SC4061 – Computer Vision

Laboratory 2 Report

Muhammed Ikbal Özbey

N2504208D

College of Computing and Data Science

November 12, 2025

Contents

1 Introduction

This report outlines the experiments completed for Lab 2, which is divided into two main parts: Image Segmentation and 3D Stereo Vision.

The first part focuses on image segmentation. The task is to write code that can separate text from the background in several sample document images. To do this, we experiment with two different approaches:

- **Otsu's global thresholding**
- **Niblack's local thresholding**

We evaluate how well each method performs by comparing the results to the provided "ground truth" images.

The second part of the lab deals with 3D Stereo Vision. The goal is to create a disparity map, which shows the depth of objects in a scene, by comparing a left and a right image. We do this by implementing a function that uses the Sum-of-Squares Difference (SSD) method. This function finds the best match for a pixel from the left image along the same line in the right image. We test our algorithm on both synthetic ("corridor") and real ("triclops") stereo images.

2 Experiments/Lab

This section details the implementation and evaluation of the two main tasks. For segmentation, we will apply and analyze Otsu's global and Niblack's local thresholding algorithms on document images. For stereo vision, we will write code to compute a disparity map by matching templates between left and right images using the Sum-of-Squares Difference (SSD) method.

2.1 Image Segmentation

2.1.1 Otsu Global Thresholding for Text Segmentation

Otsu's method is a global thresholding technique used to automatically separate the foreground (objects or text) from the background in an image. It works by analyzing the image histogram and finding the threshold value that minimizes the variance within each class (foreground and background), or equivalently, maximizes the variance between them.

This means the algorithm chooses a threshold where the two groups of pixels — dark and light — are as distinct as possible. It is simple and effective for images with bimodal histograms (two clear intensity peaks), but may perform poorly when the image has uneven lighting or noise.

The Code Implamentation

```matlab
I = imread('document01.bmp');
GT = imread('document01-GT.tiff');

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

% Compute Otsu threshold (range 0-1)
T = graythresh(Igray);                  % Automatically finds best global threshold
fprintf('Otsu threshold = %.3f\n', T);

BW = imbinarize(Igray, T);

% Compute difference image (absolute pixel-wise difference)
diff_img = abs(double(BW) - double(GT));

% Quantitative evaluation (sum of differences)
difference_score = sum(diff_img(:));
fprintf('Segmentation difference score = %d pixels\n', difference_score);

% Display results
figure;
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');
```

Listing 1: Otsu's Tresholding

Experimental Explanation

In this part of the lab, Otsu's global thresholding method was used to separate the text from the background in several document images. The method automatically determines an optimal threshold by minimizing the variance within the foreground and background pixel groups.

Each document image (`document01.bmp`–`document04.bmp`) and its corresponding binary ground truth mask were loaded into MATLAB. If the image was in RGB format, it was first converted to grayscale. The `graythresh()` and `imbinarize()` functions were then used to apply Otsu's thresholding and produce a binary segmented image. The visual outputs for each test image include:

(a) Original grayscale document image

(b) Binary image produced using Otsu's method

(c) Ground truth binary mask

(d) Difference image showing mismatched pixels

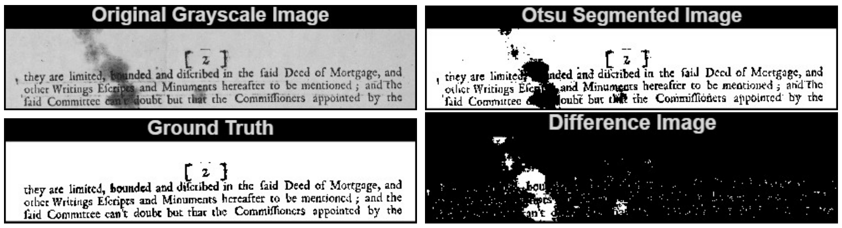

Figure 1: Outputs of code for `document01.bmp`.

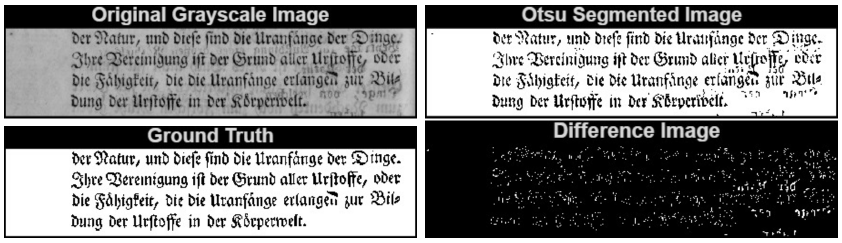

Figure 2: Outputs of code for `document02.bmp`.

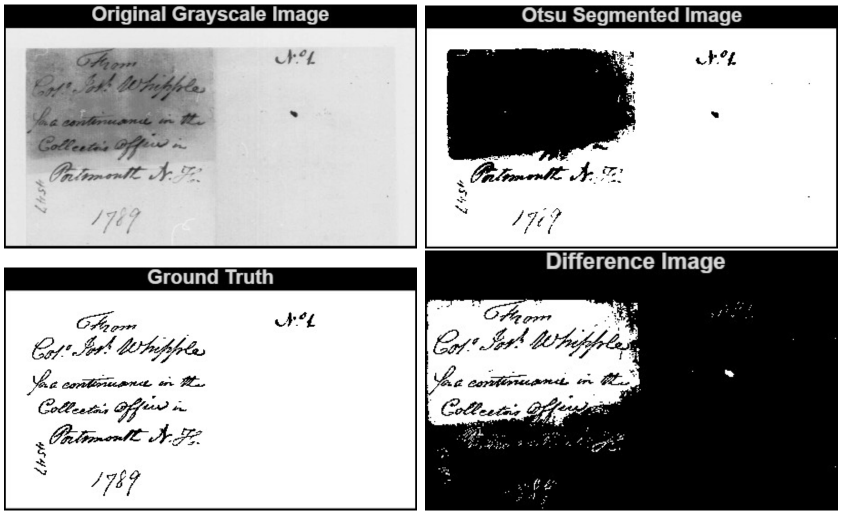

Figure 3: Outputs of code for `document03.bmp`.

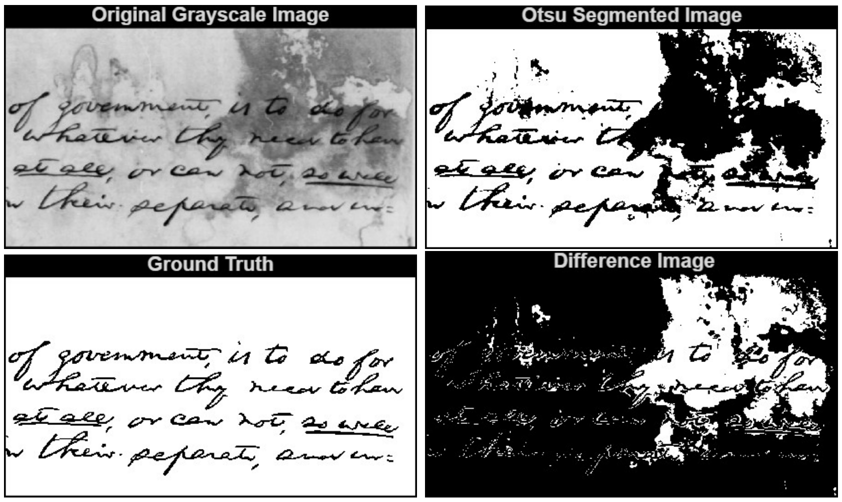

Figure 4: Outputs of code for `document04.bmp`.

Analysis and Discussion

Table 1 summarizes the threshold values and quantitative difference scores obtained for the four test images. The difference score represents the number of mismatched pixels between the segmented image and the ground truth, where lower values indicate more accurate segmentation.

Table 1: Otsu Global Thresholding Results for All Test Images

Image	Threshold (T)	Difference Score (pixels)
`document01.bmp`	0.545	27,849
`document02.bmp`	0.439	9,476
`document03.bmp`	0.690	179,165
`document04.bmp`	0.596	134,548

From the results, several observations can be made:

- **document02.bmp** achieved the lowest difference score (9,476 pixels) with a threshold of 0.439, indicating that Otsu's global method performed very well on this image. The lighting and contrast in this case were relatively uniform, allowing for accurate separation of text and background.

- **document01.bmp** produced a moderate score (27,849 pixels), which still represents acceptable performance. Minor background noise and slight illumination differences may have affected some text regions.

- **document03.bmp** and **document04.bmp** yielded much higher difference scores (179,165 and 134,548 pixels, respectively). These images likely contained strong degradation, faded text, or uneven brightness, which reduced the global thresholding accuracy. The single global threshold chosen by Otsu's method was insufficient to handle such local intensity variations.

- Overall, Otsu's method works well for clean, evenly lit document images but struggles when the image contains shadows, stains, or variable background illumination.

2.1.2 Niblack's Local Thresholding

In this part of the lab, Niblack's local thresholding method was applied for text segmentation. Unlike Otsu's global approach, which uses a single threshold for the entire image, Niblack's method computes a local threshold for each pixel based on the mean and standard deviation within a surrounding window. This allows the algorithm to adapt to varying lighting conditions and background noise, producing more accurate segmentation results for degraded document images. The parameter k was adjusted to study its influence on segmentation quality.

The Code Implamantation

```
Ipartb = imread('document01.bmp');
GTpartb = imread('document01-GT.tiff');

if size(Ipartb,3) == 3
    Igray = rgb2gray(Ipartb);
else
    Igray = Ipartb;
end
Igray = double(Igray);

% Parameter ranges
k_values = -1 : 0.2 : 1.5;
window_values = [15 25 55 75 105 125 155 175 205 255 275 305];

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

fprintf('Testing Niblack parameters...\n');

% Grid search for best k and window
for wi = 1:length(window_values)
    w = window_values(wi);
    for ki = 1:length(k_values)
        k = k_values(ki);

        % Local mean and standard deviation
        mean_local = conv2(Igray, ones(w)/(w^2), 'same');
        std_local = stdfilt(Igray, ones(w));

        % Niblack thresholding
        T = mean_local + k * std_local;
        BW = Igray > T;

        % Morphological cleanup
        BW = bwareaopen(BW, 20);
        BW = imclose(BW, strel('disk', 1));

        % Difference score
        diff_img = abs(double(BW) - double(GTpartb));
        score = sum(diff_img(:));

        scores(wi, ki) = score;
        fprintf('window = %d, k = %.2f -> diff score = %d\n', w, k, score);

        if score < best_score
            best_score = score;
            best_k = k;
            best_window = w;
            best_BW = BW;
            best_diff = diff_img;
        end
    end
end

fprintf('\n best combination  k = %.2f, window = %d, difference = %d\n', ...
    best_k, best_window, best_score);

figure('Name','Optimized Niblack Thresholding','NumberTitle','off');
subplot(2,2,1); imshow(uint8(Igray));
title('Original Image');
subplot(2,2,2); imshow(best_BW);
title(sprintf('Best Result (k=%.2f, w=%d)', best_k, best_window));
subplot(2,2,3); imshow(GTpartb);
title('Ground Truth');
subplot(2,2,4); imshow(best_diff, []);
title('Difference Image');

% Plot k vs difference for best window
[~, best_w_index] = min(min(scores,[],2));
figure('Name','Performance - k vs Difference','NumberTitle','off');
plot(k_values, scores(best_w_index,:), '-o', 'LineWidth', 2);
xlabel('k value'); ylabel('Difference Score');
title(sprintf('Effect of k (window = %d)', window_values(best_w_index)));
grid on;

% Plot window vs difference for best k
[~, best_k_index] = min(min(scores,[],1));
figure('Name','Performance - Window vs Difference','NumberTitle','off');
plot(window_values, scores(:, best_k_index), '-o', 'LineWidth', 2);
xlabel('Window Size'); ylabel('Difference Score');
title(sprintf('Effect of Window Size (k = %.2f)', k_values(best_k_index)));
grid on;
```
Listing 2: Niblack's Local Thresholding for Text Segmentation

Experimental Explanation

In this part of the lab, Niblack's local thresholding method was implemented to perform text segmentation on degraded document images. Unlike global thresholding, this approach calculates a threshold value for each pixel based on the local mean and standard deviation within a surrounding window. This allows the method to adapt to uneven illumination and noise in the image.

The MATLAB code explores different parameter combinations of the window size and the constant k through a grid search. For each pair of (w, k) values, the local mean is computed using the `conv2()` function, and the local standard deviation is obtained with `stdfilt()`. The threshold for each pixel is then defined as $T = \text{mean} + k \times \text{std}$, and pixels greater than T are classified as text.

After thresholding, morphological operations such as `bwareaopen()` and `imclose()` are applied to remove small artifacts and refine the segmented regions. Each resulting binary image is quantitatively evaluated by comparing it with the ground truth mask, using the sum of absolute pixel differences as an accuracy measure. The parameter combination yielding the lowest difference score is selected as the optimal one.

This process allows for the identification of the most effective (k, w) parameters that produce the best segmentation results, which are illustrated in the output figures below.

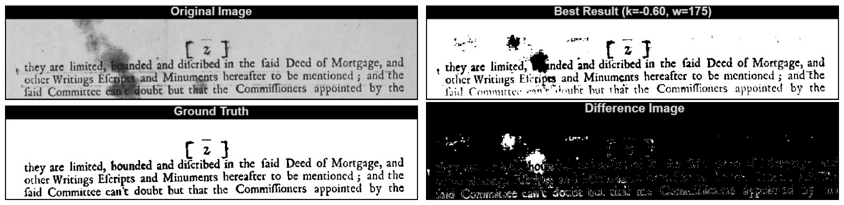

Figure 5: Outputs of code for `document01.bmp`.

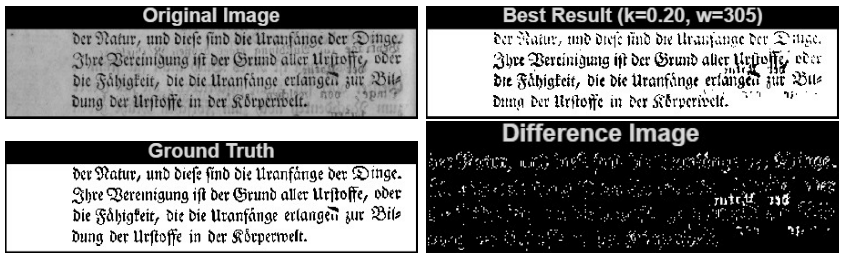

Figure 6: Outputs of code for `document02.bmp`.

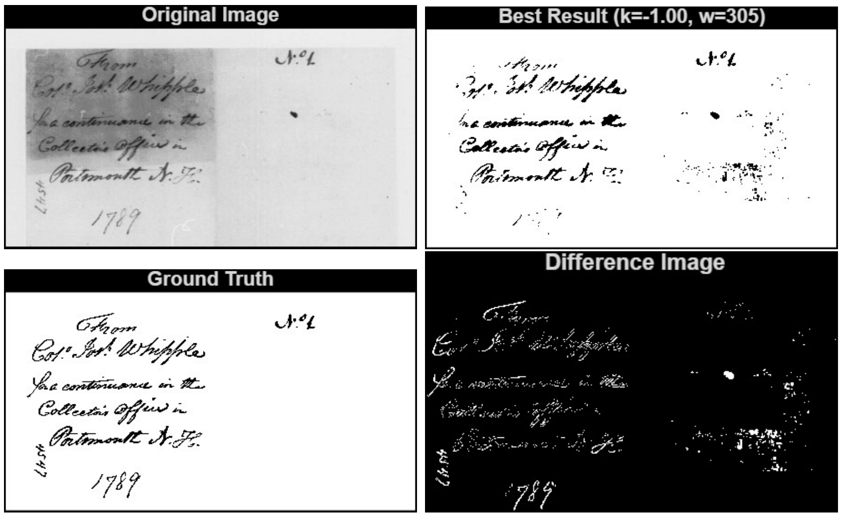

Figure 7: Outputs of code for `document03.bmp`.

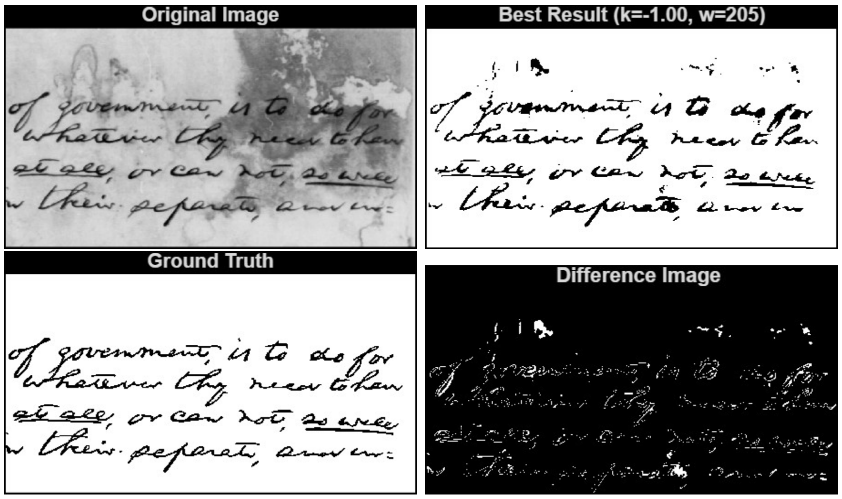

Figure 8: Outputs of code for `document04.bmp`.

Analysis and Discussion

The parameter optimization process for Niblack's local thresholding method was conducted to identify the most effective combination of the constant k and the window size w. The MATLAB implementation performed a grid search over multiple parameter values, evaluating each result by computing the sum of pixel-wise differences between the segmented image and the ground truth. The combination that produced the lowest difference score was selected as the optimal setting.

Figure 9 shows the effect of different k values on segmentation accuracy when the window size was fixed at $w = 175$. The results demonstrate that negative k values generally yield lower difference scores, indicating improved segmentation. This occurs because a smaller (negative) k reduces the threshold, helping preserve weak or faded text regions that would otherwise be lost.

Similarly, Figure 10 illustrates how the window size influences the difference score when $k = -0.60$. The results show a decreasing trend in the difference score as the window size increases, reaching its minimum around $w = 175$. Beyond this point, performance slightly degrades, suggesting that excessively large windows smooth out local variations and reduce segmentation precision.

From the evaluation, the best-performing parameters were found to be $k = -0.60$ and $w = 175$, producing the lowest overall difference score. These values provide a balance between noise reduction and text preservation. The final binary output generated with these optimal parameters achieved the most accurate separation of text and background among all tested configurations, as shown in the subsequent result images.

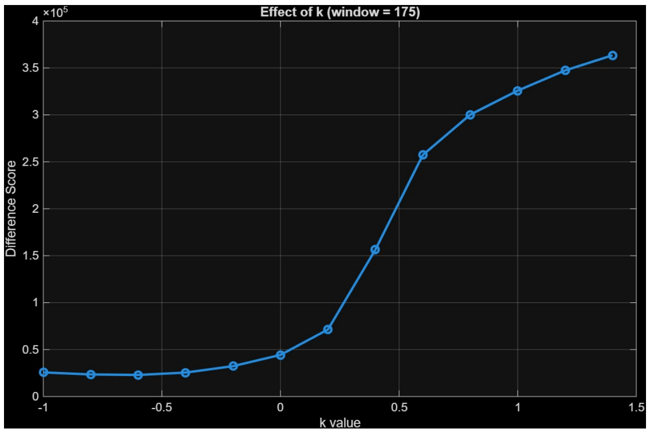

Figure 9: Effect of k on difference score (fixed window size $w = 175$).

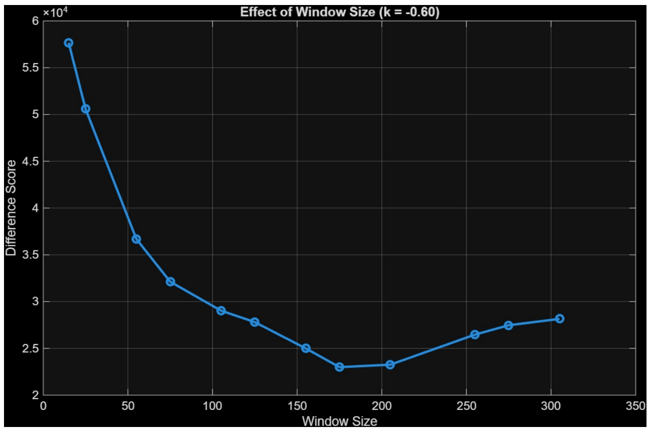

Figure 10: Effect of window size on difference score (fixed $k = -0.60$).

2.1.3 Improving Niblack's Local Thresholding Algorithm

Experimental Explanation

In this part of the experiment, additional improvements were applied to enhance the text segmentation results obtained from Niblack's local thresholding algorithm. The procedure was divided into two stages. In the first stage, the same optimal parameters ($k = -0.6$, $w = 175$) obtained in Part 3.1(b) were used, followed by simple morphological post-processing operations. The goal was to reduce background noise and fill small gaps in the character regions. This approach slightly improved the segmentation quality, as the difference score decreased marginally, but the overall visual enhancement remained limited. The improvement was therefore considered minor.

Because the first stage was not that succesfull , in the second stage, a more advanced adaptive thresholding technique—Sauvola's method—was implemented to address the limitations observed in the previous attempt. Sauvola's algorithm extends Niblack's formulation by introducing a normalization constant R and a smaller, positive control parameter k, which stabilizes the thresholding process under varying illumination and contrast conditions. The local threshold is defined as

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

where $m(x,y)$ and $s(x,y)$ represent the local mean and standard deviation, respectively. This allows the algorithm to better preserve faint text regions while suppressing background noise.

A grid search was carried out to evaluate various combinations of k and window sizes, selecting the pair that produced the lowest pixel-wise difference between the segmented output and the ground truth. As observed in the figure below, the morphological refinement offered limited improvement, while the

Sauvola method provided more stable and balanced segmentation results, particularly in degraded document regions.

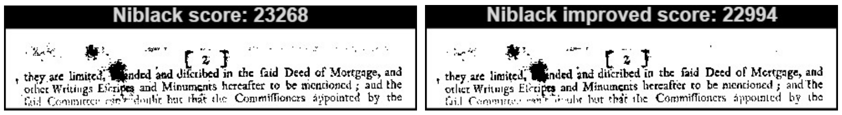

Figure 11: Comparison between basic Niblack segmentation and morphological improvement (difference scores show only minor gains).

As it shown in the figures, the improvment with this method did not really clean the images, it could be seen that the black dots in the upper parts of the image that is output of just Niblack have been removed. However there is still no big gain.

The Code Implemantation

```matlab

% Load image and ground truth
Ipartc = imread('document01.bmp');
GTpartc = imread('document01-GT.tiff');

if size(Ipartc,3) == 3
    Igray = rgb2gray(Ipartc);
else
    Igray = Ipartc;
end
Igray = double(Igray);

% Use the best parameters from previous section
best_k = -0.6;
best_window = 175;
fprintf('Using the best parameters from section 3.1(b): k = %.2f, window = %d\n\n', best_k, best_window);

% --- NIBLACK IMPROVEMENT (Morphological Refinement) ---

% Apply Niblack algorithm with best parameters
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;

% score calculation
diff_img_before = abs(double(BW_no_improvement) - double(GTpartc));
score_before = sum(diff_img_before(:));
fprintf('Difference score BEFORE improvement = %d pixels\n', score_before);

% apply morphologic method
BW_improved = bwareaopen(BW_no_improvement, 20);      % Remove small noise
BW_improved = imclose(BW_improved, strel('disk', 1)); % Fill small gaps

% Compute difference score after improvement
```

```
diff_img_after = abs(double(BW_improved) - double(GTpartc));
score_after = sum(diff_img_after(:));
fprintf('Difference score AFTER improvement = %d pixels\n\n', score_after);

% Display comparative results
figure('Name', '3.1(c) Niblack Improvement Analysis', 'NumberTitle', 'off');
subplot(2,2,1); imshow(uint8(Igray)); title('Original');
subplot(2,2,2); imshow(GTpartc); title('Ground Truth');
subplot(2,2,3); imshow(BW_no_improvement);
title(sprintf('Niblack score: %d', score_before));
subplot(2,2,4); imshow(BW_improved);
title(sprintf('Niblack improved score: %d', score_after));


% =========================================================
% --- SAUVOLA ALGORITHM (Improved Local Thresholding) ---
% =========================================================

imageNames = {'document01'};
fprintf('=== 3.1(c) Sauvola (Improvement) Full Analysis Started ===\n');

% Parameter definitions
k_values = 0.05 : 0.05 : 0.5;   % Sauvola's k (positive and small)
window_values = 15:30:300;
R_const = 128;

for idx = 1:length(imageNames)

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

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

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

    % grid search
    fprintf('Parameter scan in progress...\n');
    for wi = 1:length(window_values)
        w = window_values(wi);
        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);
            T = mean_local .* (1 + k * ((std_local / R_const) - 1));

            % Threshold
            BW = Igray > T;
            BW_clean = bwareaopen(BW, 10);

            % Calculate difference score
            diff_img = abs(double(BW_clean) - double(GT));
            score = sum(diff_img(:));
            scores(wi, ki) = score;

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

    % --- Display Visual Results ---
    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');

    %  k vs Score (for best window)
    [~, best_w_index] = min(min(scores,[],2));
    subplot(1,3,1);
    plot(k_values, scores(best_w_index,:), '-o', 'LineWidth', 2);
    xlabel('k value'); ylabel('Difference Score');
    title(sprintf('k vs Score (window=%d)', window_values(best_w_index)));
    grid on;

    %  Window vs Score (for best k)
    [~, 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('Difference Score');
    title(sprintf('Window vs Score (k=%.2f)', k_values(best_k_index)));
    grid on;

end
```

Listing 3: Comparison of Niblack improvement and Sauvola adaptive thresholding.

Experimental Explanation

In this part, the Niblack method was first refined using morphological operations to remove small noise and fill text gaps. Although a slight improvement was achieved, the overall enhancement was limited. Therefore, Sauvola's adaptive thresholding was implemented as a more robust alternative. It adjusts the local threshold using both the local mean and standard deviation, producing more stable segmentation under uneven illumination. As shown in the output images, Sauvola achieved clearer text boundaries and fewer background artifacts, confirming its superior performance in this experiment.

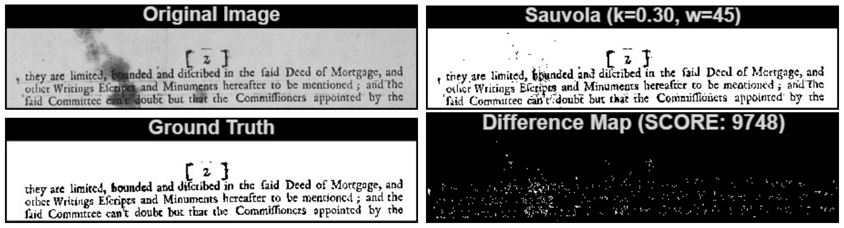

Figure 12: Outputs of code for `document01.bmp`.

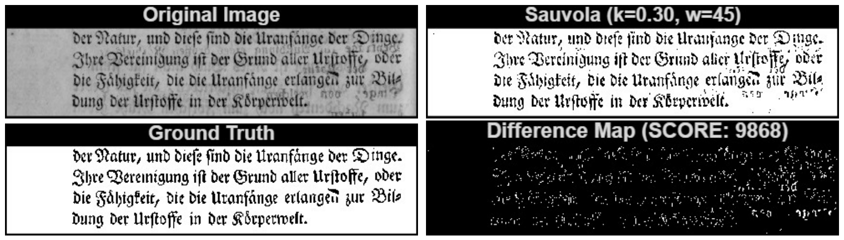

Figure 13: Outputs of code for `document02.bmp`.

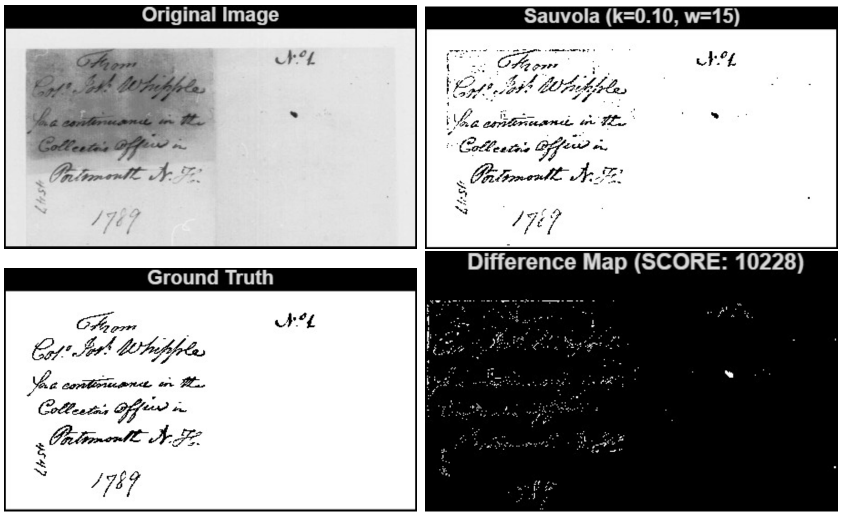

Figure 14: Outputs of code for `document03.bmp`.

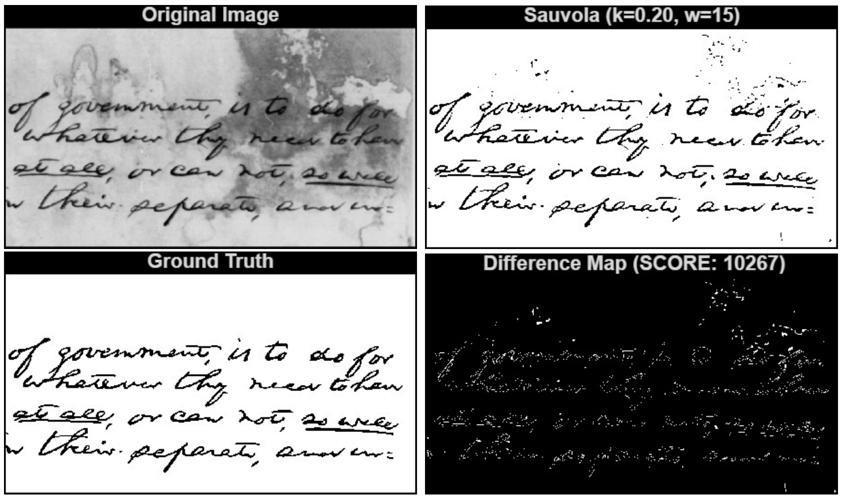

Figure 15: Outputs of code for `document04.bmp`.

Analysis and Discussion

The initial morphological refinement applied to the Niblack method resulted in only minor improvements, with difference scores remaining relatively high. This indicated that the approach was insufficient for handling background noise and uneven illumination in the document images. In contrast, the Sauvola algorithm achieved significantly lower difference scores, ranging from 9748 to 10267 pixels across the four test images. Optimal parameters were found between

$k = 0.10$–0.30 and window sizes of 15–45, showing that smaller windows performed better on clean backgrounds, while larger ones were more effective on degraded areas. These results confirm that Sauvola's adaptive thresholding provides more stable and accurate segmentation, reducing artifacts and preserving text boundaries more effectively than the improved Niblack approach.

2.2 3D Stereo Vision

2.2.1 General Explanation

In this section, a stereo vision algorithm is employed to generate disparity maps, which encode the relative depth of objects within a scene. The approach utilizes the Sum-of-Squared-Differences (SSD) rule. This method measures the similarity between a template patch T and a corresponding region in the image I using the following equation:

$$S(x, y) = \sum_{j=0}^{M} \sum_{k=0}^{N} (I(x + j, y + k) - T(j, k))^2 \tag{1}$$

Equation 1 calculates the intensity difference between the left and right image patches for each potential horizontal shift. The coordinates (x, y) that yield the minimum $S(x, y)$ value indicate the best match, and the corresponding horizontal offset defines the disparity.

Computation of this metric is computationally expensive. Therefore, it is implemented more efficiently using convolution-based operations. This allows the SSD components to be calculated in an optimized, separable manner.

The algorithm processes an 11×11 neighbourhood window for each pixel in the left image. It then searches for the matching region along the same scanline in the right image, constraining the search to disparities below 15 pixels. The disparity for each pixel is assigned based on the shift that produces the minimum SSD value.

As shown in the figures (not shown here), the synthetic *corridor* dataset produces smooth disparity gradients and well-defined planar surfaces. In contrast, the real *triclops* images exhibit higher noise and more irregular depth estimates. These irregularities are likely due to lighting variations and inconsistent textures in the real-world images.

These findings confirm that while SSD-based matching can recover depth from stereo pairs, its performance is dependent on factors such as texture richness and chosen window size.

The Code Implemantation

```

% Lab 2 - 3.2 3D Stereo Vision
% 3.2 (b)
left_tri = rgb2gray(imread('triclopsi2l.jpg'));
right_tri = rgb2gray(imread('triclopsi2r.jpg'));
```

```matlab
left_corr = rgb2gray(imread('corridorl.jpg'));
right_corr = rgb2gray(imread('corridorr.jpg'));


% 3.2 (c):
D1 = DispMap(left_corr, right_corr, 11, 11);
figure('Name', '3.2(c)');
imshow(D1, [-15 15]);       %when I use -D, its conflicting with wanted values, so I need to
    ↪ use "D" instead of "-D"
title('Corridor Disparity');

colormap gray;
colorbar;


% 3.2 (d)
D2 = DispMap(left_tri, right_tri, 11, 11);
figure('Name', '3.2(d) ');
imshow(D2, [-15 15]);       %when I use -D, its conflicting with wanted values, so I need to
    ↪ use "D" instead of "-D"
title('Triclops Disparity');

colormap gray;
colorbar;

% 3.2 (a)
function D =DispMap(PL,PR, templateH, templateW)

    [height, width] = size(PL);

    disp_map = zeros(height, width);
    min_ssd = inf(height, width);

    PL_double = double(PL);
    PR_double = double(PR);

    template_kernel = ones(templateH, templateW);

    for d = 0:14  %since max disparity should be <15
        PR_shifted = [zeros(height, d), PR_double(:, 1:width-d)];
        squared_diff = (PL_double - PR_shifted).^2;
        ssd_map_at_d = conv2(squared_diff, template_kernel, 'same');

        update_mask = ssd_map_at_d < min_ssd;
        min_ssd(update_mask) = ssd_map_at_d(update_mask);
        disp_map(update_mask) = d;
    end

    D =disp_map;
end
```

Listing 4: Estimating Disparity Maps

2.2.2 Experimental Explanation

In this experiment, the implementation aims to generate disparity maps for two stereo image pairs using a block-based approach. The provided code applies the developed disparity function to both a synthetic and a real-world dataset, allowing a direct comparison of the accuracy and robustness of the results under different image conditions.

First, the left and right stereo images are read and converted to grayscale to simplify computation, as color information is not required for disparity estimation. Two image sets are used: the *corridor* pair, representing a synthetic and structured environment, and the *triclops* pair, representing real-world imagery with more irregular features.

The main computation is performed through the custom function `DispMap(PL, PR, templateH, templateW)`. This function estimates pixel-wise disparities by comparing small local regions between the left and right images. The template dimensions are set to 11×11, defining the local search window around each pixel. The algorithm evaluates multiple disparity levels (from 0 to 14 pixels) and shifts the right image horizontally for each disparity value.

For every shift, the function computes the squared intensity differences between corresponding pixel patches and integrates them using a convolution operation with a uniform kernel. This operation effectively sums the local squared differences, representing the matching cost for that disparity. The disparity value with the lowest sum of squared differences (SSD) is then recorded for each pixel, producing a dense disparity map.

Two disparity maps are generated: one for the *corridor* pair and one for the *triclops* pair. Each resulting map is visualized using `imshow` within the disparity range of $[-15, 15]$, with grayscale colormap and colorbar to interpret depth visually. Brighter regions correspond to closer surfaces, while darker regions indicate greater depth.

During visualization, the negative disparity was intentionally avoided (i.e., `-D` replaced by `D`) to maintain consistency with the expected direction of the computed disparities. This ensures that the resulting map aligns correctly with the left-to-right disparity convention used in the experiment.

Overall, this experimental process demonstrates the practical computation of disparity using SSD matching within a fixed disparity range. The results highlight how disparity estimation performance varies with the texture and structure of the images: synthetic scenes such as the corridor yield smoother and more stable disparity maps, while real images like the triclops dataset present more noise and less distinct depth transitions due to illumination differences and surface texture complexity.

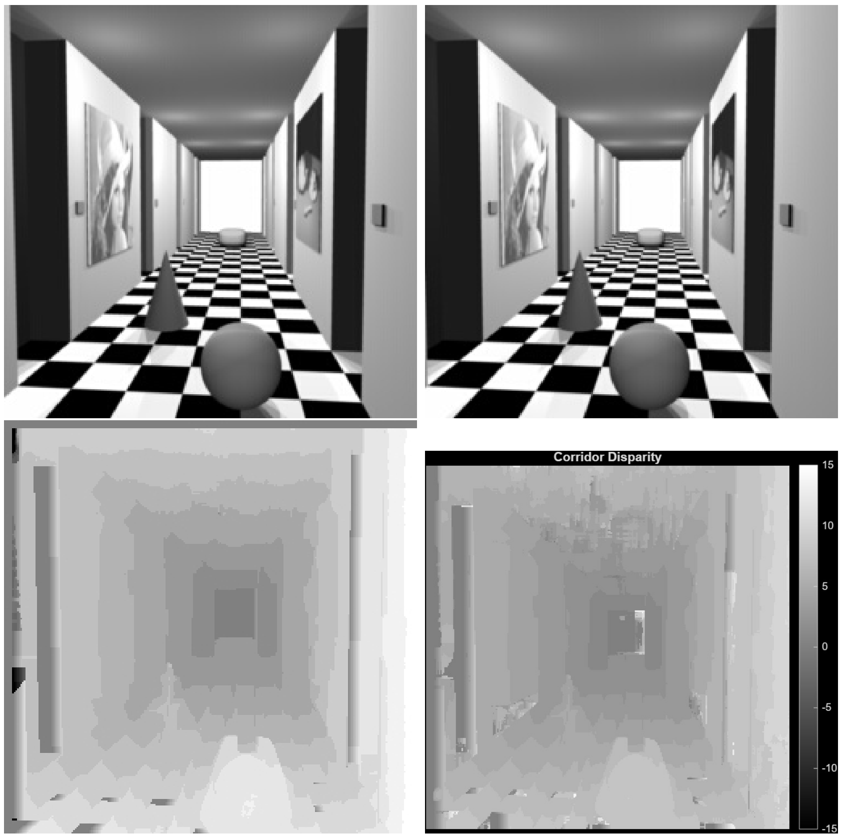

Figure 16: Outputs of code for `The Corridor` .

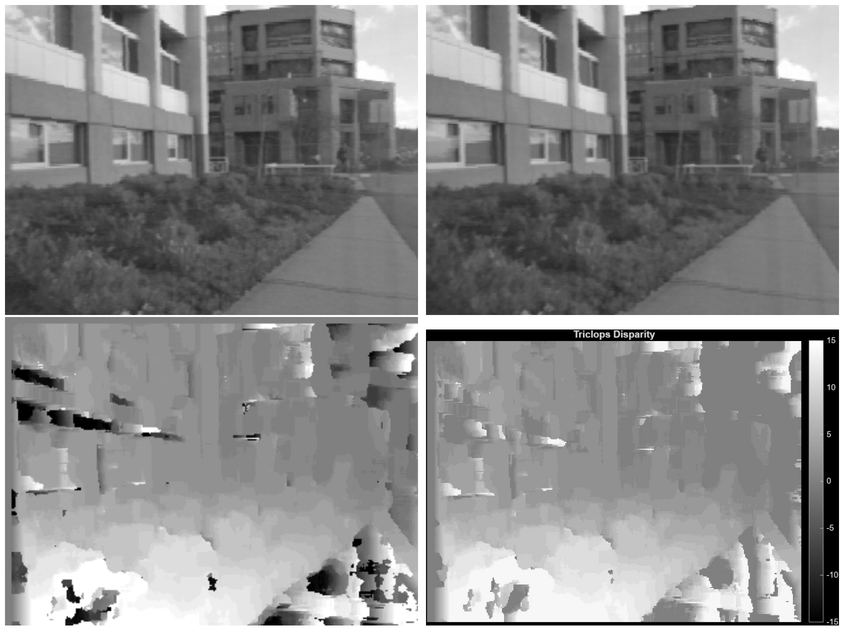

Figure 17: Outputs of code for `The Triclops`.

Visual Explanation

The top two images of both figures, shown above represent the original left and right stereo inputs used for the disparity estimation process. The bottom-left image corresponds to the reference or expected disparity map provided in the laboratory materials, while the bottom-right image shows the disparity map generated by the implemented algorithm. Although the produced result is not same with reference(given disparity), it exhibits strong structural similarities in terms of depth gradients and object boundaries. As described in the experimental explanation, this output was obtained by applying SSD-based block matching using a fixed window size and a constrained disparity range. The result confirms that the implemented method successfully reconstructs the general depth structure of the scene, with only minor differences caused by texture differences and code differences from optimal code and my code.

2.2.3 Analysis and Discussion

The implemented stereo vision algorithm effectively demonstrates how disparity maps can be generated using a simple SSD-based block matching method. By comparing pixel intensities between the left and right stereo images, the system produces reasonable depth estimations for both synthetic and real scenes.

For the *corridor* dataset, the disparity map closely matches the reference image, showing clear depth gradients along the walls and floor. The structured geometry and consistent lighting make matching straightforward, resulting in accurate and smooth disparity transitions.

The *triclops* dataset, however, shows less consistent results. Although the overall depth structure remains similar to the reference, the darker regions representing larger disparities are less visible. This is mainly due to real-world factors such as uneven lighting, reflections, and lower texture contrast, which make precise correspondence matching more difficult.

Overall, the SSD-based approach performs well for scenes with strong textures and controlled lighting but is less reliable for complex or poorly textured environments. Despite these limitations, the experiment successfully illustrates the core principle of disparity estimation and the influence of image characteristics on stereo vision performance.

3 References

1. N. Otsu, “A Threshold Selection Method from Gray-Level Histograms,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–74, 1979. doi:10.1109/TSMC.1979.4310076
2. W. Niblack, *An Introduction to Digital Image Processing.* Englewood Cliffs, NJ: Prentice Hall, 1986.
3. A. Garg, “Stereo Vision: Depth Estimation Between Object and Camera,” *Analytics Vidhya*, 2024. Accessed: Nov. 07, 2024. [Online]. Available: https://medium.com/analytics-vidhya/distance-estimation-cf2f2fd709d8