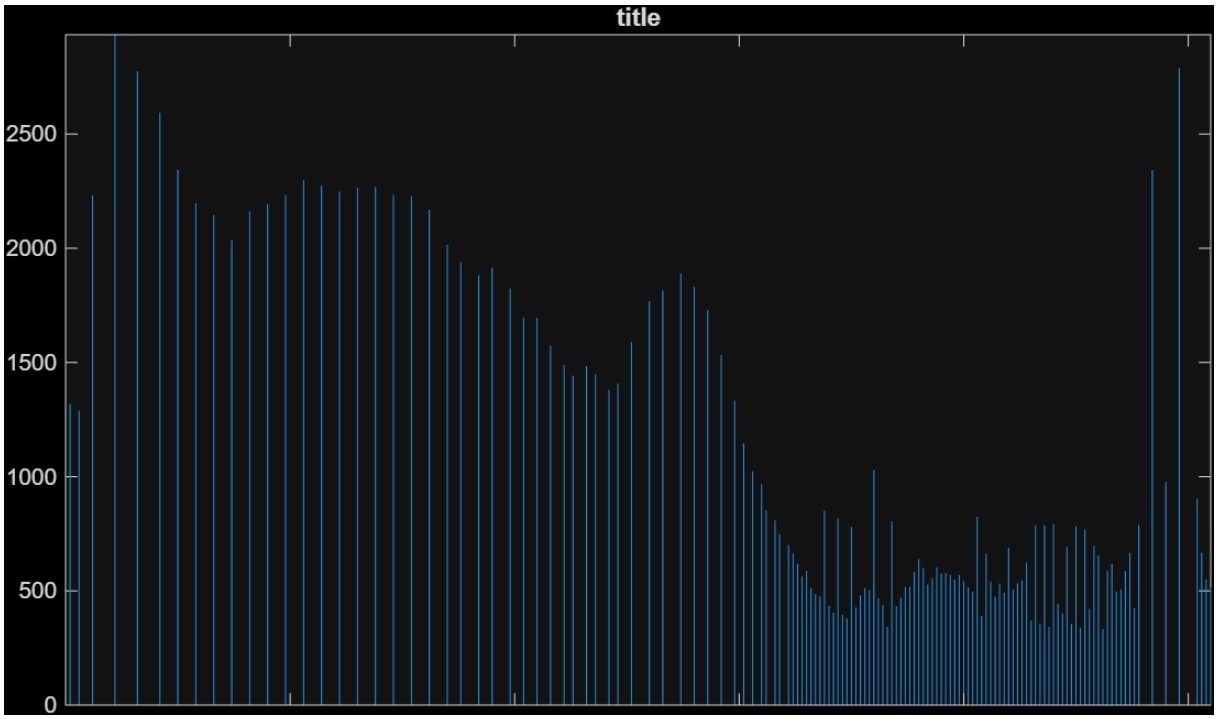

Figure 7: Two times equalized histograms

2.3 Linear Spatial Filtering

2.3.c - Answer

Figure 8: Original - Func 1 - Func 2

The convolution filter successfully removes most of the noise in the image, making it look cleaner overall. However, this noise removal comes at the cost of a slight blurring effect, which reduces the sharpness and fine details of the original image. As the sigma value increases, the blurring becomes more noticeable, and the image may start to lose important visual information. Therefore, it is generally a good idea to keep the sigma value low in order to get a good balance — reducing noise while still keeping the image sharp and detailed.

2.3.e - Answer

Figure 9: Original - Func 1 - Func 2

Even after applying a Gaussian filter to the image with speckle noise (lib-sp.jpg), some bright speckles remain visible and continue to affect the image quality. Using a second filter with a higher sigma value helps slightly, but the noise is still not fully removed. In comparison to the filtered version of lib-gn.jpg (Gaussian noise), the speckle noise is more resistant to this kind of filtering. Therefore, we can say that Gaussian filtering is not very effective for removing speckle noise — it fails to clean the image completely and also causes some loss in sharpness and detail.

2.4 Median Filtering - Comparison and Trade-offs

When dealing with Gaussian noise, Gaussian filtering is generally effective because the noise matches the filter's assumptions. Increasing the sigma value reduces the noise more, but also blurs the image and causes a loss of detail. While median filters better preserve edges, they don't perform as well for Gaussian noise and tend to leave some noise behind.

For speckle or impulse noise, median filters are more effective. A 3×3 window removes most white speckles while keeping edges intact, and a 5×5 filter reduces even more noise at the cost of some smoothness. Gaussian filters, on the other hand, mostly blur speckle noise rather than removing it.

Overall, the choice of filter depends on the type of noise. Gaussian filters are suited for Gaussian noise, while median filters are better for speckle noise. There's always a trade-off between noise reduction and preserving image sharpness.

Figure 10: Gaussian noise: Original image (left), median filter 3×3 (middle), median filter 5×5 (right).

Figure 11: Speckle noise: Original image (left), median filter 3×3 (middle), median filter 5×5 (right).

2.5 Suppressing Noise Interference Patterns

2.5.b - Display the power spectrum

```
Y=fft2(pck_int);
S=abs(Y).^2;

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

2.5.b - 2.5.c Image Comparison

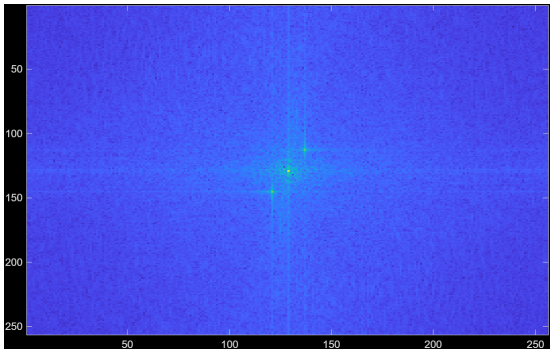

Figure 12: Without fftshitf

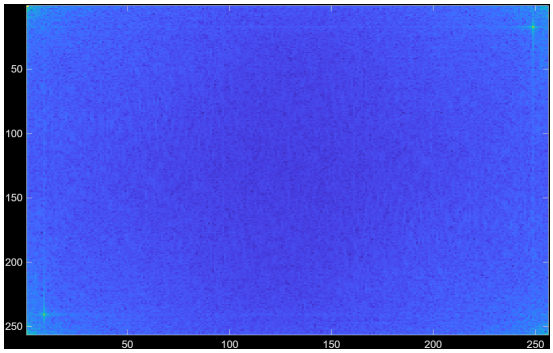

Figure 13: With fftshift

The peak points are (249,17), (9, 241). (The other points like (255,1), (1,1), (255,256) looks like peaks but when I applied functions without these point I get more proper result so I ignored them.)

2.5.d - Set to zero the 5x5 neighbourhood

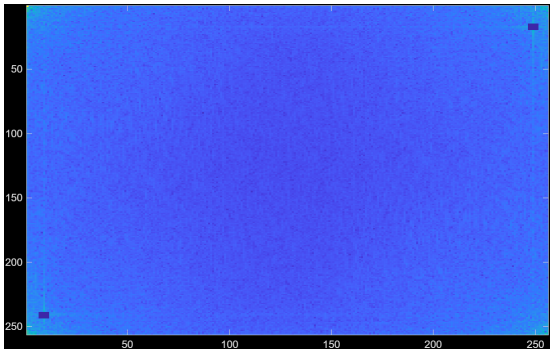

Figure 14: Without fftshitf

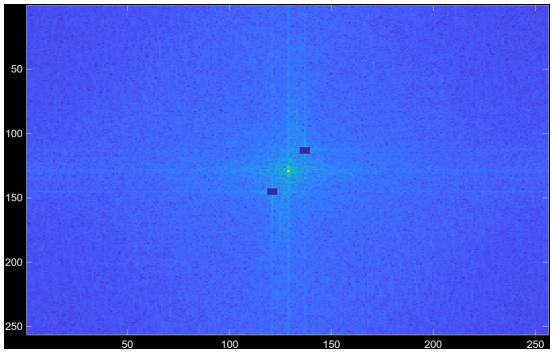

Figure 15: With fftshift

2.5.e - The result of operations

Figure 16: Original

Figure 17: Result

```
edited_img = real(ifft2(Y));
edited_img = uint8(  edited_img );
figure; imshow(edited_img);
```

The selected peak points are (249,17) and (9,241). Although points such as (255,1), (1,1), and (255,256) appeared to be peaks, including them resulted in less accurate outputs, so they were excluded. Suppressing these weaker peaks using a smaller neighborhood (e.g., 1×1) might help improve the result further. While the filtering significantly reduced the interference patterns, some traces still remain, and the removal is not fully complete.

2.5.f - Primate-caged

```
primate=imread('primate-caged.jpg');
primate=rgb2gray(primate);
Z=fft2(primate);
S_z=abs(Z).^2;

figure; imagesc((S_z.^0.1));       %[252 11] [6 247] peaks
colormap('default');
point3=[252 11]; point4=[6 247];

r_max_bndr3=point3(1)-2; r_min_bndr3=point3(1)+2;
c_max_bndr3=point3(2)-2; c_min_bndr3=point3(2)+2;

r_max_bndr4=point4(1)-2; r_min_bndr4=point4(1)+2;
c_max_bndr4=point4(2)-2; c_min_bndr4=point4(2)+2;

Z(r_max_bndr3:r_min_bndr3,c_max_bndr3:c_min_bndr3)=0;
Z(r_max_bndr4:r_min_bndr4,c_max_bndr4:c_min_bndr4)=0;
newS2=abs(Z).^2;

edited_img2 = real(ifft2(Z)); edited_img2 = uint8(edited_img2 );
```

Figure 18: Original

Figure 19: Result

The peak distribution was very scattered, which made it hard to fully remove the interference. Even after setting the main peaks like [252,11] and [6,247] to zero, the result was not completely clean. The image became slightly clearer, but the interference was still visible.

2.6 Undoing Perspective Distortion of Planar Surface

2.6.b - Taking input with "ginput" function

```
x=  [142.7391;310.3292;256.1357;5.2059]
y=  [28.9630;50.3672;215.6803;161.0313]
X = [0 210 210 0];
Y = [0 0 297 297];
```

2.6.c - Matrice set up

```
A = [
    x(1) y(1) 1 0 0 0 -X(1)*x(1) -X(1)*y(1);
    0 0 0 x(1) y(1) 1 -Y(1)*x(1) -Y(1)*y(1);
    x(2) y(2) 1 0 0 0 -X(2)*x(2) -X(2)*y(2);
    0 0 0 x(2) y(2) 1 -Y(2)*x(2) -Y(2)*y(2);

    x(3) y(3) 1 0 0 0 -X(3)*x(3) -X(3)*y(3);
    0 0 0 x(3) y(3) 1 -Y(3)*x(3) -Y(3)*y(3);
    x(4) y(4) 1 0 0 0 -X(4)*x(4) -X(4)*y(4);
    0 0 0 x(4) y(4) 1 -Y(4)*x(4) -Y(4)*y(4);
];
v = [
    X(1); Y(1); X(2); Y(2);
    X(3); Y(3); X(4); Y(4)
];
```

Yes, with this code, I can get the 4 corners of image, as it given in the code snippet in 2.6.b

2.6.e - Results

Figure 20: Warped

2.6.f - Pink Area Detection

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

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

T = maketform('projective', U');
edited_book = imtransform(org_book, T, 'XData',[0 210], 'YData
   ',[0 297]);
figure; imshow(edited_book);

Ihsv = rgb2hsv(edited_book);
H = Ihsv(:,:,1); S = Ihsv(:,:,2); V = Ihsv(:,:,3);
black_area = ((H > 0.8) | (H < 0.1)) & (S > 0.25) & (V > 0.25);

border = 20;
black_area(1:border,:) = 0; black_area(end-border+1:end,:) = 0;
black_area(:,1:border) = 0; black_area(:,end-border+1:end) = 0;

[L, num] = bwlabel(black_area);
stats = regionprops(L, 'Area', 'BoundingBox');
[~, idx] = max([stats.Area]);
```

This algorithm combines perspective correction and color-based detection. The image is first straightened, then a mask is created to find the target area by its color.

- Compute a homography from the selected corner points and apply it to correct the perspective.
- Convert the corrected image from RGB to HSV color space.
- Threshold the HSV channels to create a mask of the target colored area.
- Create a mask(blacked area) to detect pink areas using the condition: `(H > 0.8) | (H < 0.1)` and `S > 0.25` and `V > 0.25`.
- This values getted empirically.
- Label connected regions and find the largest one with `regionprops`.
- Draw a rectangle around the detected area.

This process straightens the image and marks the desired region, though the result depends on how well the thresholds are chosen.

Figure 21: Warped

Figure 22: Pink area detected

2.7 Code Two Perceptrons

```
x1 = [3 3 1];  x2 = [1 1 1];   a = 1;   w =[0 0 0];

%iteration 1
if dot(w',x1) <= 0
    w = w + a * x1;
end
fprintf('  w =[%d %d %d]\n', w);

%iteration 2
if dot(w',x2) >= 0
    w = w - a * x2;
end
fprintf('  w =[%d %d %d]\n', w);

%iteration 3
if dot(w',x1) <= 0
    w = w + a * x1;
end
fprintf('  w =[%d %d %d]\n', w);

%iteration 4
if dot(w',x2) >= 0
    w = w - a * x2;
end
fprintf('  w =[%d %d %d]\n', w);


w2 = [0 0 0];

%iteration 1
w2=w2+a*(1 - dot(w2',x1))*x1;
fprintf('  w2 =[%d %d %d]\n', w2);

%iteration 2
w2=w2+a*(-1 - dot(w2',x2))*x2;
fprintf('  w2 =[%d %d %d]\n', w2);

%iteration 3
w2=w2+a*(1 - dot(w2',x1))*x1;
fprintf('  w2 =[%d %d %d]\n', w2);

%iteration 4
w2=w2+a*(-1 - dot(w2',x2))*x2;
fprintf('  w2 =[%d %d %d]\n', w2);
```

Two perceptron algorithms were implemented based on the lecture slides (Slide L1-part 5.2). The first one is the standard perceptron algorithm (Algorithm 1), and the second one is a slightly modified version (Algorithm 2) where updates are done more aggressively.

We try to find a weight vector w that can separate the given linearly separable classes.

Below results are the recorded weights during each iteration for both algorithms:

Algorithm 1: Misclassified Sample Update

$$\begin{aligned} \boldsymbol{w} &\leftarrow \boldsymbol{w} + \alpha \boldsymbol{x}_k, \quad \text{if } r_k = +1 \text{ and misclassified,} \\ \boldsymbol{w} &\leftarrow \boldsymbol{w} - \alpha \boldsymbol{x}_k, \quad \text{if } r_k = -1 \text{ and misclassified.} \end{aligned}$$

Algorithm 2: Gradient-based Update

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

Table 1: Weight updates during iterations

Iteration	**Algorithm 1** (w)	**Algorithm 2** (w_2)
1	[3, 3, 1]	[3, 3, 1]
2	[2, 2, 0]	[-5, -5, -7]
3	[2, 2, 0]	[109, 109, 31]
4	[1, 1, -1]	[-141, -141, -219]

In Algorithm 1, weights were updated gradually and the changes were relatively small. It tried to correct mistakes step by step. In Algorithm 2, the updates were much larger, so the weights changed very quickly. While this can help in faster learning, it can also cause the weights to grow too large, as seen in the last row. Both algorithms are able to improve the separation, but Algorithm 1 provides more stable updates.