Ioannis Petrakis

1115201900155

The primary objective of this project was to design and implement a comprehensive computer vision pipeline capable of performing document rectificatiov. This tool simulates the core functionality of document scanning applications, requiring a bottom-up approach to image processing.

The development process aligns directly with the fundamental modules of the Computer Vision course, progressing from low-level pixel manipulation to high-level geometric interpretation. Specifically, the project addresses four key areas:

1. **Image Fundamentals & Filtering:** Understanding image representation (floating-point vs. integer) and implementing noise reduction using linear filters from scratch.
2. **Edge Detection:** Identifying structural boundaries using the Canny algorithm to understand scene geometry.
3. **Feature Detection:** Implementing and comparing corner detection algorithms (Moravec and Harris) to locate robust keypoints within the image.
4. **Geometric Transformations:** Utilizing projective geometry (Homography) to map the detected keypoints to a target coordinate system, effectively rectifying the image.

By integrating these distinct modules, the final tool demonstrates how individual processing steps—such as noise removal and feature extraction—are prerequisites for complex tasks like object recognition and perspective correction.

Methodology (Description of Implementations)

Draft Text:

2. Methodology & Implementation

The pipeline was implemented in Python, utilizing the OpenCV library for basic I/O and specific complex algorithms, while strictly adhering to the requirement of implementing core convolution and feature detection logic "from scratch." The processing workflow is divided into four stages.

2.1. Image Representation and Noise Filtering (Part A)

Before processing, all images were converted from standard 8-bit integers (uint8) to floating-point representation (float64 in the range [0, 1]). This ensured mathematical precision during convolution operations and prevented overflow/underflow errors.

We implemented a **manual 2D convolution** function (`convolve2d_manual`), explicitly bypassing `cv2.filter2D`. This function utilizes vectorized NumPy operations to slide a kernel over the image, handling boundary conditions via reflection padding. To test robustness, we introduced synthetic noise (Gaussian, Salt & Pepper, and Poisson) to clean images.

- **Gaussian Smoothing:** We generated Gaussian kernels mathematically based on a user-defined sigma (σ) and kernel size. This filter was effective in suppressing Gaussian noise by averaging pixels with weighted importance given to the center.
- **Mean Filtering:** A simple box filter was also implemented for comparison, primarily to demonstrate the trade-off between noise reduction and edge preservation.

2.2. Edge Detection (Part B)

For structural analysis, we employed the Canny Edge Detector. This multi-stage algorithm was chosen for its ability to define strong edges while suppressing weak ones through hysteresis thresholding. The process involved Gaussian smoothing (to reduce noise sensitivity), computing intensity gradients, performing non-maximum suppression (to thin edges to 1-pixel width), and finally linking edges based on high and low thresholds. This step was crucial for visualizing the geometric structure of the target objects (e.g., buildings).

2.3. Corner Detection (Part C)

We implemented and compared two corner detection algorithms to find keypoints:

- **Moravec Detector:** This early approach was implemented by computing the Sum of Squared Differences (SSD) between an image patch and its shifted versions in four cardinal/diagonal directions. While simple, it proved sensitive to noise and anisotropic features.
- **Harris Corner Detector:** We implemented the more robust Harris detector, which computes the autocorrelation matrix M for a window around each pixel. By analyzing the eigenvalues of M (via the response function R = det(M) - k(trace(M))^2), we could differentiate between flat regions, edges, and corners. This method provided rotation-invariant and significantly more stable keypoints compared to Moravec.

2.4. Document Rectification (Part D)

The final tool integrates the previous steps to perform projective rectification.

- **Keypoint Selection Strategy:** Since the Harris detector finds *all* corners (including text and texture), we developed a heuristic-based selection method. The image is first pre-processed with a heavy Gaussian blur to suppress text details. We then divide the image into four quadrants (Top-Left, Top-Right, Bottom-Left, Bottom-Right) and select the single Harris corner in each quadrant that maximizes the

distance from the image center (or minimizes distance to the image corners).
- **Homography & Warping:** Using the four selected source points and a set of destination points representing a flat rectangle (calculated based on the estimated width and height of the document), we computed the Homography matrix. Finally, an inverse perspective warp was applied to map the source pixels to the destination grid, producing the rectified output.

Moreover, we present you with a summary of what each function achieves!

1. Helper & Conversion Functions

- **to_float64**: Converts an input 8-bit integer image (0-255) into a 64-bit floating-point representation normalized to the range [0, 1] for precise mathematical operations.
- **to_uint8**: Converts a floating-point image back to 8-bit integer format (0-255) by scaling and clipping values, suitable for display and saving.

2. Convolution & Filtering (Part A)

- **convolve2d_manual**: Implements 2D spatial convolution from scratch. It pads the input image (using reflection), flips the kernel, and computes the weighted sum of the kernel and image patch at every pixel location using vectorized NumPy operations.
- **create_gaussian_kernel**: Generates a normalized 2D Gaussian filter kernel based on a specified standard deviation (sigma) and kernel size, used for smoothing and noise reduction.
- **create_mean_kernel**: Generates a normalized box filter (mean kernel) where all coefficients are equal, used for basic smoothing.
- **filter_color_image**: specific wrapper that applies the manual convolution to color images. It can operate in two modes: processing the Value channel (HSV) only to preserve chroma, or processing each RGB channel independently.

3. Noise Generation

- **add_noise**: Injects synthetic noise into an image to test filter robustness. Supports Gaussian noise (additive), Salt & Pepper (impulse), and Poisson (photon shot noise) models.

4. Feature Detection (Part C)

- **detect_moravec_corners**: Implements the Moravec corner detector by calculating the Sum of Squared Differences (SSD) between an image patch and its neighbors in four directions (horizontal, vertical, diagonal).
- **detect_harris_corners**: Implements the Harris corner detector. It computes the structure tensor (autocorrelation matrix) for each pixel and calculates a corner response score based on the determinant and trace of that matrix.

5. Rectification (Part D)

- **find_corners_using_harris_heuristic**: A higher-level function that automates corner selection. It applies Gaussian blur to suppress text, runs the Harris detector, divides the image into four quadrants, and selects the corner in each quadrant furthest from the image center.
- **rectify_document**: Performs the geometric transformation. It takes the four selected corners, computes the destination dimensions, calculates the Homography matrix using four-point correspondence, and applies an inverse perspective warp to produce the flattened output.

Experiments:

Part A: Convolution, Noise, and Smoothing Results

1. Noise Simulation Models

Referencing image_154b00.jpg

We implemented three noise models to test filter robustness:

- **Gaussian Noise:** Additive noise with a normal distribution. Visually appears as a uniform "grain" across the texture. The high-frequency detail of the brick texture partially masks this noise compared to flat regions.
- **Salt & Pepper (S&P) Noise:** Impulse noise randomly saturating pixels to min (0) or max (1). It appears as distinct black and white artifacts that destroy local pixel information. This noise type is the most visually intrusive on the structural pattern.
- **Poisson Noise:** Signal-dependent shot noise. It resembles Gaussian noise but is correlated with pixel intensity. In the well-lit brick texture, it appears similar to Gaussian grain but maintains a more natural photographic variance.

2. Smoothing & Denoising Performance

Referencing image_154b7d.jpg

We compared Manual Mean and Manual Gaussian filters (kernel size 5 x 5, σ=1.5) on S&P noise:

- **Mean Filter (Box Blur):** Performs poorly on impulse noise. By averaging the outlier pixels (salt/pepper) with neighbors, it spreads the error, creating large, blurry gray smudges and significantly degrading the sharpness of the mortar lines.
- **Gaussian Filter:** Offers slightly better preservation of structural edges compared to the mean filter due to the center-weighted kernel. However, like all linear filters, it struggles to eliminate S&P noise without blurring the image significantly. It is mathematically better suited for suppressing Gaussian noise than impulse noise.

3. Color Processing Strategy

Referencing image_154bb5.jpg

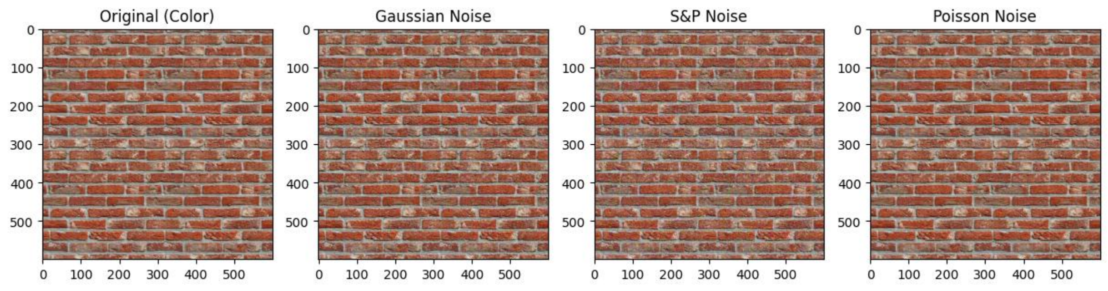
We evaluated two strategies for filtering color images:

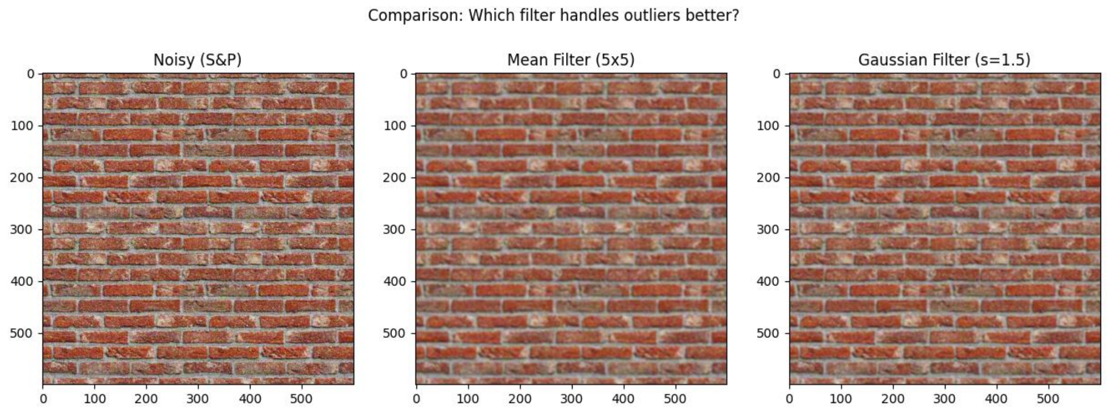
- **Per-Channel (RGB):** Applying convolution independently to R, G, and B channels.
- **Intensity-Only (HSV):** Converting to HSV, filtering only the Value (V) channel, and merging back.

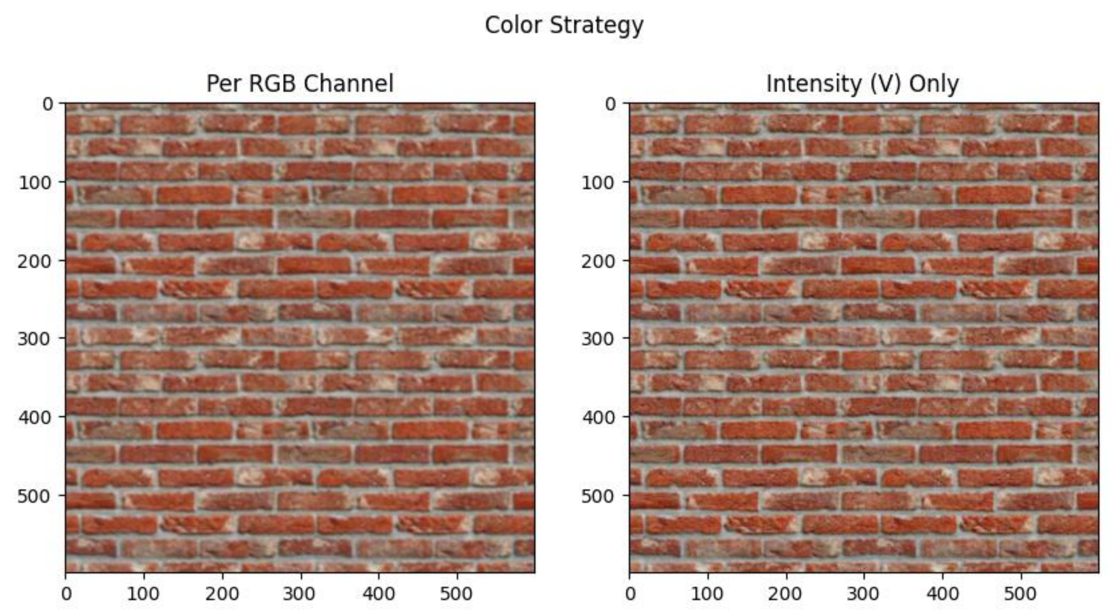
Conclusion & Selection:

While visually similar in static textures, the Intensity-Only (HSV) strategy was selected for the pipeline.

1. **Computational Efficiency:** Requires only 1 convolution operation instead of 3.
2. **Color Integrity:** Filtering RGB channels independently can introduce chromatic aberrations (color shifting) at edges. Processing only the intensity preserves the original Hue and Saturation data , ensuring that smoothing only affects luminance, not color information.







Part B: Edge Detection Results

1. Gradient Magnitude vs. Simple Thresholding

Referencing the Gradient Magnitude image

We visualized the raw gradient magnitude computed from the Sobel filters. While the structural outlines of the building were visible, the edges were thick and blurred. Applying a simple binary threshold to this magnitude resulted in bloated, multi-pixel wide edges. This demonstrated that simple thresholding is insufficient for precise edge localization, justifying the need for the Non-Maximum Suppression step implemented in the Canny algorithm to thin edges to a single pixel width.

2. Manual Canny Implementation Performance

Referencing the "My Canny" image

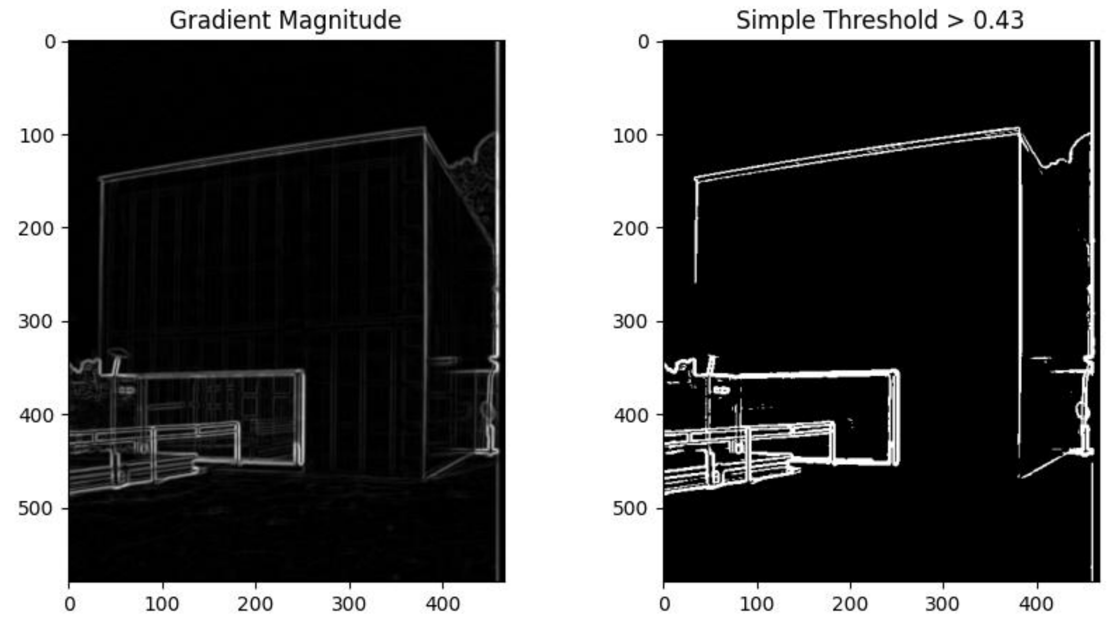
Our manual implementation of the Canny detector was tuned to prioritize major structural elements. It successfully extracted the dominant geometric lines of the building (roofline, corners, and railing) while effectively suppressing high-frequency texture noise. The result is a clean, sparse edge map that highlights the object's shape without being cluttered by environmental details like the grass or sky.

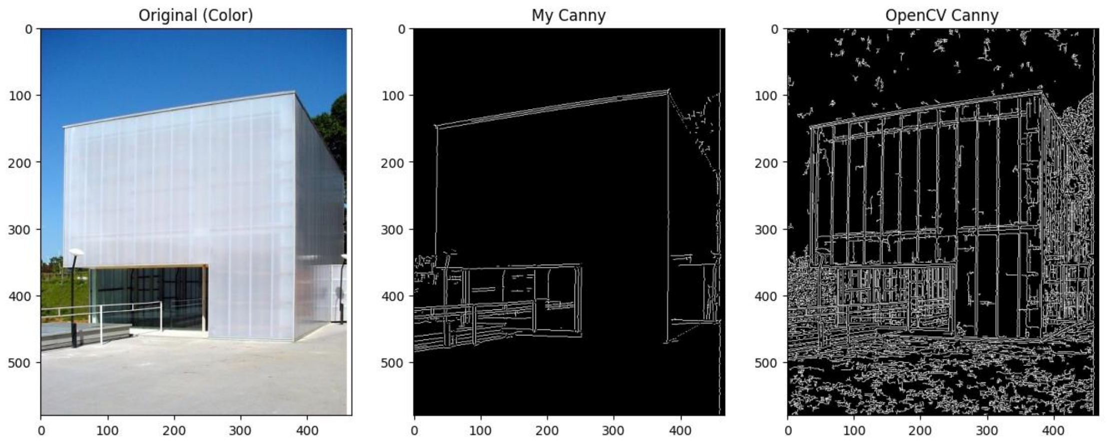
3. Comparison with OpenCV Canny

Referencing the "OpenCV Canny" image

The OpenCV implementation, likely using optimized gradient operators (like Scharr) and different default sensitivities, produced a much denser edge map. It detected fine textures including the grass on the ground, leaves on the trees, and subtle reflections in the windows.

Conclusion: While the OpenCV result contains more data, our manual implementation is great for the specific task of identifying the building's geometry, as it naturally filtered out the irrelevant "texture noise" that would complicate downstream tasks like corner detection or rectification.





Part C: Corner Detection Results

1. Moravec Detector Performance *Referencing the Moravec images (Book, Building, Sign)* The Moravec algorithm successfully identified high-contrast regions but demonstrated significant limitations in distinguishing true corners from edges.

- **Edge Sensitivity:** In the book and sign images, the detector generated dense clusters of points along straight lines (e.g., the book spine, the diagonal edges of the sign). This confirms that Moravec is sensitive to unidirectional intensity

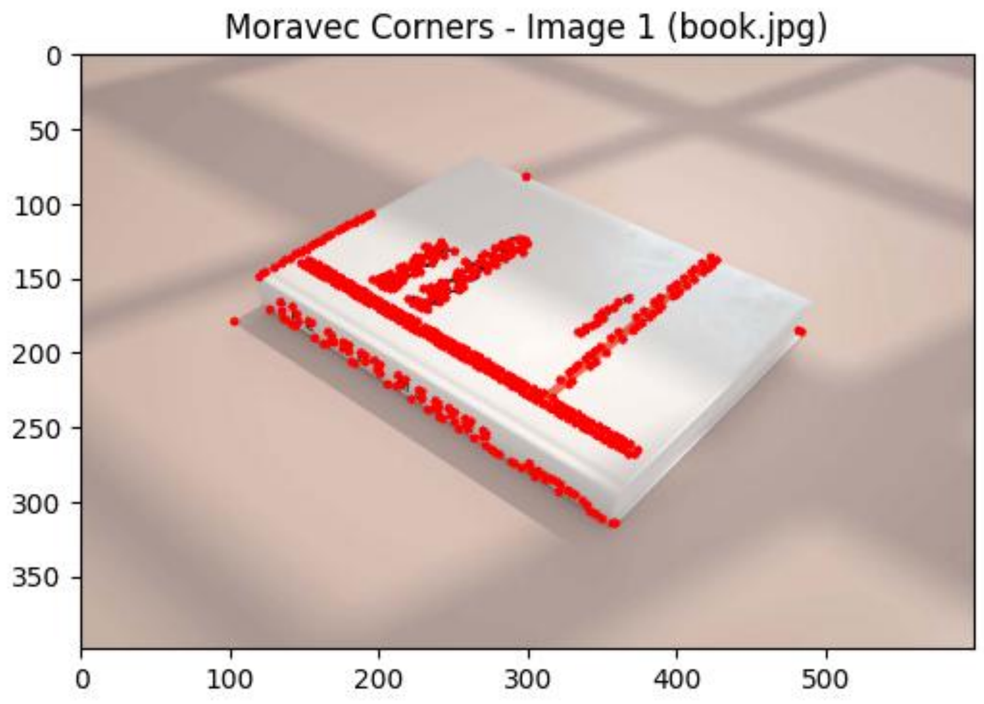
changes, failing to suppress linear edges where the shift in the direction of the edge yields a low score.
- **Noise Sensitivity:** In the building image, the detector was overwhelmed by the high-frequency texture of the trees and ground, treating random noise and foliage as corners.

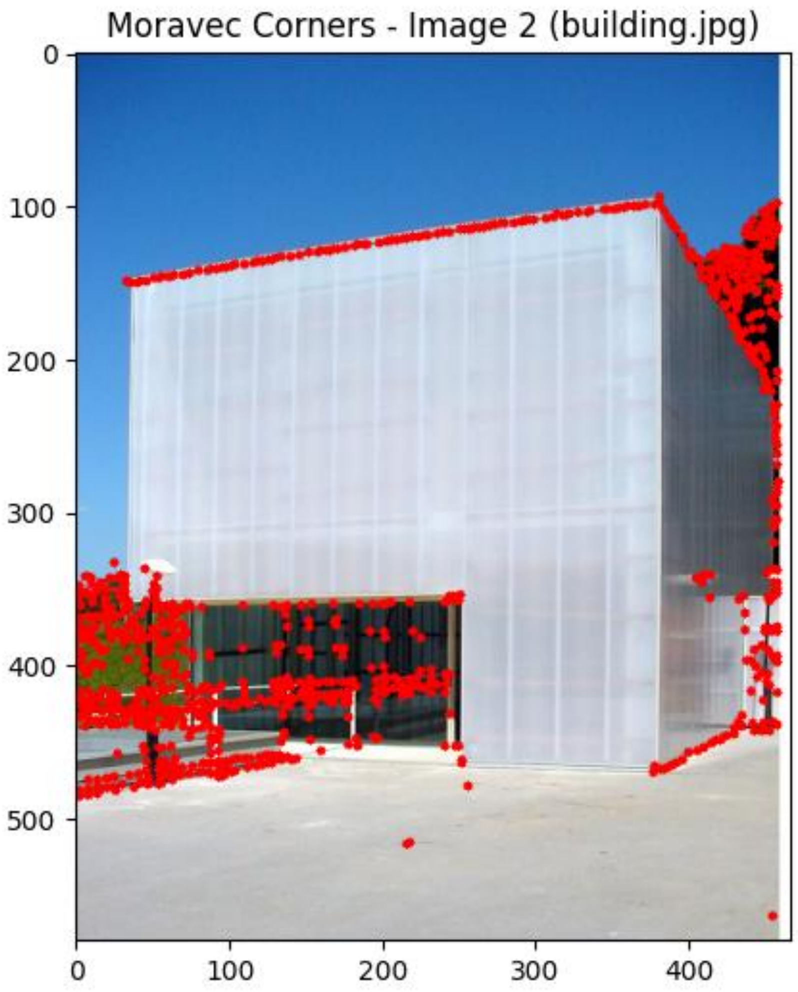
2. Comparison: Harris vs. Moravec *Referencing the Comparison image (Book)*
Comparing the two algorithms on the same subject reveals the robustness of the Harris detector.

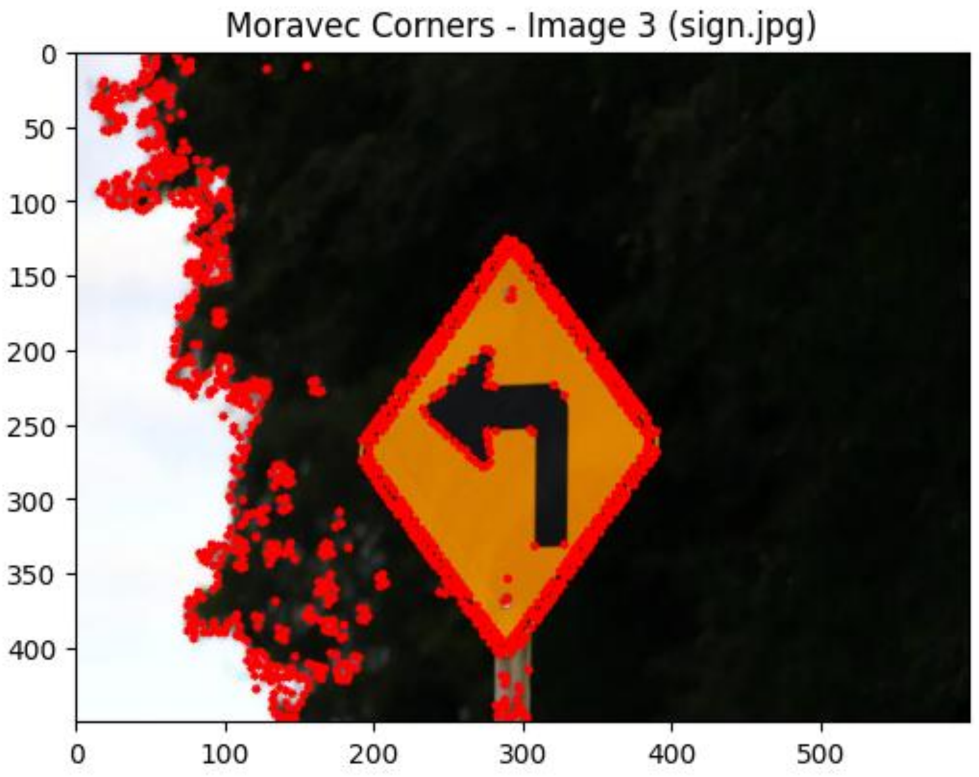
- **Selectivity:** While Moravec triggered along the straight edges of the book cover, the Harris detector ignored these linear features. Harris successfully identified points where intensity changed in two orthogonal directions, resulting in a sparse map focused on the actual corners of the book and the sharpest text characters.
- **Stability:** The Harris points were more stable and spatially localized compared to the scattered response of the Moravec detector.

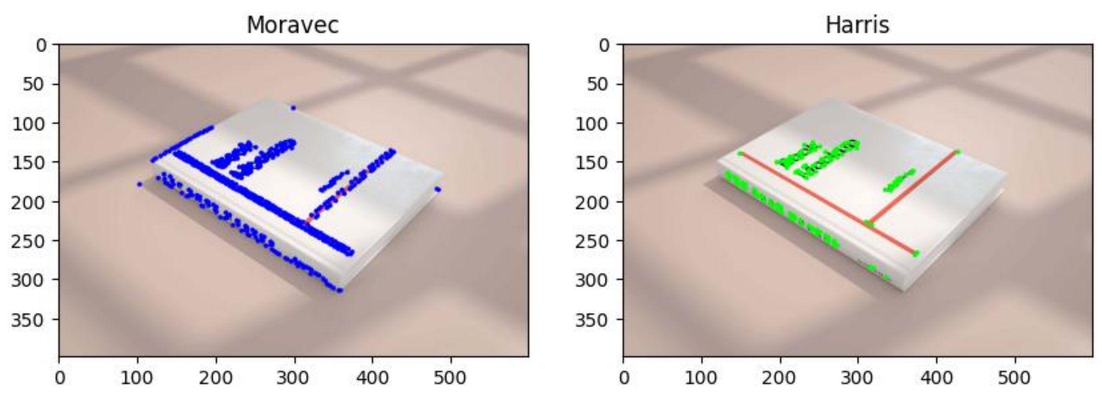
3. Multi-Scale Analysis (Gaussian Pyramid) *Referencing the Pyramid Level images*
We applied the Harris detector across three levels of a Gaussian pyramid to evaluate scale dependence.

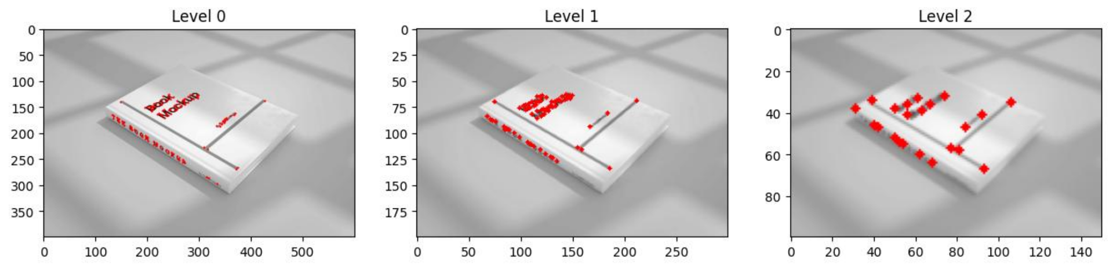
- **Level 0 (Original):** The detector picked up fine details, including the serifs of the text characters on the book cover.
- **Level 2 (Downsampled):** As the image resolution decreased and high-frequency details were smoothed out, the text corners disappeared. Only the dominant structural corners of the book object remained. This demonstrates that keypoint detection is scale-dependent; lower resolutions favor macro-geometry, while higher resolutions capture micro-texture.




Moravec Corners - Image 2 (building.jpg)
0
100
200
300
400
500
0
100
200
300
400


Moravec Corners - Image 3 (sign.jpg)
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500





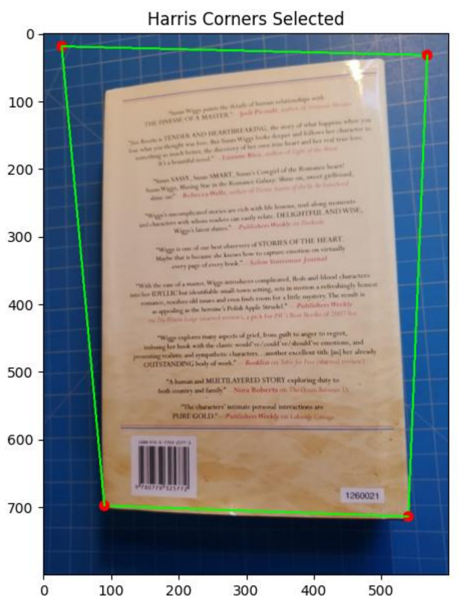
Part D: Projective Rectification Results

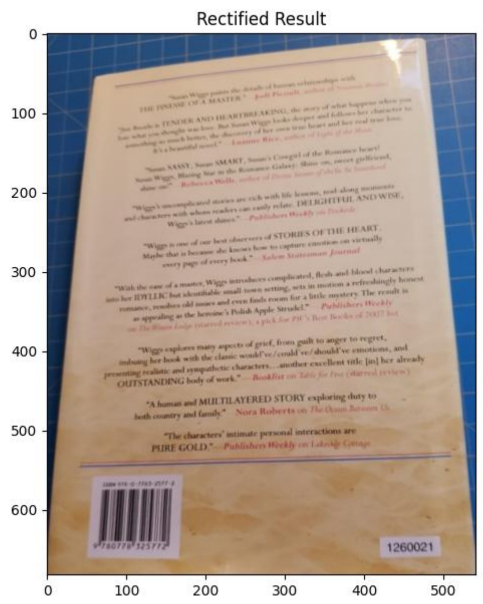
1. Automated Corner Selection Strategy *Referencing the "Harris Corners Selected" image* To automate the extraction of the document without manual intervention, we employed a geometric heuristic combined with the Harris detector. Crucially, we applied a heavy **manual Gaussian blur** (kernel size 15) prior to detection. This pre-processing step effectively suppressed high-frequency internal features like text and barcodes, ensuring the Harris response was dominated by the low-frequency structural corners of the book object.

2. Heuristic Performance The results demonstrate the success of the quadrant-based selection logic. Despite the presence of high-contrast text which typically generates strong corner responses, the algorithm successfully identified the four physical vertices of the book cover. By selecting the single strongest Harris point in each quadrant that maximized the distance from the image center, the system "snapped" to the document boundaries rather than the content.

3. Quality of Rectification *Referencing the "Rectified Result" image* The computed Homography transformation successfully removed the perspective distortion. The output demonstrates correct geometric recovery:

- **Parallelism:** The vertical edges of the book, originally converging due to perspective, are now parallel.
- **Alignment:** Horizontal text lines are straightened, and the barcode is restored to a rectangular shape.
- **Aspect Ratio:** The dimension calculation based on Euclidean distance provided a realistic aspect ratio for the rectified output.





3. **Sensitivity and Limitations** The approach relies heavily on the "blank" background assumption. While the blur removes internal text, the quadrant heuristic is sensitive to background clutter. If a high-contrast object appeared in the background near a corner (e.g., the white grid lines on the mat becoming too distinct), the algorithm might mistakenly select it as a document corner. Therefore, the blurring parameter is the critical factor balancing text suppression against background noise isolation. Contour Detection is the standard best!