DIP Assignment-4 Report

(Kedarnath P - 20902)

(1) Image Denoising

Implementation Details:

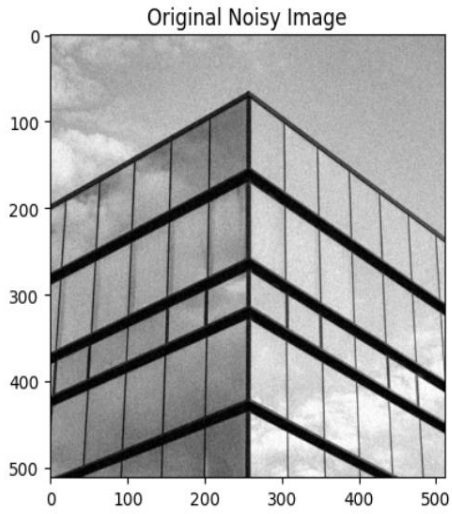
- The `bilateral_filter` function applies a bilateral filter to the input image to reduce noise while preserving edges. It does this by considering both the spatial distance and the intensity difference when computing the weights for blurring.
- The `gaussian_filtered` function applies a Gaussian filter to the input image, which smooths the image by giving more weight to the pixels closest to the center of the kernel and less to those further away.
- The `apply_laplacian` function applies a Laplacian filter to the input image to detect edges.
- The Implementation reads a noisy grayscale image and applies the bilateral filter and Gaussian filter using the defined functions and displays the original noisy image, the bilateral filtered image, and the Gaussian smoothed image for comparison.
- After filtering, the Laplacian filter is applied to all three above images using the `apply_laplacian` function. These results are also displayed to show the effect of the Laplacian filter on each image.

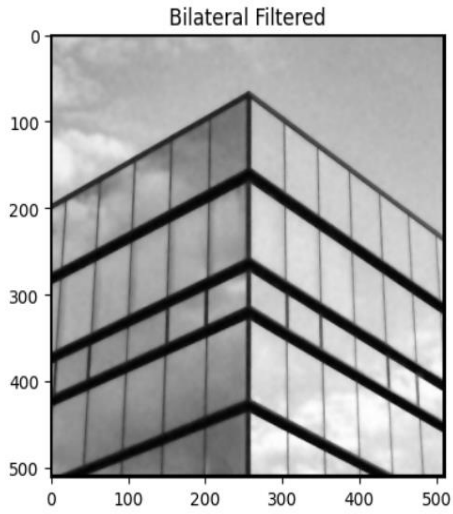
Observations and Analysis:

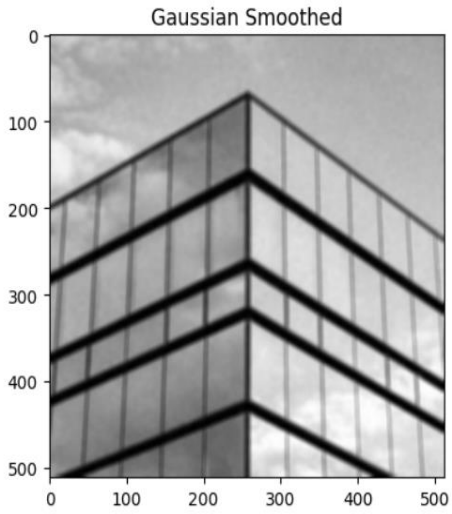
- **Original Noisy Image**: The image is affected by Gaussian noise, which is apparent as random variations in the Image's intensity.
- **Bilateral Filtered**: The bilateral filter effectively reduces noise while keeping the edges sharp. This is evident in the filtered image where the building's edges are well-preserved compared to the original noisy image.
- **Gaussian Smoothed**: The Gaussian filter smooths the image, reducing noise but also blurring the edges slightly. This is seen in the softer appearance of the building's edges.
- **Laplacian of Noisy Image**: The Laplacian filter is an edge-detection filter that highlights rapid intensity changes, including noise. When it is applied to the noisy image amplifies and highlights the noise, making the image look grainier and the edges less distinct, indicating that the noise significantly affects the edge detection.
- **Laplacian of Bilateral Filtered**: The Laplacian filter on the bilateral filtered image shows clear and distinct edges with minimal noise, indicating good noise reduction while preserving edge information.
- **Laplacian of Gaussian Smoothed**: The Laplacian filter applied to the Gaussian smoothed image shows less pronounced edges due to the blurring effect of the Gaussian filter.
- **Original Noisy Image**: The image is affected by Gaussian noise, which is apparent as random variations in the Image's intensity.

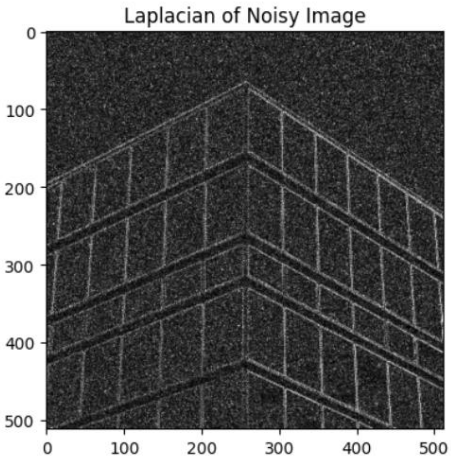
In summary, the bilateral filter is more effective for noise reduction in images with edges that need to be preserved. The Gaussian filter is a good general-purpose smoothing filter but does not preserve edges as well as the bilateral filter. The Laplacian filter is sensitive to noise and is best applied to images that have already been smoothed.

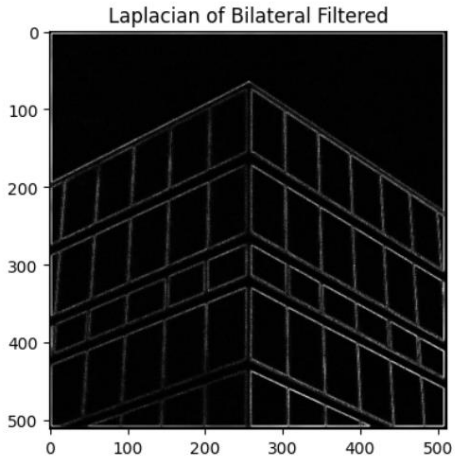
Visualization of the Results:

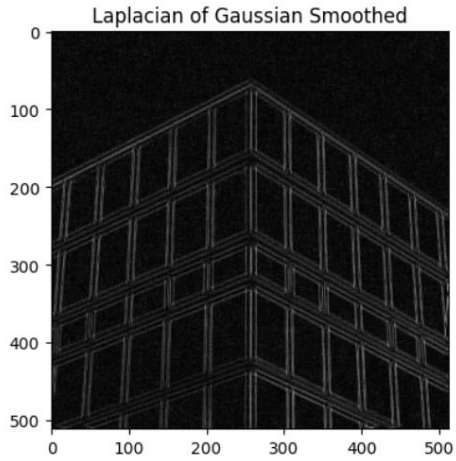












(2) Edge Detection

Implementation Details:

- **Image Loading**: Given Images are loaded and converted to grayscale using the PIL library.
- **Gaussian Filter**: A Gaussian filter function `gaussian_filter` is defined, which creates a Gaussian kernel based on input size and sigma values. It is applied to the images to smooth them.
- **Sobel Operator**: The Sobel operator `sobel_operator` is defined to calculate the gradient magnitudes of the images, which are used to detect edges.
- **Edge Detection on Original Noisy vs. Smoothed Images**: Edge detection is performed on both the original noisy and the smoothed images, and the results are plotted side by side for comparison.
- **Effect of Gaussian Smoothing**: The effect of different levels of Gaussian smoothing (varying sigma values) on edge detection is visualized.
- **Thresholding**: A function `threshold_image` for applying thresholding to images is defined and used to threshold the edge-detected images with different threshold values.
- **Visualization**: The results of edge detection with different smoothing levels and thresholds are visualized in a series of 4x4 grids.

Observations:

- **Gaussian Smoothing**:
  - **Reduces Noise**: Smoothing the images with a Gaussian filter helps to reduce noise, which can lead to cleaner edge detection.
  - **Blurs Details**: Higher levels of smoothing (larger sigma values) can blur important details and edges in the images.
  - **Balance is Key**: There's a trade-off between noise reduction and edge preservation. Optimal sigma values should be chosen based on the specific characteristics of the images.
- **Gradient-Based Edge Detection (Sobel Operator)**:
  - **Effective for High Contrast Edges**: Works well for detecting edges where there's a significant change in intensity.
  - **Sensitivity to Noise**: Direct application on noisy images can result in detecting noise as edges.
- **Thresholding**:
  - **Binary Edge Maps**: Thresholding converts gradient magnitudes to binary edge maps, distinguishing between edge and non-edge pixels.
  - **Importance of Threshold Value**: The choice of threshold value significantly impacts the outcome. A lower threshold detects more edges, but it may also include noise and a higher threshold result in fewer, but more prominent, edges.

Analysis:

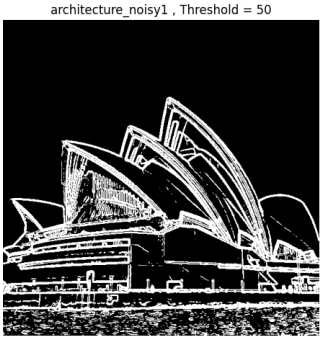
- Smoothing with a Gaussian filter before edge detection can significantly reduce noise, which helps in obtaining clearer and more distinct edges.
- Increasing the Gaussian filter's sigma value increases the smoothing effect, which can cause finer details to be lost, as edges become less distinct.
- The trade-off with higher sigma values is that while noise is further reduced, important structural details may also be blurred out.
- The threshold value in edge detection acts as a sensitivity control. Lower thresholds may include more details but also noise, while higher thresholds may result in missing finer details.

Visualization of the Results:

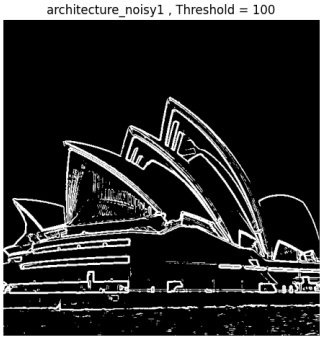
Effect of Smoothing : Sobel Edge Detection on Smoothed Images with varying sigma

Effect of Thresholding : Sobel Edge Detection on Smoothed Images with sigma = 1

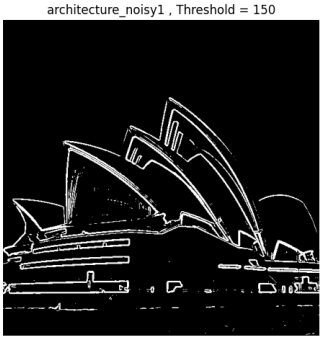
architecture_noisy1 , Threshold = 50



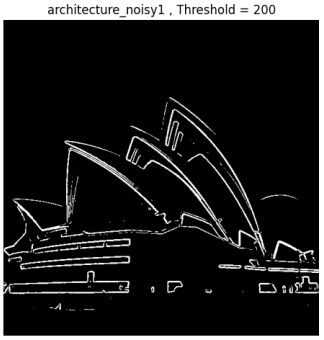
architecture_noisy1 , Threshold = 100



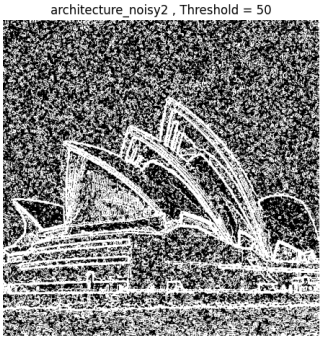
architecture_noisy1 , Threshold = 150



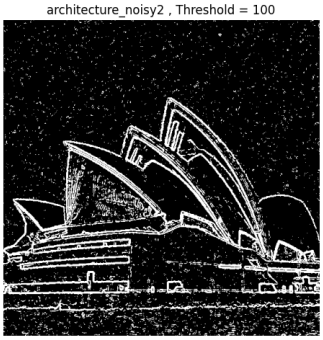
architecture_noisy1 , Threshold = 200



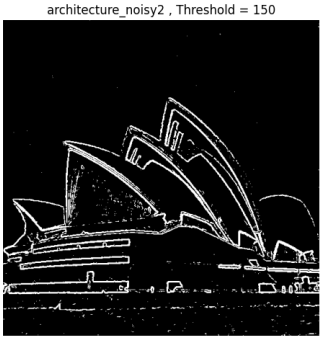
architecture_noisy2 , Threshold = 50



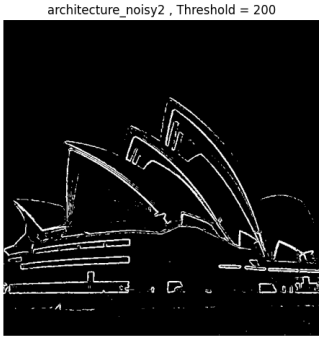
architecture_noisy2 , Threshold = 100



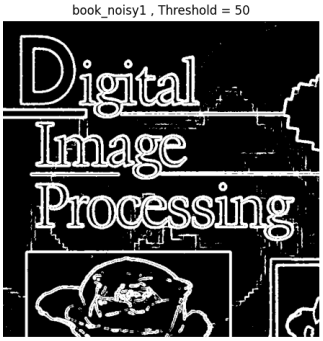
architecture_noisy2 , Threshold = 150



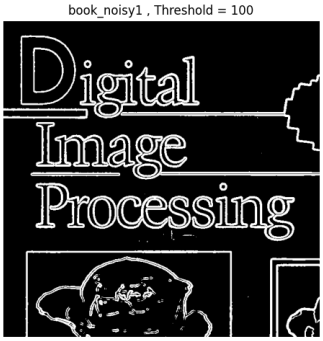
architecture_noisy2 , Threshold = 200



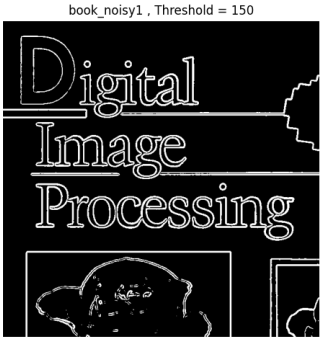
book_noisy1 , Threshold = 50



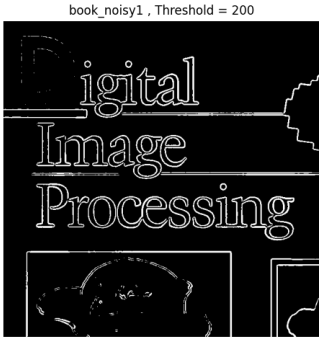
book_noisy1 , Threshold = 100



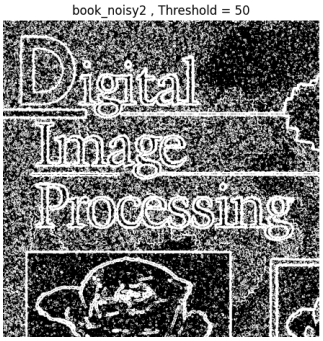
book_noisy1 , Threshold = 150



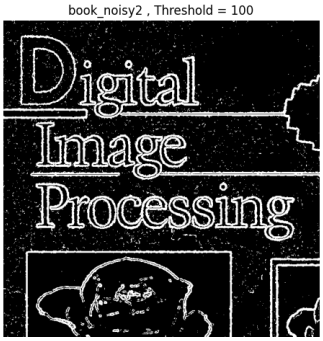
book_noisy1 , Threshold = 200



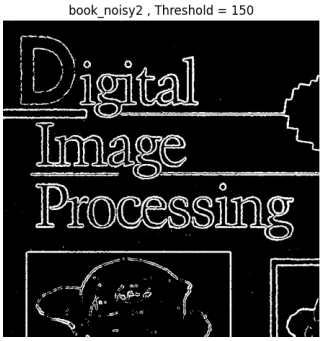
book_noisy2 , Threshold = 50



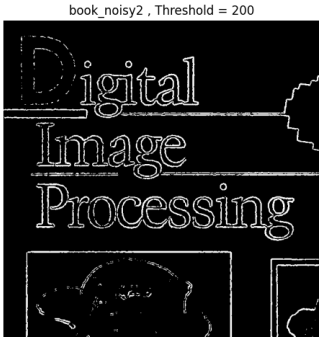
book_noisy2 , Threshold = 100



book_noisy2 , Threshold = 150



book_noisy2 , Threshold = 200



(3) Hough Transform:

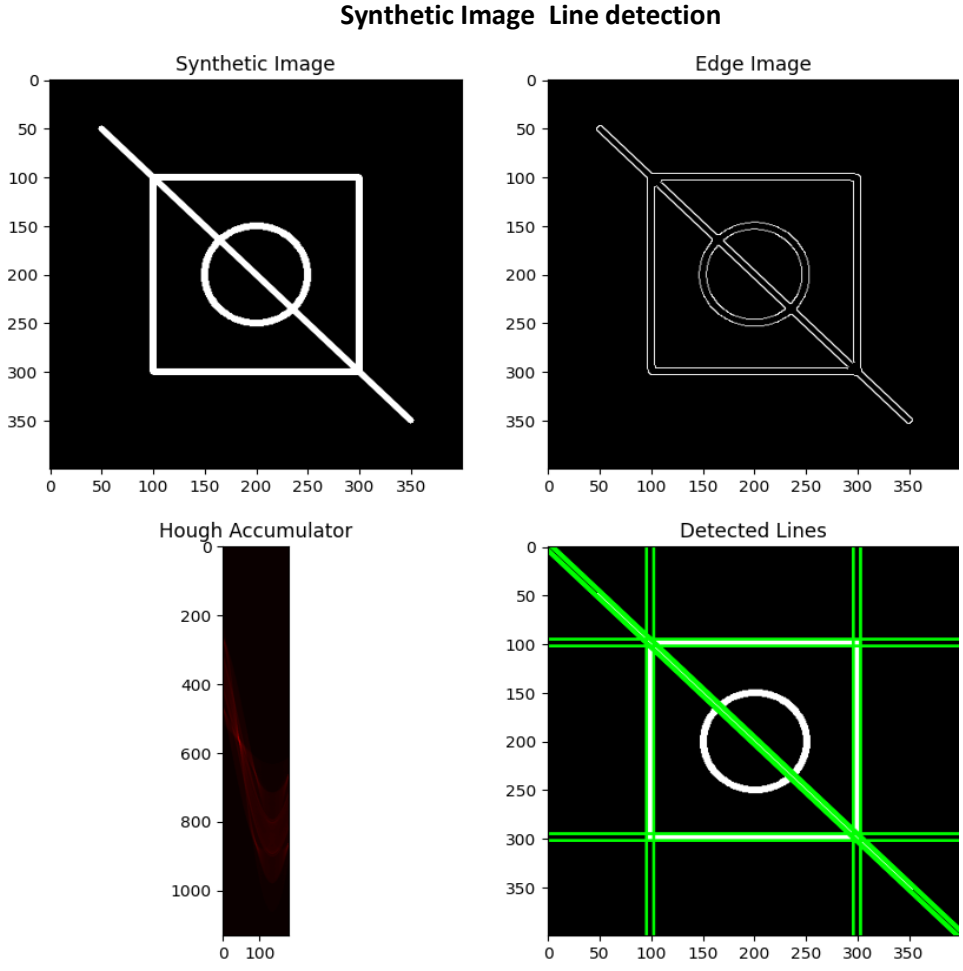
Implementation details:

- **Synthetic Image Creation**:
  - A black canvas of 400x400 pixels is created.
  - A white line, rectangle, and circle are drawn onto this canvas.
- **Edge Detection**:
  - The `edge_detection` function applies a Gaussian blur to the image and then uses the Canny edge detector to find edges.
- **Hough Line Transform**:
  - The `hough_line_transform` function computes the Hough Transform for the given edge-detected image.
  - Theta values are computed from -90 to 90 degrees, with the number of bins specified by `num_theta_bins`. (used num_theta_bins = 180)
  - Rho values are computed based on the image diagonal, with the number of bins specified by `num_rho_bins`. (used num_rho_bins = 1132 = 2*diagonal_length)
  - An accumulator array is created and filled based on the non-zero edge pixels, casting votes for potential lines.
- **Line Detection**:
  - The `detect_lines` function iterates over the accumulator array.
  - It selects bins where the number of votes exceeds the specified threshold, which indicates the presence of a line.
- **Line Drawing**:
  - Detected lines are drawn onto the original synthetic image using the `draw_lines_colored` function, which converts the image to color and draws the lines in green.
- **Noise and Occlusion**:
  - The `add_noise` function adds random noise to the synthetic image.
  - The `add_occlusions` function adds white square occlusions to random locations on the image.
- **Testing with Different Parameters**:
  - The Hough Transform is applied to the original, noisy, and occluded images using different numbers of theta and rho bins.
- **Results Visualization**:
  - Displayed the synthetic image, the edge-detected image, the accumulator array from the Hough Transform, and the image with detected lines drawn.
- **Real-World Images**:
  - The process is repeated for real-world images, with the grayscale conversion applied before edge detection.
  - Detected lines are drawn on the original real-world images and displayed.

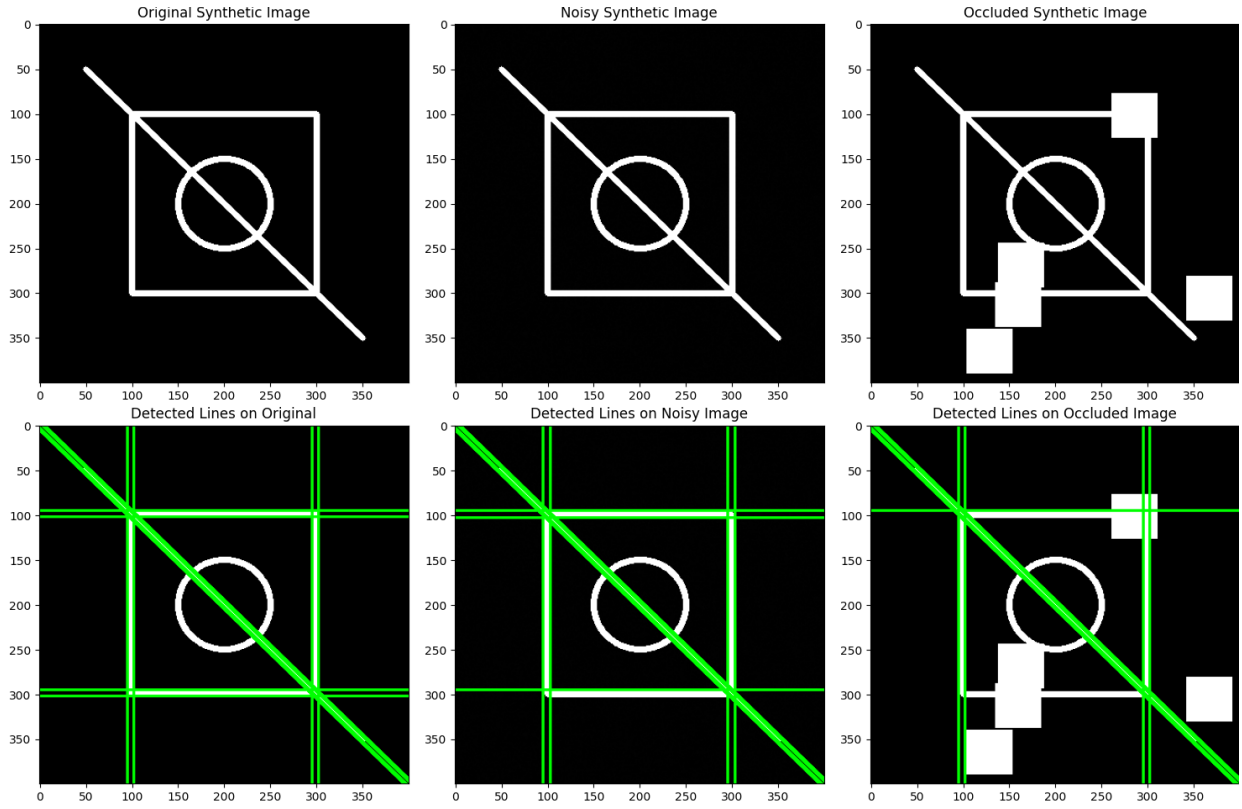
Analysis:

- **Shape Limitation**: The Hough Transform is specialized for straight line detection and does not detect curves or circular shapes.
- **Noise Tolerance**: The algorithm shows a degree of tolerance to noise, but its effectiveness is directly related to the noise level. Low-level noise can be managed, while high-level noise significantly affects performance.
- **Occlusion Handling**: Small occlusions do not prevent line detection, but larger ones can obscure lines enough to prevent their detection.
- **Parameter Sensitivity**: The sensitivity of rho and theta bin sizes in the Hough Transform is pivotal to line detection accuracy. More bins offer finer resolution, allowing for precise differentiation of closely spaced or nearly parallel lines, but increase computational load and may reduce overall detections. Conversely, fewer bins enhance processing speed at the cost of precision. The Threshold acts as a filter, with higher values ensuring only the most prominent lines are detected, thus reducing noise but possibly omitting subtler lines (weaker edges). Balancing these parameters is essential for optimal detection performance.
- **Real-World Applicability**: The Hough Transform's ability to detect road lanes in the real-world image suggests its suitability for computer vision tasks in real environments, provided that the images have clear, distinguishable lines.

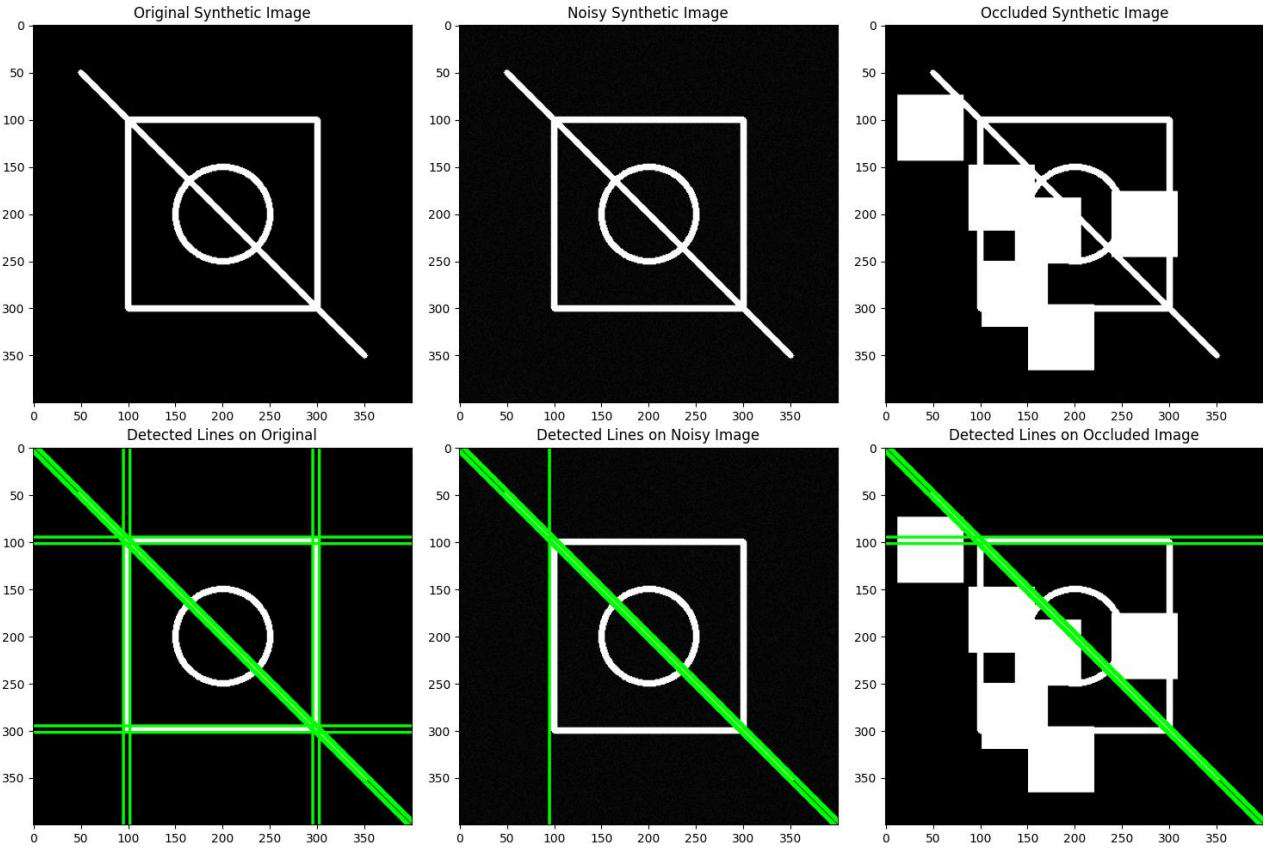
Visualization:



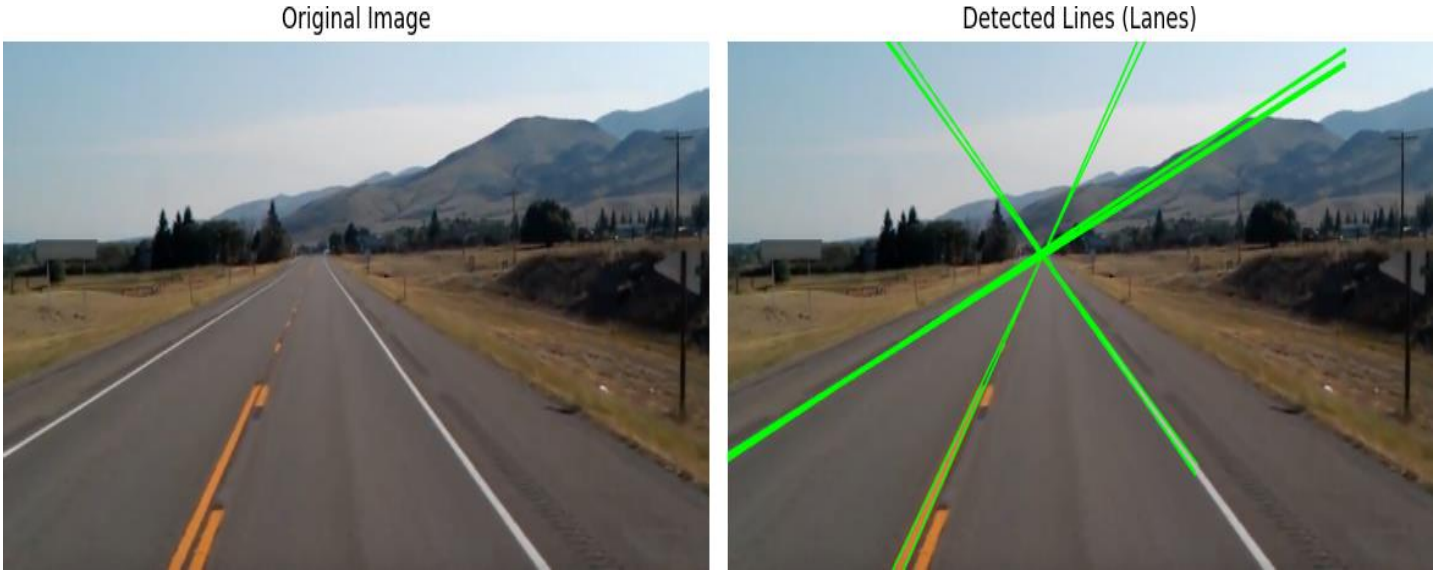
Low level of Noise and Occlusion



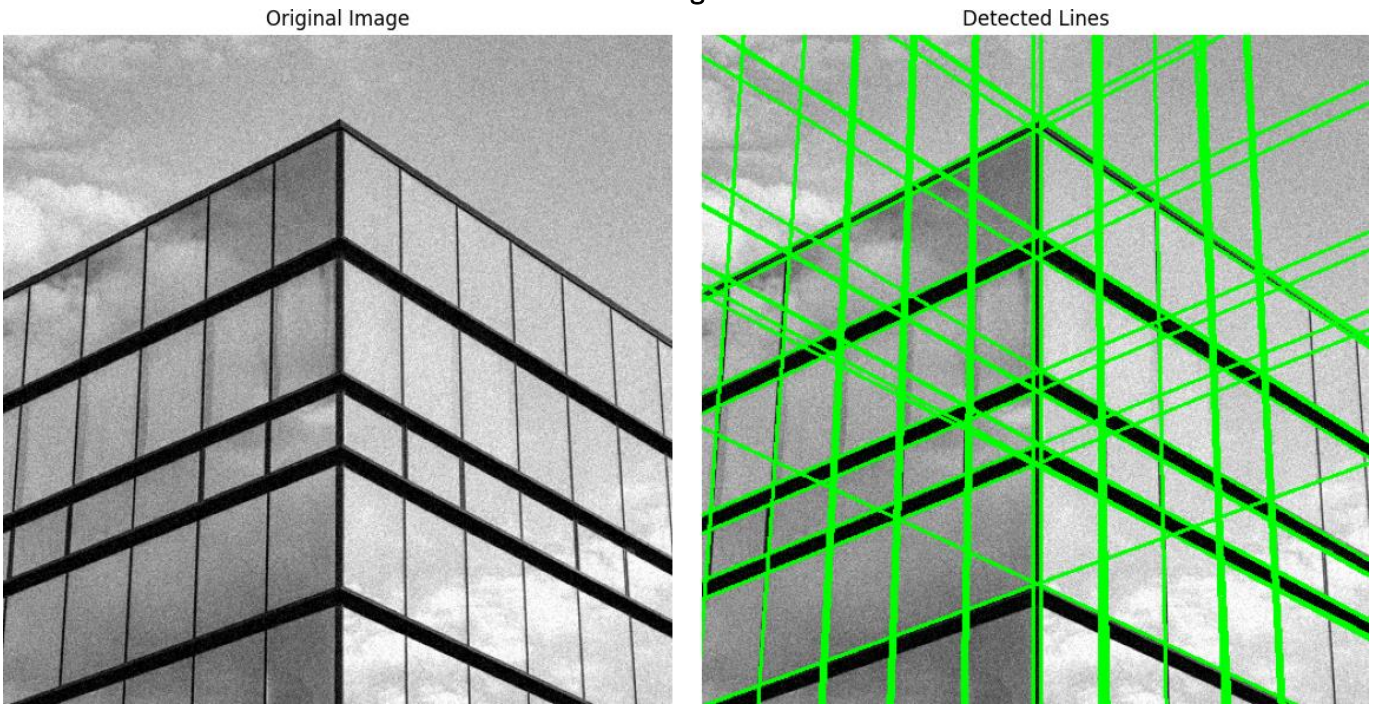
High level of Noise and Occlusion



Real Image Test –1



Real Image Test –2



.