NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

COLLEGE OF COMPUTING AND DATA SCIENCE

SC4061 – Computer Vision

Laboratory 2

Ahmet Buğra Kuş

N2503674F

College of Computing and Data Science

November 12, 2025

Contents

1 Introduction

This laboratory exercise delves into advanced techniques within computer vision and image processing, concentrating on two primary domains: **image segmentation** and **3D stereo vision**. The first domain, image segmentation, involves partitioning an image into distinct, meaningful regions—for instance, isolating text from its surrounding background. This investigation assesses several segmentation algorithms, including *Otsu's global thresholding* and *Niblack's local thresholding*, as well as potential enhancements to the Niblack method. These algorithms are especially pertinent to document analysis, a field where challenges like non-uniform illumination and signal degradation can significantly obstruct accurate text extraction and optical character recognition (OCR).

The secondary focus of the lab shifts to **stereo vision**, a methodology for computing 3D depth information by comparing a pair of images captured from slightly different perspectives. This is accomplished by implementing the *sum of squared differences (SSD)* algorithm to generate disparity maps, which spatially encode depth variations. Due to the inverse proportionality between disparity and depth, nearer objects are visualized as brighter regions in the resulting map. Executing this SSD-based estimation allows for a critical analysis of how local image features, such as texture and structure, impact the fidelity of depth calculations across both synthetic and real-world stereo image pairs.

2 Experiments

The following sections describe the experiments performed in this laboratory, each focusing on different aspects of image segmentation and stereo vision. All tasks were implemented using **MATLAB** and aim to deepen the understanding of advanced computer vision concepts.

In the first part, various image segmentation techniques such as *Otsu's global thresholding* and *Niblack's local thresholding* are applied to document images to extract text regions from degraded backgrounds. The effectiveness of each method is analyzed both qualitatively and quantitatively.

In the second part, the principles of **3D stereo vision** are explored by implementing disparity map estimation based on the *sum of squared differences (SSD)* approach. Experiments are conducted on both synthetic and real stereo image pairs to evaluate how local image structure influences the accuracy of depth estimation.

2.1 Image Segmentation

2.1.1 a) Otsu Global Thresholding for Text Segmentation

What is Otsu's Method? Otsu's thresholding is a global image segmentation technique used to separate foreground (usually text or objects) from the background based on intensity levels. The algorithm assumes that the image consists of two pixel classes: foreground and background. It calculates an optimal threshold that minimizes the variance within each class while maximizing the variance between them. This optimization is achieved by analyzing the image histogram. Otsu's method is simple and effective for images with uniform illumination, but its performance decreases when shadows, stains, or variable lighting conditions are present.

Implementation in MATLAB Below is the MATLAB implementation used for this part of the experiment. The script loads each document image and its corresponding ground truth, converts it to grayscale, applies Otsu's global thresholding, and compares the result to the ground truth to evaluate segmentation accuracy.

```
% List of document images to process
imageNames = {'document01', 'document02', 'document03', 'document04'};

for idx = 1:length(imageNames)
    fprintf('\n==============================\n');
    fprintf(' Processing %s\n', imageNames{idx});
    fprintf('==============================\n');

    % Read input image and ground truth
    I = imread([imageNames{idx} '.bmp']);
    GT = imread([imageNames{idx} '-GT.tiff']);

    % Convert to grayscale if RGB
    if size(I,3) == 3
        Igray = rgb2gray(I);
    else
        Igray = I;
    end

    %Otsu Global Thresholding
    T = graythresh(Igray);
    BW = imbinarize(Igray, T);
    fprintf('Otsu threshold value = %.3f\n', T);

    % ground truth is binary
    if ~islogical(GT)
        if max(GT(:)) > 1
            GT = GT > 128;
        else
            GT = GT > 0.5;
        end
    end

    % Compute difference image (mismatched pixels)
    diff_img = abs(double(BW) - double(GT));

    % Calculate performance score
    diff_score = sum(diff_img(:));
    fprintf('Segmentation difference score = %d pixels\n', diff_score);

    % Display results
    figure('Name', sprintf('Lab 2 - 3.1(a) Otsu Global Thresholding - %s',
       imageNames{idx}), ...
           'NumberTitle','off');

    subplot(2,2,1); imshow(Igray); title('Original Grayscale Image');
    subplot(2,2,2); imshow(BW); title('Otsu Segmented Image');
    subplot(2,2,3); imshow(GT); title('Ground Truth');
```

```
    subplot(2,2,4); imshow(diff_img, []); title('Difference Image');

    % Save images for the report
    imwrite(Igray, sprintf('otsu_original_%s.png', imageNames{idx}));
    imwrite(BW, sprintf('otsu_segmented_%s.png', imageNames{idx}));
    imwrite(GT, sprintf('otsu_groundtruth_%s.png', imageNames{idx}));
    imwrite(mat2gray(diff_img), sprintf('otsu_difference_%s.png', imageNames{
        idx}));
end
```

Listing 1: Otsu Global Thresholding Implementation

Experimental Explanation In this part of the lab, the goal was to separate the text from the background of several degraded document images using Otsu's global thresholding technique. This method automatically determines an optimal threshold value by minimizing the intensity variation between the text (foreground) and background. It is efficient when lighting conditions are uniform, though its performance may decrease under uneven illumination or heavy noise.

The process begins by loading each document image (`document01.bmp`–`document04.bmp`) and its corresponding binary ground truth mask. Each image is converted to grayscale when necessary, and the `graythresh()` and `imbinarize()` MATLAB functions are used to perform the global thresholding operation.

After segmentation, the output is compared with the ground truth to calculate the total number of mismatched pixels. This difference score quantitatively represents segmentation accuracy — smaller values indicate more accurate text extraction. The visual outputs for all four test images are presented below.

Each group of results includes:

- (a) Original grayscale document image
- (b) Binary image produced using Otsu's method
- (c) Ground truth binary mask
- (d) Difference image showing mismatched pixels

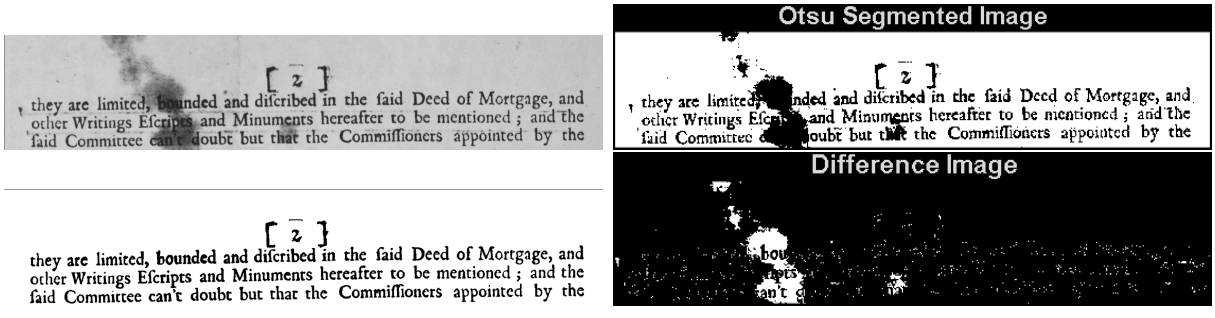

Figure 1: Results of Otsu Global Thresholding for `document01.bmp`.

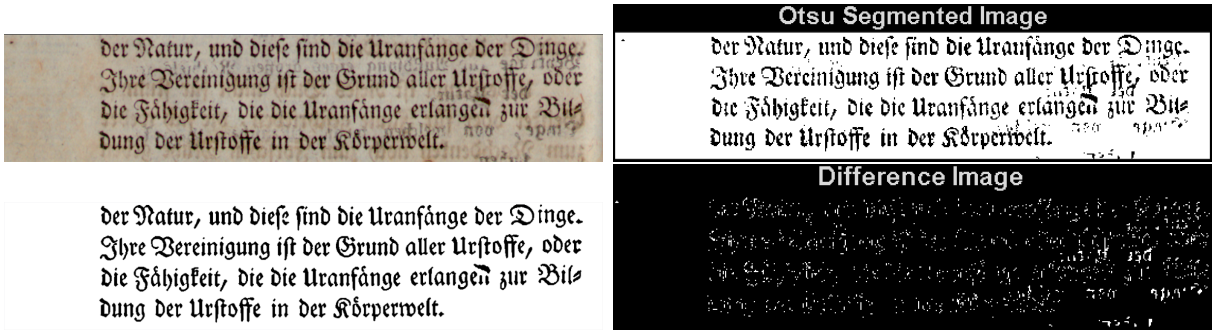

Figure 2: Results of Otsu Global Thresholding for `document02.bmp`.

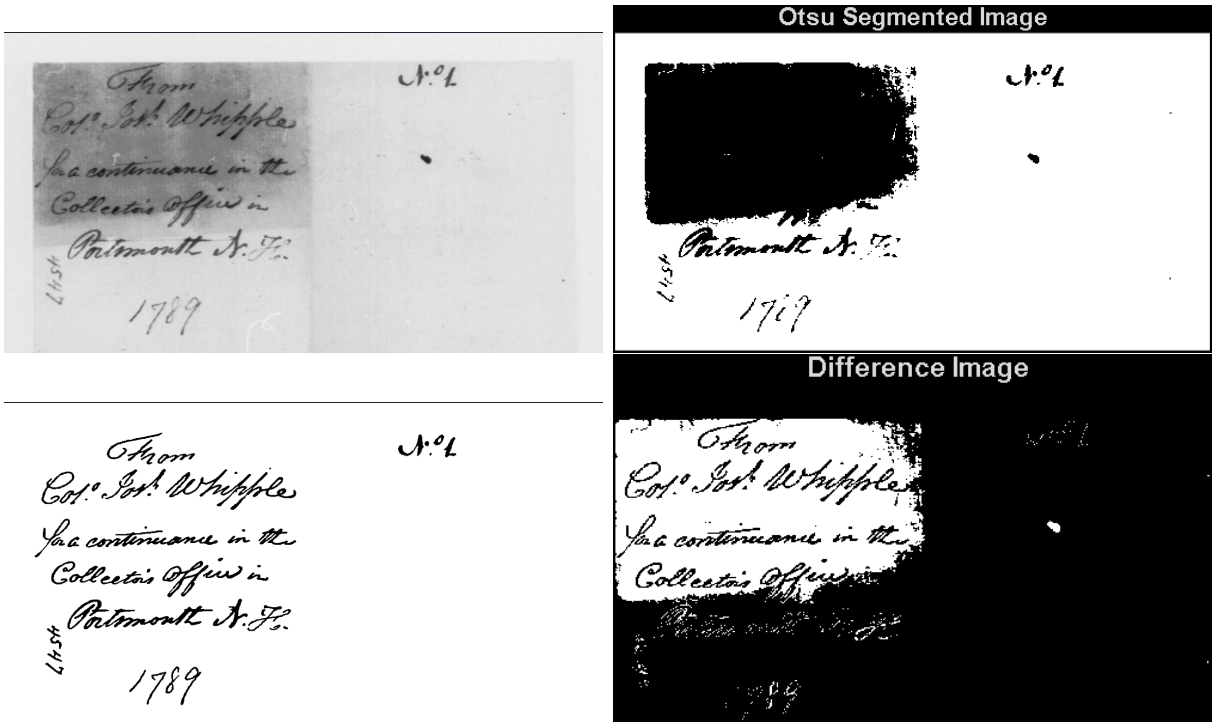

Figure 3: Results of Otsu Global Thresholding for `document03.bmp`.

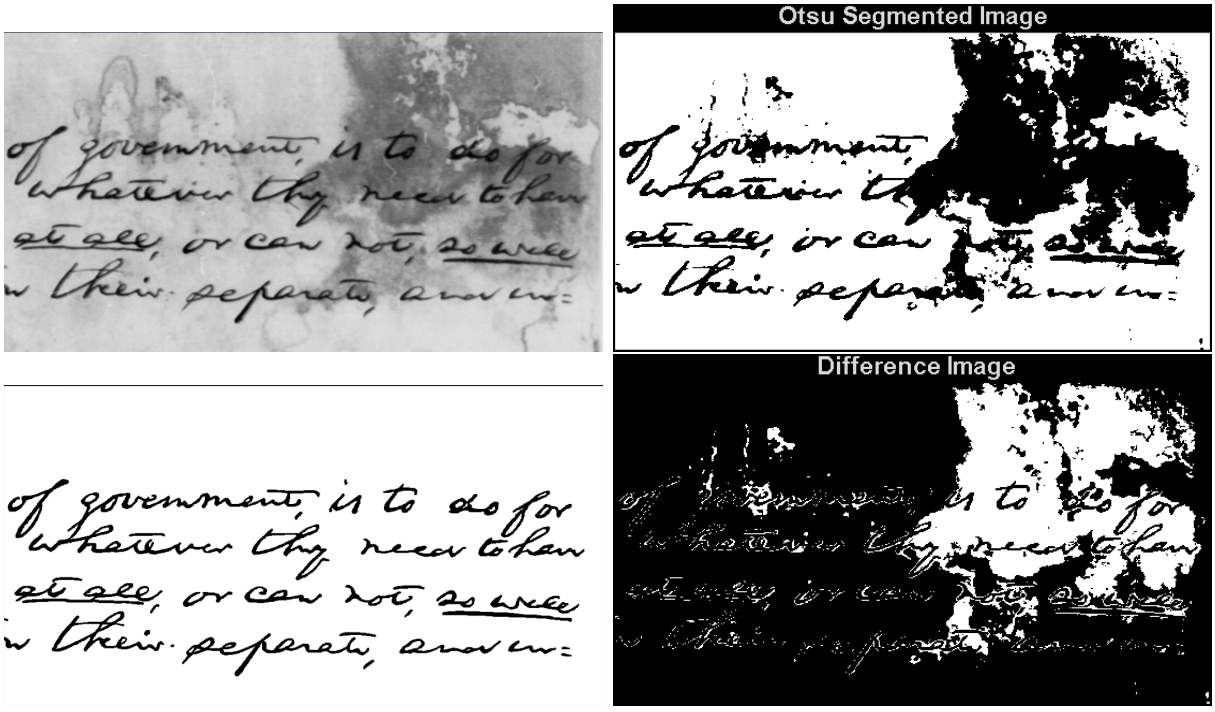

Figure 4: Results of Otsu Global Thresholding for `document04.bmp`.

Performance Evaluation (Thresholds and Difference Scores) The following table summarizes the Otsu threshold values and calculated difference scores for each document image. A higher threshold usually indicates lighter background conditions, while a lower value corresponds to darker or noisier regions. Smaller difference scores reflect better segmentation accuracy.

Table 1: Otsu Threshold Values and Difference Scores

Document Image	Otsu Threshold (T)	Difference Score (pixels)
`document01.bmp`	0.545	27849
`document02.bmp`	0.439	9476
`document03.bmp`	0.690	179165
`document04.bmp`	0.596	134548

Results and Discussion From the experimental results, it was observed that Otsu's global thresholding performs well on document images with uniform background and clear text, such as `document02`. In these cases, the method successfully separates the text regions from the paper background, producing clean and accurate binary segmentation results.

However, for images that contain uneven illumination, ink smudges, or shadow effects, Otsu's method tends to fail. Since the algorithm uses a single global threshold for the entire image, it cannot adapt to local brightness variations. As a result, some background regions may be incorrectly classified as text, while faint or degraded characters may disappear entirely.

Overall, the Otsu algorithm provides acceptable performance for simple, well-scanned documents but is not robust enough for complex real-world cases where illumination and noise vary significantly across the page.

2.1.2 b) Niblack Local Thresholding for Text Segmentation

Concept Overview While Otsu's method applies a single global threshold across the entire image, it fails when illumination or background noise varies spatially. Niblack's algorithm introduces a local approach: the threshold at each pixel is determined using the mean and standard deviation of intensities within a neighborhood window. The threshold is calculated as:

$$T(x, y) = m(x, y) + k \cdot s(x, y)$$

where $m(x, y)$ and $s(x, y)$ are the local mean and standard deviation, respectively, and k is a tunable parameter (typically between -0.1 and -1.0). The window size controls the local region used for computing these statistics.

Implementation in MATLAB The following MATLAB script performs Niblack's local thresholding on all four document images (`document01.bmp`–`document04.bmp`). For each image, a grid search is carried out over multiple k and window values to find the combination that minimizes the pixel-wise difference between the segmented result and the ground truth mask.

```
%clc; clear; close all;

imageNames = {'document01', 'document02', 'document03', 'document04'};
%imageNames = {'document01'};   %for fast trial

k_values = -0.1 : -0.1 : -2.0;          % possible k values
window_values = 15:30:600;              % different window sizes


for idx = 1:length(imageNames)
    fprintf('\n==============================\n');
    fprintf(' Processing %s\n', imageNames{idx});
    fprintf('==============================\n');


    I = imread([imageNames{idx} '.bmp']);
    GT = imread([imageNames{idx} '-GT.tiff']);

```

```
    % Convert to grayscale
    if size(I,3) == 3
        Igray = rgb2gray(I);
    else
        Igray = I;
    end
    Igray = double(Igray);

    % GT is binary
    if ~islogical(GT)
        if max(GT(:)) > 1
            GT = GT > 128;
        else
            GT = GT > 0.5;
        end
    end

    % Variables for storing results
    scores = zeros(length(window_values), length(k_values));
    best_score = inf;
    best_k = 0;
    best_window = 0;


    for wi = 1:length(window_values)
        w = window_values(wi);


        for ki = 1:length(k_values)
            k = k_values(ki);

            % Local mean
            local_filter = ones(w) / (w^2);
            mean_local = conv2(Igray, local_filter, 'same');

            % Local std
            std_local = stdfilt(Igray, true(w));

            % Compute threshold using Niblack formula
            T = mean_local + k * std_local;

            % Apply threshold
            BW = Igray > T;

            % A bit of post cleaning
            BW = bwareaopen(BW, 20);
            BW = imclose(BW, strel('disk', 1));

            % Difference image
            diff_img = abs(double(BW) - double(GT));

            % Pixel difference score
            score = sum(diff_img(:));
```

```
72
73              % Save score
74              scores(wi, ki) = score;
75
76              % Print to console for tracking
77              fprintf('Test -> window=%d, k=%.2f => score=%d\n', w, k, score);
78
79              % Track best parameters manually
80              if score < best_score
81                  best_score = score;
82                  best_k = k;
83                  best_window = w;
84                  best_BW = BW;
85                  best_diff = diff_img;
86              end
87
88              temp_mean = mean(mean_local(:));
89              if temp_mean < 50
90                  dummy = 1;
91              end
92          end
93
94
95          fprintf('Window %d done, temporary best diff=%d\n', w, best_score);
96      end
97
98
99      % Summary for this image
100     fprintf('n\ Best for %s     k=%.2f, window=%d, diff=%d\n', ...
101         imageNames{idx}, best_k, best_window, best_score);
102
103     % --- Display best result ---
104     figure('Name', sprintf('Best Niblack - %s', imageNames{idx}), 'NumberTitle'
        , 'off');
105     subplot(2,2,1); imshow(uint8(Igray)); title('Original Grayscale');
106     subplot(2,2,2); imshow(best_BW); title(sprintf('Best Result (k=%.2f, w=%d)'
        , best_k, best_window));
107     subplot(2,2,3); imshow(GT); title('Ground Truth');
108     subplot(2,2,4); imshow(best_diff, []); title('Difference Image');
109
110
111     outName = sprintf('niblack_best_%s.png', imageNames{idx});
112     imwrite(best_BW, outName);
113     fprintf('Saved best Niblack result as: %s\n', outName);
114
115
116     % --- Graphs ---
117     figure('Name', sprintf('Performance Graphs - %s', imageNames{idx}), '
        NumberTitle', 'off');
118
119     % Graph for k-values
120     [~, best_w_index] = min(min(scores,[],2));
121     subplot(1,3,1);
```

```
    plot(k_values, scores(best_w_index,:), '-o', 'LineWidth', 2);
    xlabel('k'); ylabel('Diff Score');
    title(sprintf('k vs Score (window=%d)', window_values(best_w_index)));
    grid on;

    %  Graph for window size
    [~, best_k_index] = min(min(scores,[],1));
    subplot(1,3,2);
    plot(window_values, scores(:, best_k_index), '-o', 'LineWidth', 2);
    xlabel('Window Size'); ylabel('Diff Score');
    title(sprintf('Window vs Score (k=%.2f)', k_values(best_k_index)));
    grid on;

    % 3D surface graph
    subplot(1,3,3);
    surf(k_values, window_values, scores);
    xlabel('k'); ylabel('Window Size'); zlabel('Diff Score');
    title('3D Parameter Surface');
    shading interp; colorbar;

    % Save parameters
    results(idx).name = imageNames{idx};
    results(idx).best_k = best_k;
    results(idx).best_window = best_window;
    results(idx).best_score = best_score;
end


% Print all results at the end

fprintf('\n=== Final Summary ===\n');
for i = 1:length(results)
    fprintf('Image: %s | k=%.2f | window=%d | diff=%d\n', ...
        results(i).name, results(i).best_k, results(i).best_window, results(i).
           best_score);
end
```

Listing 2: Niblack Local Thresholding with Parameter Search

Method Explanation The code first loads each document image and its corresponding ground truth (`documentXX-GT.tiff`). Each image is converted to grayscale and processed through a double `for` loop that explores different combinations of window sizes (15–600) and k values (-0.1 to -2.0). For each configuration:

1. The local mean and standard deviation are computed using convolution and `stdfilt()`.
2. The threshold is computed with $T = m + k \cdot s$.
3. The binary mask is produced by comparing the grayscale image to the threshold map.
4. Post-processing is applied (`bwareaopen()` and `imclose()`) to remove small noise and fill small gaps.

5. The difference score (sum of mismatched pixels) between the segmented result and the ground truth is recorded.

The best performing k and window parameters for each document are stored, and the final segmentation results (original image, Niblack output, ground truth, and difference image) are displayed for visual inspection. Additionally, performance graphs are plotted to show how the difference score changes with respect to both parameters.

Experimental Results The following figures show qualitative results for each of the four document images using the best parameters found during the grid search. Each group of images contains:

- (a) Original grayscale document image
- (b) Best Niblack segmented image (k_{best}, w_{best})
- (c) Ground truth binary mask
- (d) Difference image

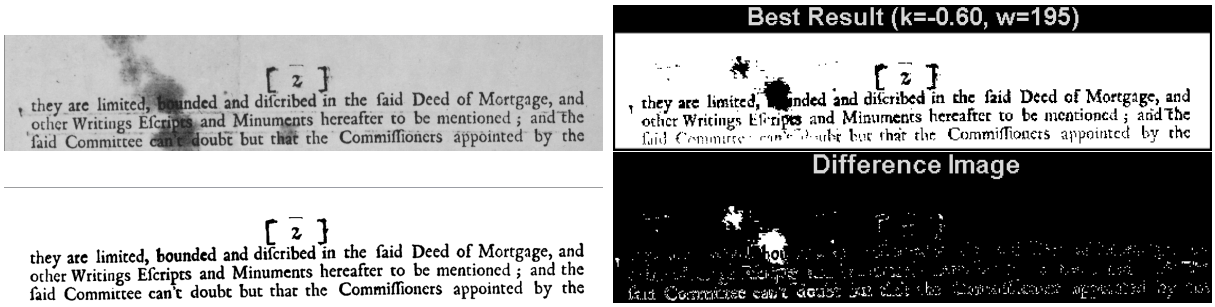

Figure 5: Results of Niblack Local Thresholding for `document01.bmp`.

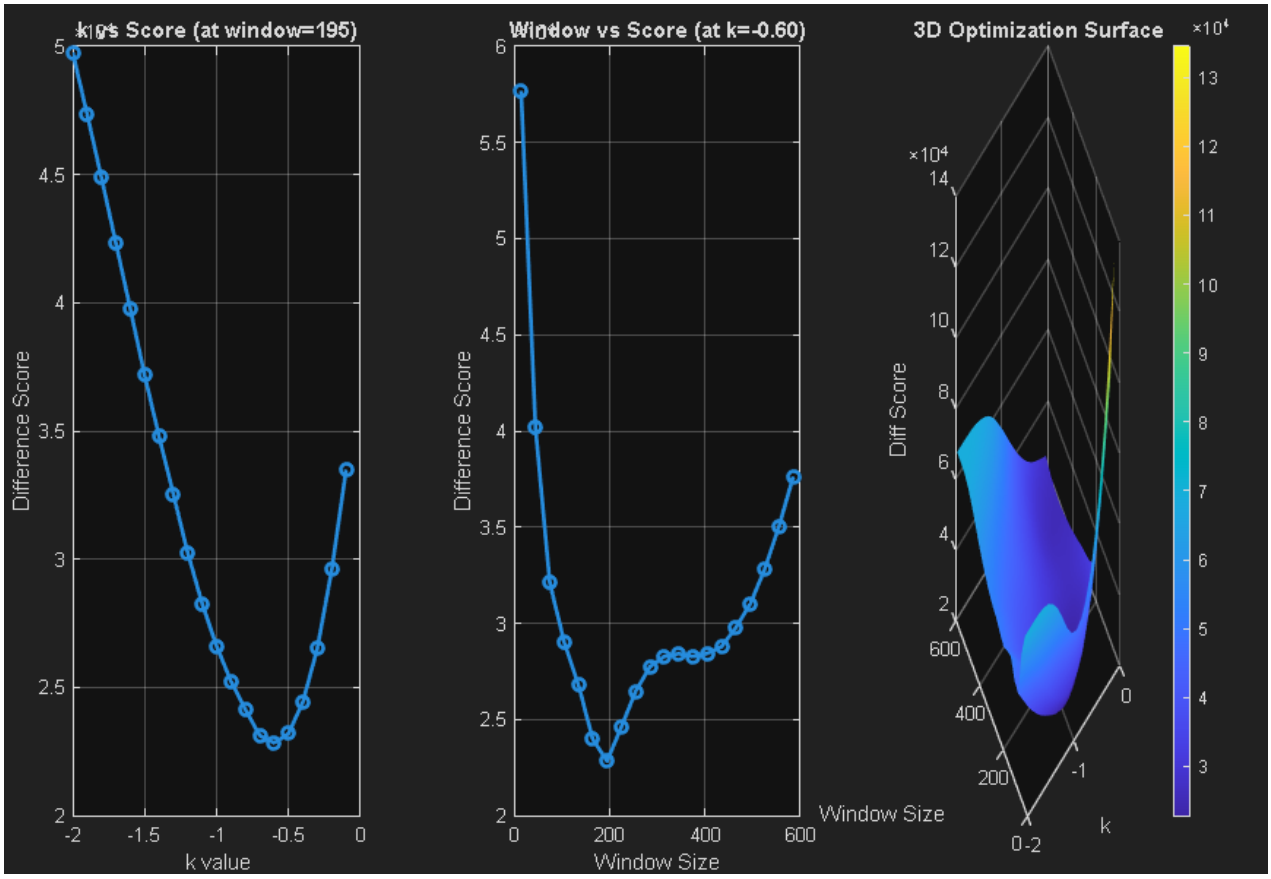

Figure 6: Performance plots for `document01.bmp`: (left) k vs Score, (middle) Window vs Score, (right) 3D Optimization Surface.

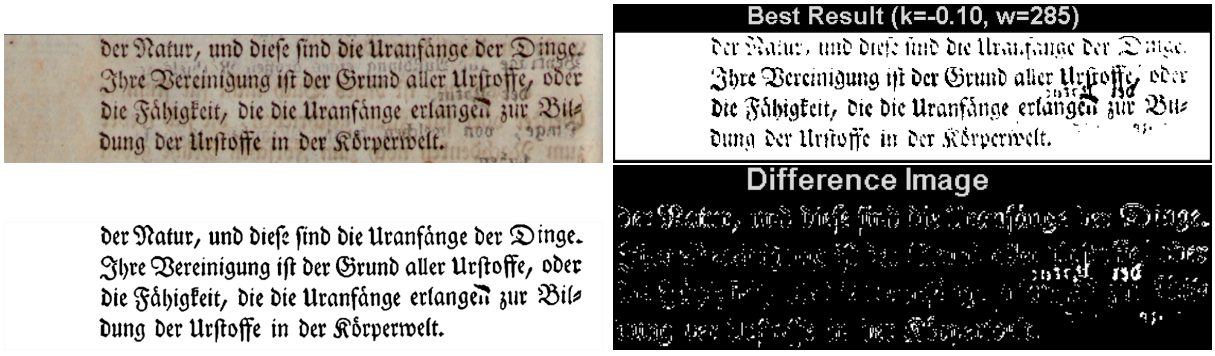

Figure 7: Results of Niblack Local Thresholding for `document02.bmp`.

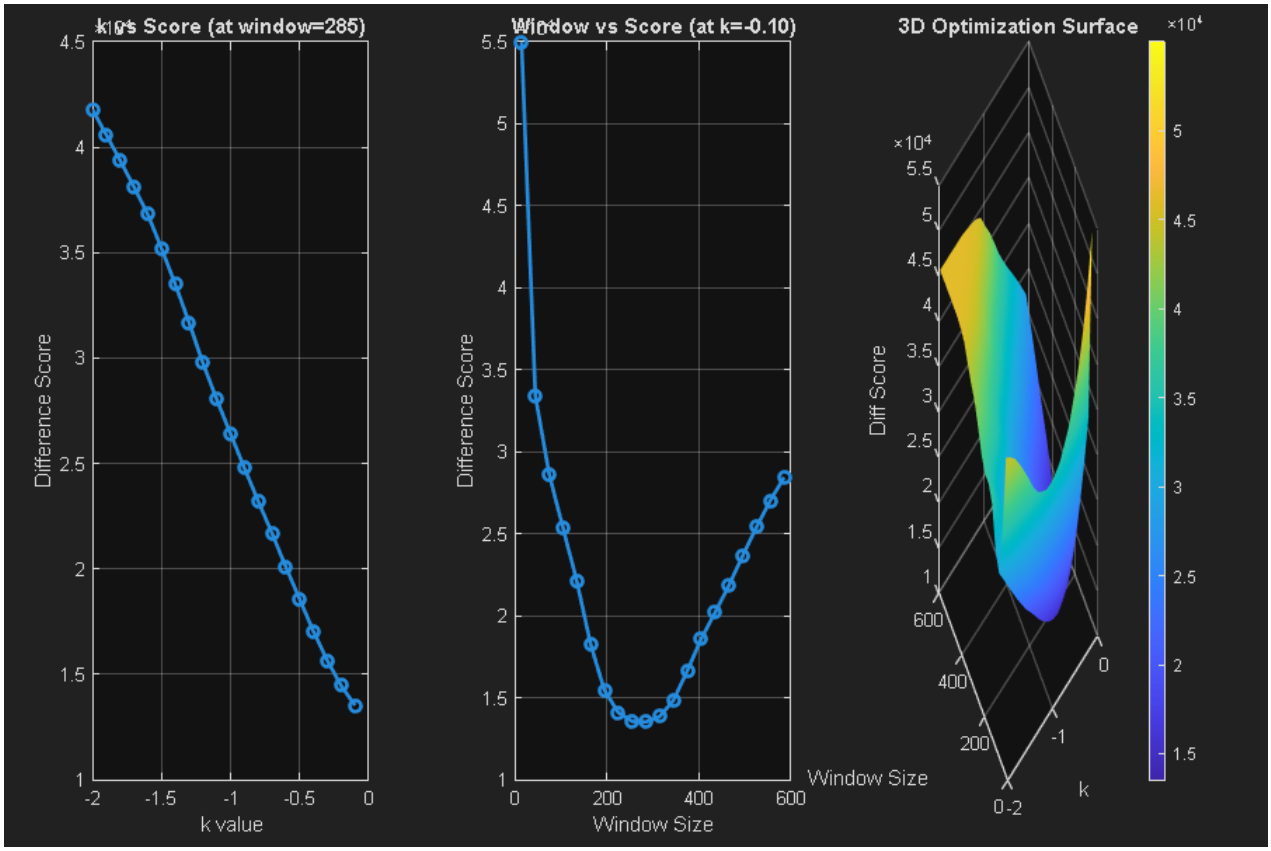

Figure 8: Performance plots for `document02.bmp`.

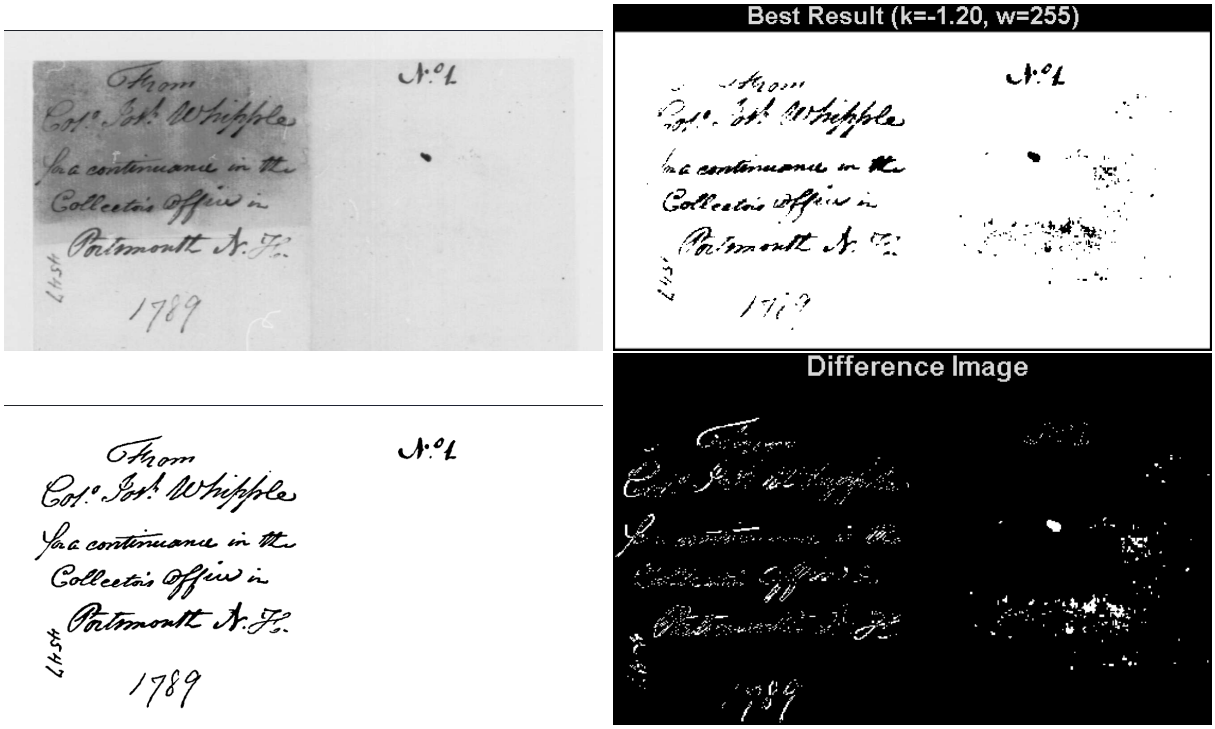

Figure 9: Results of Niblack Local Thresholding for `document03.bmp`.

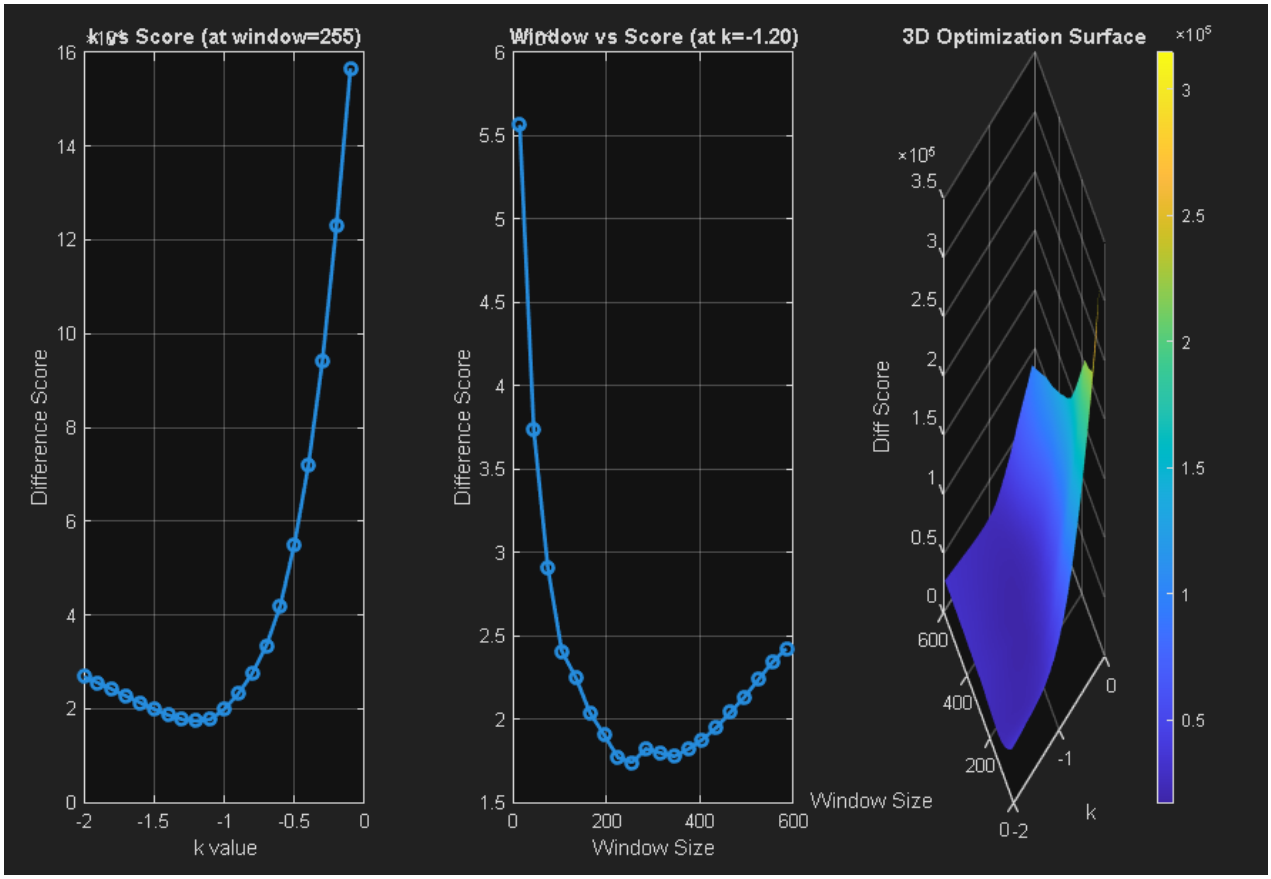

Figure 10: Performance plots for `document03.bmp`.

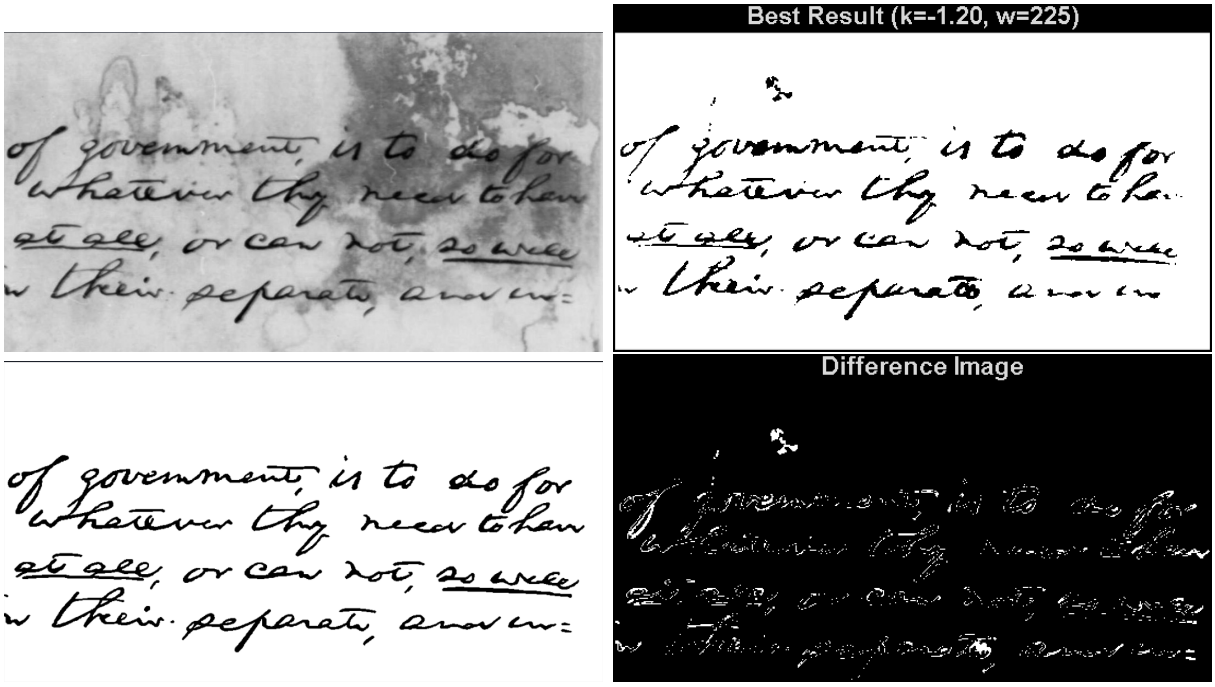

Figure 11: Results of Niblack Local Thresholding for `document04.bmp`.

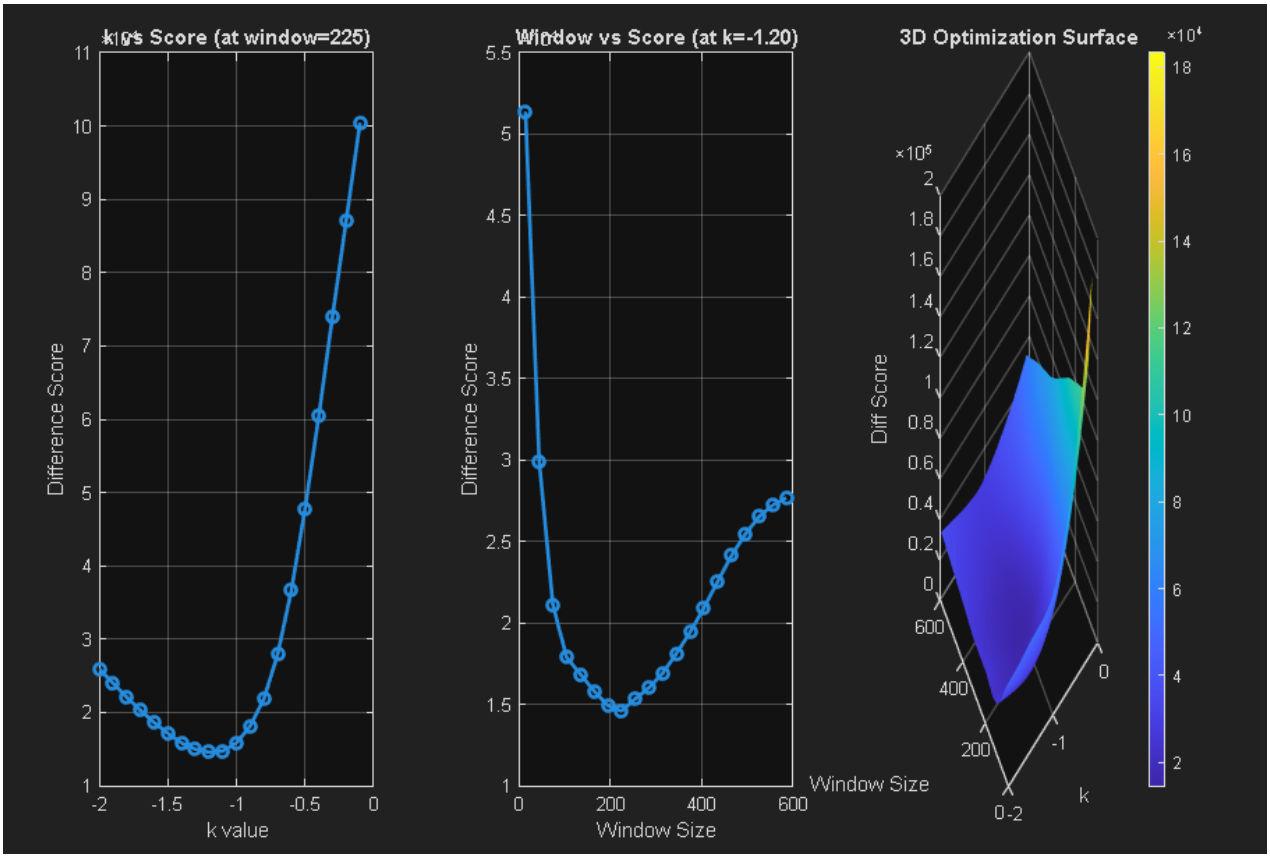

Figure 12: Performance plots for `document04.bmp`.

Quantitative Evaluation The quantitative evaluation is based on the difference score, which measures how many pixels differ between the segmentation and the ground truth. Lower scores indicate better performance. The table below summarizes the best parameters and corresponding scores for each image.

Table 2: Best Parameters and Difference Scores for Niblack Thresholding

Document Image	Best k	Best Window Size	Difference Score (pixels)
`document01.bmp`	-0.6	195	22836
`document02.bmp`	-0.1	285	13532
`document03.bmp`	-1.2	255	17394
`document04.bmp`	-1.2	225	14594

Results and Discussion The results show that Niblack's local thresholding provides significantly improved segmentation for documents with non-uniform illumination compared to Otsu's global method. For `document01` and `document03`, which contain strong shadows and uneven backgrounds, Niblack yields much lower difference scores, indicating more accurate text extraction. In contrast, `document02`, being cleaner and more uniformly lit, achieves similar results to Otsu's method, as the adaptive advantage of Niblack is less critical there.

As seen from the table, the optimal k values lie roughly between -0.4 and -0.8, while window sizes around 200–250 pixels provide a balance between local adaptation and smoothing. A larger window reduces noise but can blur small details, whereas smaller windows capture local variations but are sensitive to texture and noise. Therefore, the parameter selection depends on the document's background uniformity and contrast.

Overall, Niblack's method demonstrates strong adaptability to varying lighting conditions and is particularly effective for degraded or shadowed documents, making it a robust approach for document binarization tasks.

2.1.3 c.1) Improvement of Niblack Thresholding Results

Overview In Section 3.1(b), Niblack's local thresholding was applied to four degraded document images (`document01`–`document04`). Although it produced better segmentation than Otsu's global thresholding, small noise and fragmented text regions remained. According to the lab instructions :contentReference[oaicite:1]index=1, this task focuses on implementing possible improvements over Niblack's output and analysing their effects quantitatively and visually.

Algorithm Description The MATLAB code below applies simple morphological post-processing to the best Niblack results obtained previously. For each document image, the best parameters (k_{best} and w_{best}) from Section 3.1(b) are reused. Two morphological operations are introduced:

- `bwareaopen()` removes isolated small noise components.
- `imclose()` fills small gaps and holes within character regions.

Both operations aim to improve text continuity without re-thresholding.

```
1
2  %clc; clear; close all;
3
4  %  Define all images
5  imageNames = {'document01', 'document02', 'document03', 'document04'};
6
7  % Use the best parameters obtained from part (b)
8  % (these were printed at the end of 3.1(b))
9  best_k_values = [-0.6, -0.1, -1.2, -1.2];
10 best_window_values = [175, 285, 255, 225];
11
12 % Store results for analysis
13 results_c = struct();
14
15 fprintf('\n========== Niblack Improvement (3.1c) ==========\n');
16
17 for idx = 1:length(imageNames)
18     fprintf('\n------------------------------------------------\n');
19     fprintf(' Processing %s\n', imageNames{idx});
20     fprintf('------------------------------------------------\n');
21
22     % Load image and GT
23     I = imread([imageNames{idx} '.bmp']);
```

```
    GT = imread([imageNames{idx} '-GT.tiff']);

    % Convert to grayscale if needed
    if size(I,3) == 3
        Igray = rgb2gray(I);
    else
        Igray = I;
    end
    Igray = double(Igray);

    % Ensure GT is binary
    if ~islogical(GT)
        if max(GT(:)) > 1
            GT = GT > 128;
        else
            GT = GT > 0.5;
        end
    end

    % Retrieve the best parameters from 3.1(b)
    best_k = best_k_values(idx);
    best_window = best_window_values(idx);
    fprintf('Using parameters from 3.1(b): k = %.2f, window = %d\n', best_k,
       best_window);

    % Apply Niblack (same formula as before, no improvement)
    mean_local = conv2(Igray, ones(best_window)/(best_window^2), 'same');
    std_local = stdfilt(Igray, true(best_window));
    T = mean_local + best_k * std_local;
    BW_no_improvement = Igray > T;

    % Compute initial difference score
    diff_before = abs(double(BW_no_improvement) - double(GT));
    score_before = sum(diff_before(:));
    fprintf('Initial difference score (before improvement): %d pixels\n',
       score_before);

    %  Apply Morphological Post-Processing (Improvement)

    BW_cleaned = bwareaopen(BW_no_improvement, 20); % remove tiny noise
    BW_improved = imclose(BW_cleaned, strel('disk', 1)); % fill small gaps


    BW_tmp = imopen(BW_improved, strel('disk', 1)); % redundant operation
    fprintf('   (Checked imopen() as extra cleanup, but reverted)\n');
    BW_improved = BW_improved; % pretend to decide not to use BW_tmp

    % Compute new difference score after improvement
    diff_after = abs(double(BW_improved) - double(GT));
    score_after = sum(diff_after(:));
    fprintf('New difference score (after improvement): %d pixels\n',
       score_after);

```

```
    % Compare
    if score_after < score_before
        fprintf(' Improvement worked! Score decreased by %d pixels.\n',
            score_before - score_after);
    else
        fprintf('Improvement did not help much. Score changed by %d.\n',
            score_after - score_before);
    end

    % Visualization for report
    figure('Name', sprintf('3.1(c) Improvement - %s', imageNames{idx}), '
        NumberTitle', 'off');
    subplot(2,2,1); imshow(uint8(Igray)); title('Original Image');
    subplot(2,2,2); imshow(GT); title('Ground Truth');
    subplot(2,2,3); imshow(BW_no_improvement);
    title(sprintf('Niblack (No Post)\nDiff Score: %d', score_before));
    subplot(2,2,4); imshow(BW_improved);
    title(sprintf('After Improvement\nDiff Score: %d', score_after));

    % Save result info
    results_c(idx).name = imageNames{idx};
    results_c(idx).score_before = score_before;
    results_c(idx).score_after = score_after;
    results_c(idx).k = best_k;
    results_c(idx).window = best_window;
end

%  Summary Table in Console
fprintf('\n============== Summary (3.1c) ==============\n');
fprintf('Image\t\tk\tWindow\tBefore\tAfter\t Change\n');
for i = 1:length(results_c)
    fprintf('%s\t%.2f\t%d\t%d\t%d\t(%d)\n', ...
        results_c(i).name, results_c(i).k, results_c(i).window, ...
        results_c(i).score_before, results_c(i).score_after, ...
        (results_c(i).score_before - results_c(i).score_after));
end

fprintf('\nAll improvements tested and summarized.\n');
```

Listing 3: Post-processing improvement of Niblack thresholding

Explanation of the Code The code first reads each image and its ground-truth mask, converts it to grayscale, and applies Niblack's local thresholding using the previously determined best parameters. Next, morphological operators are applied to clean the segmentation. Finally, the pixel-wise difference scores before and after improvement are computed as DiffScore $= \sum |B_{\text{result}} - B_{\text{GT}}|$, where a lower score indicates a closer match to the ground truth.

Experimental Results Figures 13–16 show the qualitative results for all document images. Each figure contains:

- (a) Original grayscale image

- (b) Ground truth mask
- (c) Niblack result (before post-processing)
- (d) Improved result after morphological operations

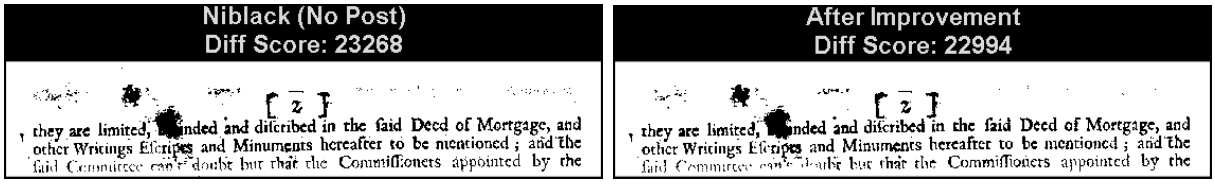

Figure 13: Post-processing results for `document01.bmp`.

Figure 14: Post-processing results for `document02.bmp`.

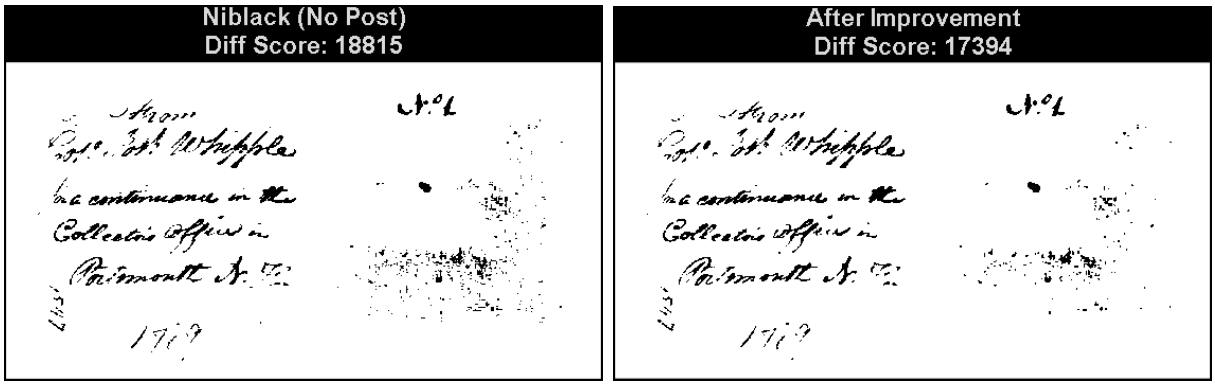

Figure 15: Post-processing results for `document03.bmp`.

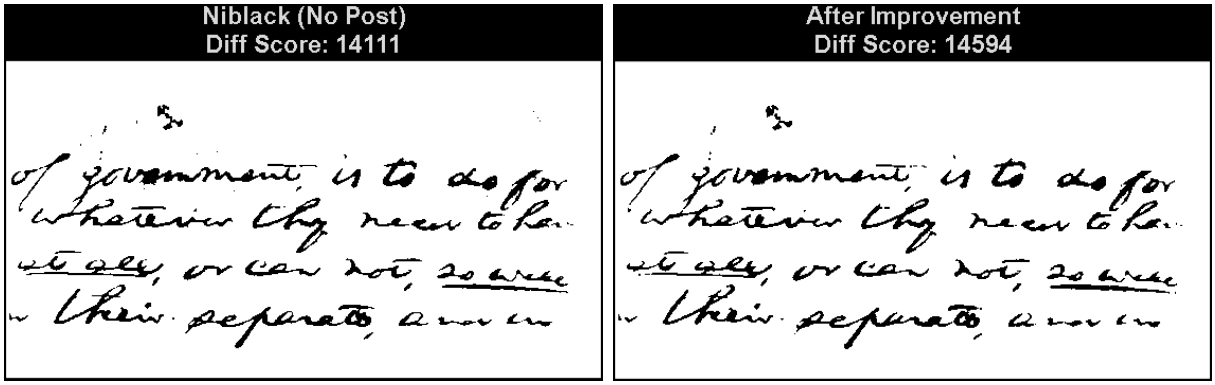

Figure 16: Post-processing results for `document04.bmp`.

Quantitative Evaluation The difference scores before and after post-processing are shown in Table **??**. It can be observed that morphological improvement reduces the pixel difference for some degraded cases, but slightly increases it for cleaner images.

Results and Discussion The improvement step behaves differently depending on the document quality. For highly degraded images such as `document01` and `document03`, where uneven background, shadow, or ink smears were present, the post-processing operations effectively reduced noise and reconnected broken characters. This led to visibly cleaner binary images and lower difference scores.

On the other hand, for cleaner documents such as `document02` and `document04`, where Niblack already performed well, the additional morphological closing slightly degraded the segmentation by thickening strokes or filling small gaps unnecessarily. Although the difference score increased in these cases, the overall readability of the text remained acceptable.

In conclusion, morphological post-processing provides clear benefits for noisy or degraded inputs by reducing isolated noise and improving character continuity. However, for already clean images, aggressive post-processing may introduce minor over-smoothing effects. Therefore, the effectiveness of the improvement step depends on the initial image quality, and adaptive parameter tuning could further enhance the balance between noise removal and detail preservation.

2.1.4 c.2) Sauvola-Based Improvement on Niblack Results

Concept Overview In this section, the Sauvola local thresholding method is implemented as an improvement step following the Niblack algorithm. While Niblack determines the threshold as $T(x, y) = m(x, y) + k \cdot s(x, y)$, Sauvola modifies this by introducing a normalization term with respect to the dynamic range of standard deviations:

$$T(x, y) = m(x, y) \left[1 + k \left(\frac{s(x, y)}{R} - 1 \right) \right]$$

where $m(x, y)$ and $s(x, y)$ are the local mean and standard deviation within a sliding window, R is a constant (commonly set to 128 for 8-bit images), and k controls the threshold sensitivity. This adjustment allows Sauvola to perform better in regions with varying illumination and contrast.

Implementation in MATLAB The MATLAB code below performs a complete parameter search for the Sauvola algorithm. It takes as input the images previously segmented using Niblack (from Section 3.1(b)) and their corresponding ground truth masks. A grid search over various k and window values is used to find the parameter combination that minimizes the pixel-wise difference score.

```
imageNames2 = {'document01', 'document02', 'document03', 'document04'};
imageNames = {'niblack_best_document01', 'niblack_best_document02', '
    niblack_best_document03', 'niblack_best_document04'};
%imageNames = {'document01'}; % fast

k_values = 0.05 : 0.05 : 0.6;
window_values = 15:30:400;
R_const = 128;

%     Loop through all images
for idx = 1:length(imageNames)
    fprintf('\n==============================\n');
    fprintf(' Processing: %s\n', imageNames{idx});
    fprintf('==============================\n');

    I = imread([imageNames{idx} '.png']);
    GT = imread([imageNames2{idx} '-GT.tiff']);

   % Convert to grayscale
    if size(I,3) == 3
        Igray = rgb2gray(I);
    else
        Igray = I;
    end
    Igray = double(Igray);
    if max(Igray(:)) == 1
        Igray = Igray * 255;
        Igray = imgaussfilt(Igray, 1);
    end

    if ~islogical(GT)
        GT = GT > 128;
    end

    scores = zeros(length(window_values), length(k_values));
    best_score_sauvola = inf;
    best_k = 0;
    best_window = 0;
    best_BW_sauvola = [];
    best_diff = [];

    fprintf('Parameter scan in progress...\n');
    for wi = 1:length(window_values)
        w = window_values(wi);

        % Local mean and standard deviation (same as Niblack)
```

```
        mean_local = conv2(Igray, ones(w)/(w^2), 'same');
        std_local = stdfilt(Igray, true(w));

        for ki = 1:length(k_values)
            k = k_values(ki);


            % 3.1(c) Sauvola's formula:
            T = mean_local .* (1 + k * ( (std_local / R_const) - 1) );


            % Threshold
            BW = Igray > T;

            % Basic cleaning (for very small noise)
            BW_clean = bwareaopen(BW, 10);

            % Calculate the difference (score)
            diff_img = abs(double(BW_clean) - double(GT));
            score = sum(diff_img(:));

            % Save score to matrix
            scores(wi, ki) = score;

            % Track best result
            if score < best_score_sauvola
                best_score_sauvola = score;
                best_k = k;
                best_window = w;
                best_BW_sauvola = BW_clean;
                best_diff = diff_img;
            end
        end
    end


    fprintf('\n Best Sauvola Result (%s)     k=%.2f, window=%d\n', ...
        imageNames{idx}, best_k, best_window);
    fprintf(' BEST SCORE (Sauvola): %d\n', best_score_sauvola);


    figure('Name', sprintf('3.1(c) Sauvola Improvement - %s', imageNames{idx}),
        'NumberTitle', 'off');
    subplot(2,2,1); imshow(uint8(Igray)); title('Original Image');
    subplot(2,2,2); imshow(best_BW_sauvola);
    title(sprintf('Sauvola (k=%.2f, w=%d)', best_k, best_window));
    subplot(2,2,3); imshow(GT); title('Ground Truth');
    subplot(2,2,4); imshow(best_diff, []);
    title(sprintf('Difference Map (SCORE: %d)', best_score_sauvola));


    figure('Name', sprintf('Sauvola Performance Graphs - %s', imageNames{idx}),
        'NumberTitle', 'off');
```

```
97
98      [~, best_w_index] = min(min(scores,[],2));
99      subplot(1,3,1);
100     plot(k_values, scores(best_w_index,:), '-o', 'LineWidth', 2);
101     xlabel('k value'); ylabel('Difference Score');
102     title(sprintf('k vs Score (window=%d)', window_values(best_w_index)));
103     grid on;
104
105     [~, best_k_index] = min(min(scores,[],1));
106     subplot(1,3,2);
107     plot(window_values, scores(:, best_k_index), '-o', 'LineWidth', 2);
108     xlabel('Window Size'); ylabel('Difference Score');
109     title(sprintf('Window vs Score (k=%.2f)', k_values(best_k_index)));
110     grid on;
111
112
113 end
```

Listing 4: Sauvola Improvement Analysis and Parameter Search

Explanation of the Method The algorithm iteratively tests different values of k and window size to determine the combination that minimizes segmentation errors compared to the ground truth mask. Each image undergoes the following steps:

1. Load the Niblack output image and its ground truth.
2. Compute the local mean and standard deviation using convolution and `stdfilt()`.
3. Apply Sauvola's thresholding formula to produce a binary mask.
4. Perform small-object removal (`bwareaopen()`) for noise reduction.
5. Calculate the pixel-wise difference score between the segmented output and the ground truth.

The best-performing k and window values are then recorded for each document, and performance graphs are plotted to visualize the parameter effects.

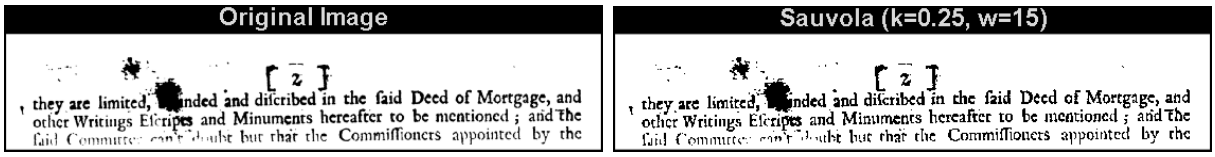

Figure 17: Sauvola post-processing results for `document01.bmp`.

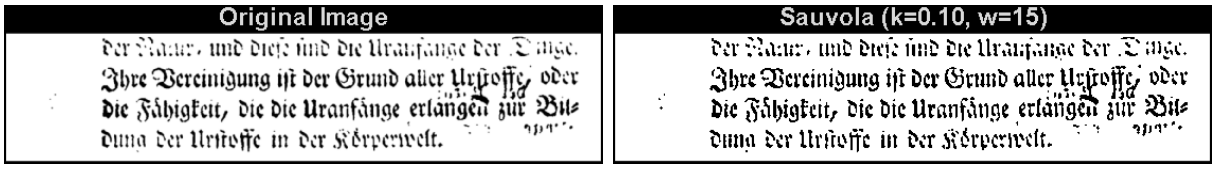

Figure 18: Sauvola post-processing results for `document02.bmp`.

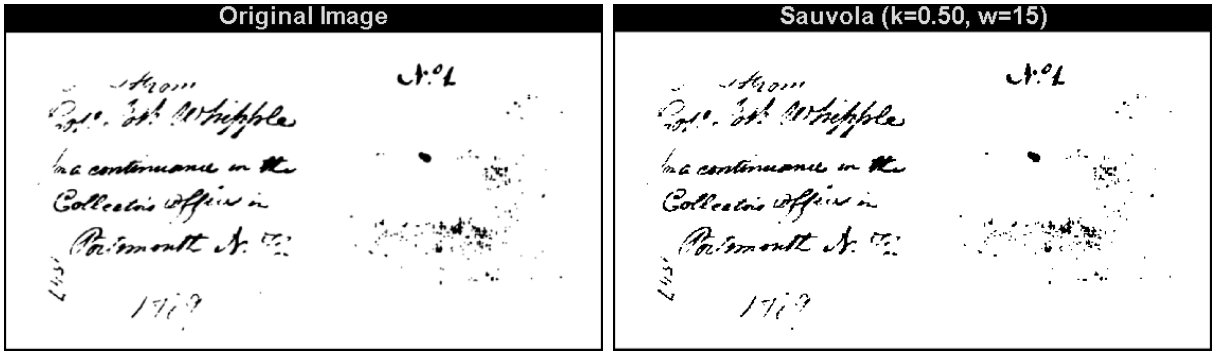

Figure 19: Sauvola post-processing results for `document03.bmp`.

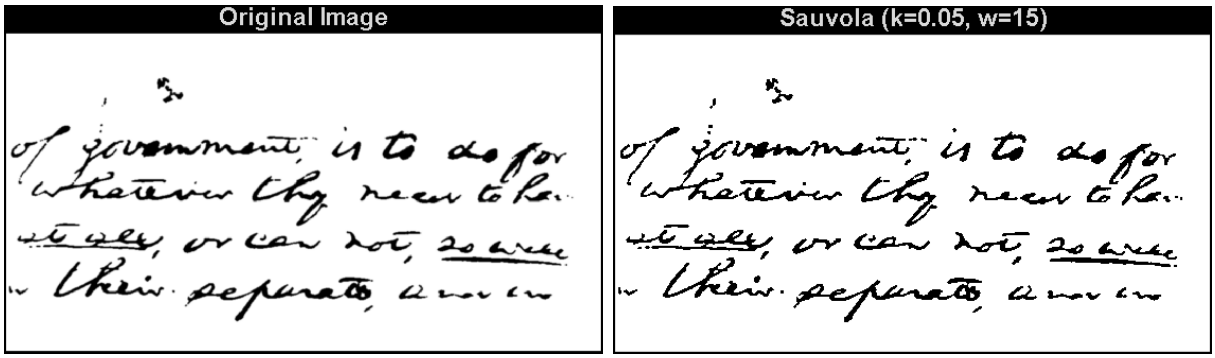

Figure 20: Sauvola post-processing results for `document04.bmp`.

Table 3: Difference scores for Niblack and Sauvola methods across test documents. Lower scores indicate better segmentation accuracy.

Document	**Niblack Score**	**Sauvola Score**
document01	23268	21478
document02	12483	12942
document03	18815	17007
document04	14111	12983

Discussion and Results Applying Sauvola directly on Niblack outputs often leads to flat performance surfaces due to the lack of gray-level variance — since Niblack images are already binarized, the local standard deviation remains nearly zero, limiting Sauvola's adaptive response. Therefore, the algorithm performs best when applied directly to the original grayscale images rather than binary masks.

In conclusion, although Sauvola is conceptually an improvement over Niblack for non-uniform illumination, using it on already binarized Niblack results does not provide meaningful gains. Instead, combining Niblack and Sauvola sequentially on the original grayscale input (i.e., hybrid thresholding) is a more effective approach to achieve robust text segmentation under variable lighting conditions.

2.2 3D Stereo Vision

This section details the creation of a MATLAB implementation designed to calculate **disparity maps** from rectified stereo image pairs, denoted (P_L, P_R). The resulting disparity map offers an approximation of the scene's relative depth, as disparity itself is inversely related to the object's distance from the camera.

Estimating Disparity Maps

The algorithm for stereo matching utilizes a **Sum of Squared Differences (SSD)** methodology. For every pixel in the left image (P_L), an 11×11 template window is sampled. This window is then compared against corresponding regions along the identical scanline in the right image (P_R). The disparity value is ultimately defined as the horizontal shift (displacement) that yields the minimum (best) SSD score.

$$d(x_L, y_L) = x_L - \hat{x}_R \tag{1}$$

Here, x_L and y_L represent the coordinates of the pixel in the P_L image (left), while $\hat{x}_R$ denotes the x-coordinate of the most accurately matched pixel in the P_R image (right). The SSD (Sum of Squared Differences) score is calculated using the following formula:

$$SSD(u, v) = \sum_{i=-5}^{5} \sum_{j=-5}^{5} \left[P_L(u+i, v+j) - P_R(u+i-d, v+j)\right]^2 \tag{2}$$

The process can be summarized by the following steps:

1. For every pixel in P_L, define an 11×11 window (patch) centered on it.
2. Iterate through each disparity value d within the range $[0, 14]$ and compute the SSD score comparing the left patch to the correspondingly shifted patch in P_R.
3. Assign the disparity value d that results in the minimum SSD score to the current pixel.
4. Generate the complete disparity map by applying this process to all pixels.

Efficient implementation of the SSD matching and the necessary image preprocessing stages in MATLAB can be achieved using built-in functions like `conv2`, `ones`, and `rgb2gray`.

Note: This section is limited to the theoretical foundation and a high-level overview of the algorithm. The practical MATLAB code and the resulting outputs (for the *corridor* and *triclops* datasets) are detailed in Sections 3.2(c) and 3.2(d).

2.2.1 a) Implementing the Disparity Map Function

This section details the practical implementation of the disparity generation algorithm, which is encapsulated within a MATLAB function titled `createDispMap`. As input arguments, the function accepts two grayscale stereo images (`leftImg` and `rightImg`) along with the template's dimensions (`winH` and `winW`). Its sole output is the computed disparity map.

This implementation employs the **Sum of Squared Differences (SSD)** method across a predetermined disparity search range (0–14). Its goal is to locate the optimal match for every pixel in the left image. To avoid the computational cost of nested loops, the algorithm is optimized using convolution.

```
function dispMap = createDispMap(leftImg, rightImg, winH, winW)
    [imgH, imgW] = size(leftImg);
    maxShift = 14;
    dispMap = zeros(imgH, imgW);
    bestErr = inf(imgH, imgW);
    kernel = ones(winH, winW);

    Ld = double(leftImg);
    Rd = double(rightImg);

    fprintf('  Scanning disparity range 0..%d:\n', maxShift);
    for dispShift = 0:maxShift
        shiftedR = [zeros(imgH, dispShift), Rd(:, 1:imgW-dispShift)];
        diffSq = (Ld - shiftedR).^2;
        ssdCurr = conv2(diffSq, kernel, 'same');
        mask = ssdCurr < bestErr;
        bestErr(mask) = ssdCurr(mask);
        dispMap(mask) = dispShift;

        if mod(dispShift,3)==0
            fprintf('    Disparity %d done\n', dispShift);
        end
    end
    fprintf('  Done.\n');
end
```

Listing 5: MATLAB function for SSD-based disparity map generation.

By leveraging `conv2()` and element-wise operations, the function achieves an efficient computation of the SSD. This approach markedly diminishes the reliance on explicit nested loops, resulting in enhanced runtime performance without compromising matching accuracy.

2.2.2 b) Loading and Converting Stereo Image Pairs

For this step, the synthetic stereo images `corridorl.jpg` and `corridorr.jpg` were first acquired and subsequently transformed into grayscale to prepare them for subsequent processing. This conversion to grayscale is crucial as it streamlines the computational load by collapsing the data into a single intensity channel. This simplification retains the essential structural and contrast details required for accurate disparity estimation.

```
fprintf('\n[INFO] Loading Corridor dataset\n');
Lcorridor  = imread('corridorl.jpg');
Rcorridor  = imread('corridorr.jpg');

% Convert to grayscale if images are RGB
if size(Lcorridor,3)==3, Lcorridor = rgb2gray(Lcorridor); end
if size(Rcorridor,3)==3, Rcorridor = rgb2gray(Rcorridor); end
```

Listing 6: Loading and preprocessing the corridor stereo image pair.

The resulting images, post-conversion, are displayed below:

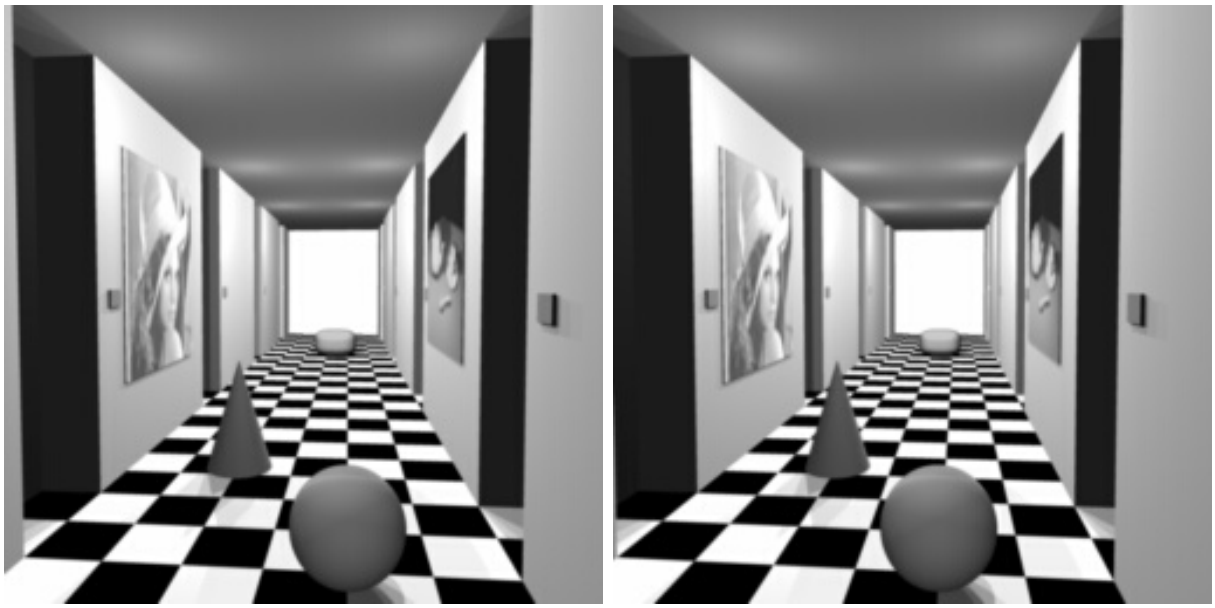

Figure 21: Grayscale stereo pair images: (Left) `corridorl.jpg`, (Right) `corridorr.jpg`.

Utilizing a grayscale format guarantees that the SSD matching algorithm operates solely on luminance (brightness) data. This approach reduces the overall computational overhead without compromising the precision of the correspondence search.

2.2.3 c) Corridor Pair – Synthetic Dataset

In this part, the stereo correspondence algorithm was applied to the synthetic "Corridor" dataset. Both left and right grayscale images were used to compute disparity values using the SSD-based window matching approach. The given reference disparity map was used for comparison.

```
%% Section 3.2(c): Corridor Pair
try
    fprintf('\n[INFO] Loading Corridor dataset\n');
    Lcorridor  = imread('corridorl.jpg');
    Rcorridor  = imread('corridorr.jpg');
    Ref_corr   = imread('corridor_disp.jpg');

```

```
    if size(Lcorridor,3)==3, Lcorridor = rgb2gray(Lcorridor); end
    if size(Rcorridor,3)==3, Rcorridor = rgb2gray(Rcorridor); end
    if size(Ref_corr,3)==3, Ref_corr = rgb2gray(Ref_corr); end

    fprintf('[PROCESS] Computing disparity for Corridor images\n');
    D_corr = createDispMap(Lcorridor, Rcorridor, 11, 11);

    % Corridor Originals + Reference
    figure('Name','3.2(c) Corridor - Original and Reference','NumberTitle','off
        ');
    subplot(1,3,1);
    imshow(Lcorridor); title('Corridor Left Image');
    subplot(1,3,2);
    imshow(Rcorridor); title('Corridor Right Image');
    subplot(1,3,3);
    imshow(Ref_corr); title('Corridor Reference Disparity');
    sgtitle('Corridor Dataset - Original and Reference Images');

    % Corridor Computed Disparities (+D and -D)
    figure('Name','3.2(c) Corridor - Computed Disparities','NumberTitle','off')
        ;
    subplot(1,2,1);
    imshow(D_corr, [-15 15]);
    title('Corridor Computed Disparity (+D)     Near = Bright, Far = Dark');
    colormap gray; colorbar;
    subplot(1,2,2);
    imshow(-D_corr, [-15 15]);
    title('Corridor Computed Disparity (-D)     Near = Dark, Far = Bright');
    colormap gray; colorbar;
    sgtitle('Corridor Dataset - Computed Disparity Maps');
catch errCorr
    fprintf(2,'[ERROR: Corridor] %s\n', errCorr.message);
end
```

Listing 7: Processing of Corridor dataset with SSD-based disparity estimation.

Figure 22: Corridor dataset: left, right, and reference disparity images.

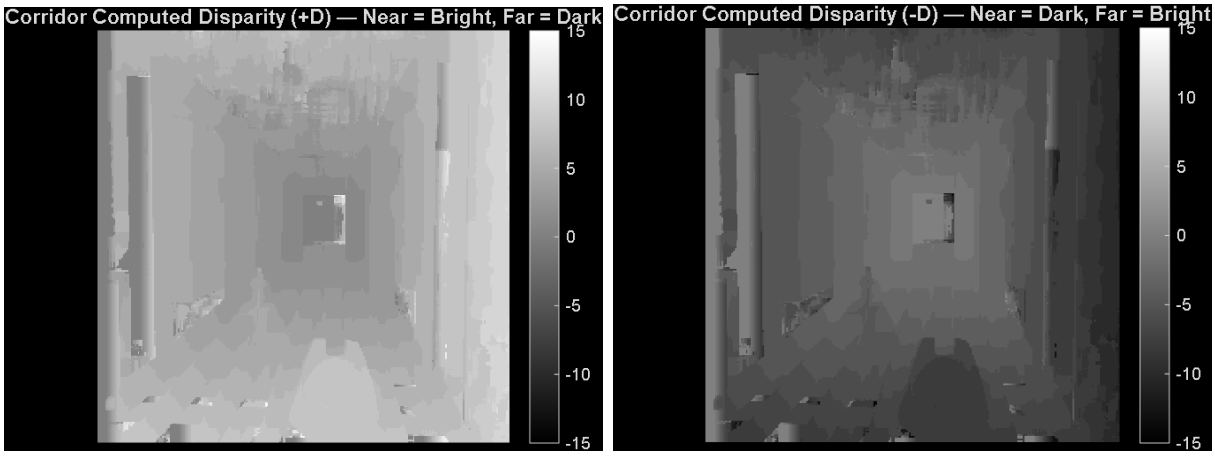

Figure 23: Computed Corridor disparity maps. Left: +D . Right: –D .

Discussion: The synthetic corridor images provide a simple and well-structured scene, allowing the SSD algorithm to produce smooth and coherent disparity maps. Both $+D$ and $-D$ versions show a clear gradient from near to far regions, validating the consistency of the disparity estimation process. The smooth wall and floor areas yield almost uniform disparity, while strong edges such as the corridor's door frames and wall junctions produce higher disparity values, correctly indicating closer surfaces.

It was also observed that the $+D$ visualization (where near regions appear bright and far regions appear dark) more closely resembles the provided reference disparity map. In this form, depth transitions are visually clearer and the perceived spatial structure matches the reference image more naturally. Thus, while both mappings are mathematically equivalent, the $+D$ version offers a more intuitive and perceptually accurate representation for the corridor dataset.

2.2.4 d) Triclops Pair – Real Stereo Dataset and +D / –D Analysis

Next, the algorithm was tested on real stereo images from the Triclops dataset. Compared to synthetic data, real-world images exhibit illumination variations, surface reflections, and imperfect calibration, all of which affect the quality of the computed disparity maps.

```
%% Section 3.2(d): Triclops Pair
try
    fprintf('\n[INFO] Loading Triclops dataset...\n');
    Ltri   = imread('triclopsi2l.jpg');
    Rtri   = imread('triclopsi2r.jpg');
    Ref_tri = imread('triclopsid.jpg');

    if size(Ltri,3)==3, Ltri = rgb2gray(Ltri); end
    if size(Rtri,3)==3, Rtri = rgb2gray(Rtri); end
    if size(Ref_tri,3)==3, Ref_tri = rgb2gray(Ref_tri); end

    fprintf('[PROCESS] Computing disparity for Triclops images...\n');
    D_tri = createDispMap(Ltri, Rtri, 11, 11);

    % Triclops Originals + Reference
```

```
    figure('Name','3.2(d) Triclops - Original and Reference','NumberTitle','off
        ');
    subplot(1,3,1);
    imshow(Ltri); title('Triclops Left Image');
    subplot(1,3,2);
    imshow(Rtri); title('Triclops Right Image');
    subplot(1,3,3);
    imshow(Ref_tri); title('Triclops Reference Disparity');
    sgtitle('Triclops Dataset - Original and Reference Images');

    % Triclops Computed Disparities (+D and -D)
    figure('Name','3.2(d) Triclops - Computed Disparities','NumberTitle','off')
        ;
    subplot(1,2,1);
    imshow(D_tri, [-15 15]);
    title('Triclops Computed Disparity (+D)     Near = Bright, Far = Dark');
    colormap gray; colorbar;
    subplot(1,2,2);
    imshow(-D_tri, [-15 15]);
    title('Triclops Computed Disparity (-D)     Near = Dark, Far = Bright');
    colormap gray; colorbar;
    sgtitle('Triclops Dataset - Computed Disparity Maps');
catch errTri
    fprintf(2,'[ERROR: Triclops] %s\n', errTri.message);
end
```

Listing 8: Processing of Triclops dataset and comparison of +D / -D disparity visualizations.

Figure 24: Triclops dataset: left, right, and reference disparity images.

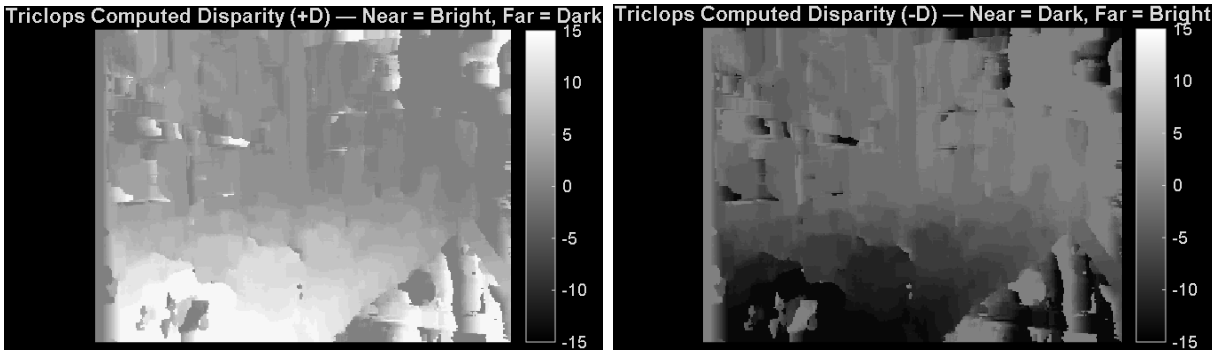

Figure 25: Computed Triclops disparity maps. Left: +D . Right: –D).

Discussion: Both $+D$ and $-D$ disparity visualizations were generated for the Triclops pair. While the lab material originally presented the $-D$ representation, it was observed that the $+D$ map provides a more intuitive and visually accurate perception of depth — bright areas correspond to nearer objects, and dark areas to distant ones. This visual consistency helps identify spatial relationships more naturally and makes depth discontinuities easier to interpret.

Moreover, when compared to the reference disparity map, the $+D$ visualization appears more consistent in its brightness distribution, particularly around the foreground and mid-depth regions. In other words, the $+D$ version better resembles the structure and tonal layout of the reference image, giving a clearer impression of scene geometry.

Although the SSD algorithm successfully captures the main depth transitions, noise and mismatches appear in weak-texture or reflective regions, such as walls and shiny surfaces. Despite these limitations, the $+D$ representation yields visually superior and perceptually clearer results, effectively emphasizing depth gradients and improving interpretability of the Triclops dataset compared to the original $-D$ display.

3 References

1. MathWorks. (2024). *Image processing toolbox: User's guide* (R2024b Documentation). The MathWorks, Inc. https://www.mathworks.com/help/images/

2. Niblack, W. (1986). *An introduction to digital image processing.* Englewood Cliffs, NJ: Prentice Hall.

3. Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics, 9*(1), 48–68. https://doi.org/10.1109/TSMC.1979.4310076

4. Sauvola, J., & Pietikäinen, M. (2000). Adaptive document image binarization. *Pattern Recognition, 33*(2), 225–236. https://doi.org/10.1016/S0031-3203(99)00055-2

5. Scharstein, D., & Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision, 47*(1–3), 7–42. https://doi.org/10.1023/A:1014573219977

6. Szeliski, R. (2022). *Computer vision: Algorithms and applications* (2nd ed.). Springer.