Giorgio Mendoza
CS549-E24-E01
Dr. Alexandros Lioulemes

Lab Report #2

Part 1:

Part 1 requires implementing three new features to the camera app: Sobel X, Sobel Y and Canny edge detector using APIs from OpenCV. For the Sobel filters, the application can process the video stream to detect horizontal (Sobel X) or vertical (Sobel Y) edges. The trackbar allows the user to adjust the Sobel kernel size dynamically, ensuring it remains an odd number for effective edge detection.

The Canny edge detection was incorporated to highlight edges clearly in the video feed. Users can adjust the upper and lower thresholds for the Canny detector using two separate trackbars.
Additionally, we added global variables as before for keyboard controls that allow users to toggle the application of the Sobel X, Sobel Y, and Canny filters.

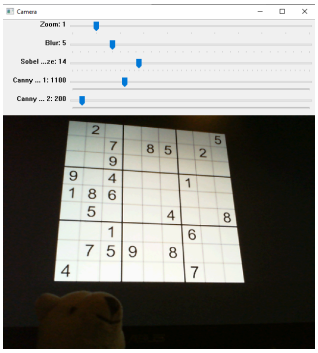

Figure 1: Original Image

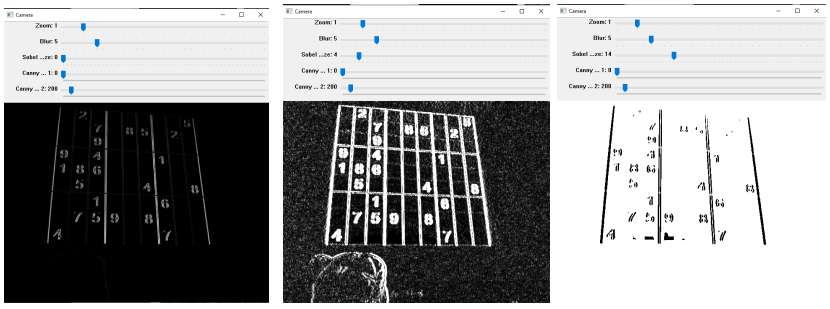

Figure 2: Sobel X Comparison

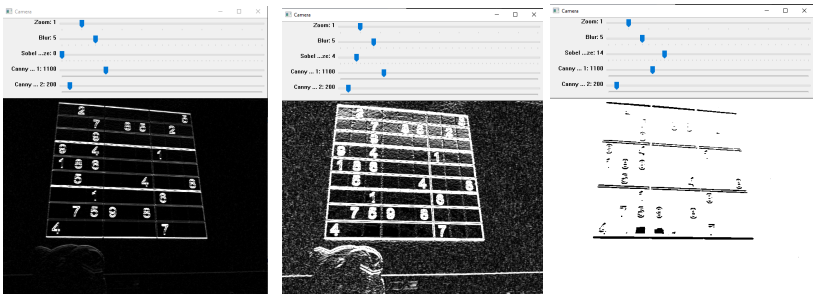

Figure 3: Sobel Y Comparison

The Sobel X and Y filters are applied to the images to enhance horizontal and vertical edges, respectively. From the provided screenshots, it is evident that even at low kernel sizes, both Sobel X and Sobel Y effectively highlight the relevant directional edges. Specifically, Sobel X emphasizes vertical edges and minimizes the visibility of horizontal lines, while Sobel Y enhances horizontal lines and suppresses vertical ones.

As the kernel size increases, the images begin to exhibit more noise, as visible in the second set of screenshots. This added noise, coupled with the increased averaging effect of larger kernels, tends to obscure finer details. In the third set of images, what might appear as data loss is actually the blurring of fine details and merging of close features due to this averaging. Despite this, the primary characteristics of the Sobel X and Y filters are still visible illustrating the filters edge detection properties.

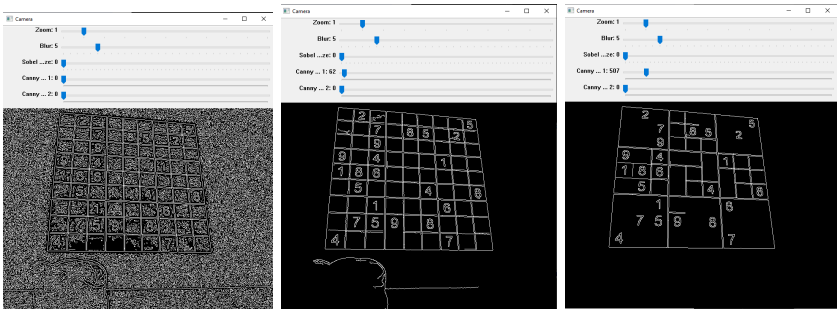

Figure 4: Canny Edge First Threshold Only

Figure 4 illustrates the impact of varying the first threshold value in the Canny edge detection algorithm. When the threshold is set at 0 in the leftmost image, the edge detector is extremely sensitive, picking up minute changes in pixel intensity, which results in a noisy and cluttered image.

As the threshold is increased to 62 in the middle image, the algorithm filters out less significant edges, reducing noise and focusing only on the more pronounced edges. This enhancement retains important structural features such as the outer borders and major grid lines of the Sudoku puzzle.

Further increasing the threshold to 507, as shown in the rightmost image, leads to the detection of only the most pronounced edges. This higher setting causes finer details, like the internal grid lines of the Sudoku squares, to disappear, emphasizing the strongest contrasts and simplifying the visual output. Each adjustment to the threshold demonstrates how the sensitivity of the Canny algorithm can be tuned to either capture finer details or to focus on the most significant edges, depending on the desired outcome.

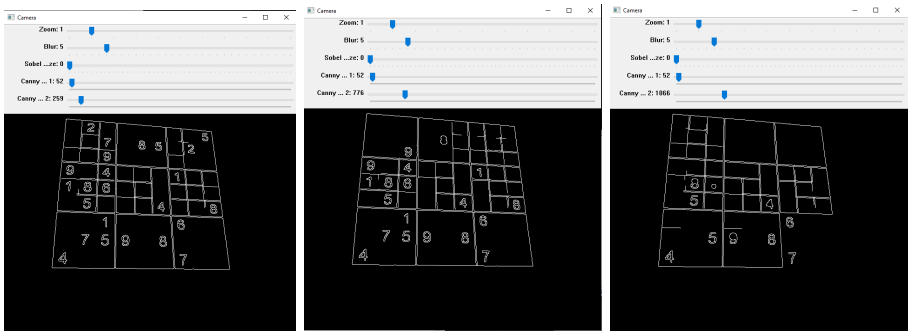

Figure 5: Canny Edge Second Threshold + Small First Threshold Value

Figure 5 uses both the first and second thresholds in the Canny edge detection on a Sudoku puzzle image; the first threshold is maintained at a constant value of 152 across all three images for consistency.

In the leftmost image with a second threshold of 259, the algorithm effectively retains most of the major grid lines while beginning to suppress some of the internal square edges. As the second threshold increases to 776 in the middle image, the edge detection becomes more selective, further reducing the visibility of internal square edges and focusing primarily on the most pronounced edges, such as the outer borders and major grid lines of the Sudoku puzzle.

In the rightmost image, the selectivity is heightened to a point where even some outer borders begin to disappear, showcasing how higher second threshold values progressively eliminate weaker and moderately strong edges, leaving only the most distinct edges visible.

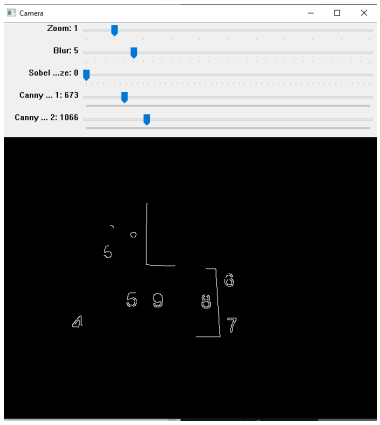

Figure 6: Both Thresholds with High Values

The image above illustrates the case when both thresholds have high values. The key point here is that for a real application it's important to balance both thresholds to retain the areas of interest. Otherwise, even relevant data can be lost.

Part 2:

Part 2 is similar to part 1 except that we are implementing the filters from scratch instead of using the API from OpenCV additionally we are adding the Laplacian filter.

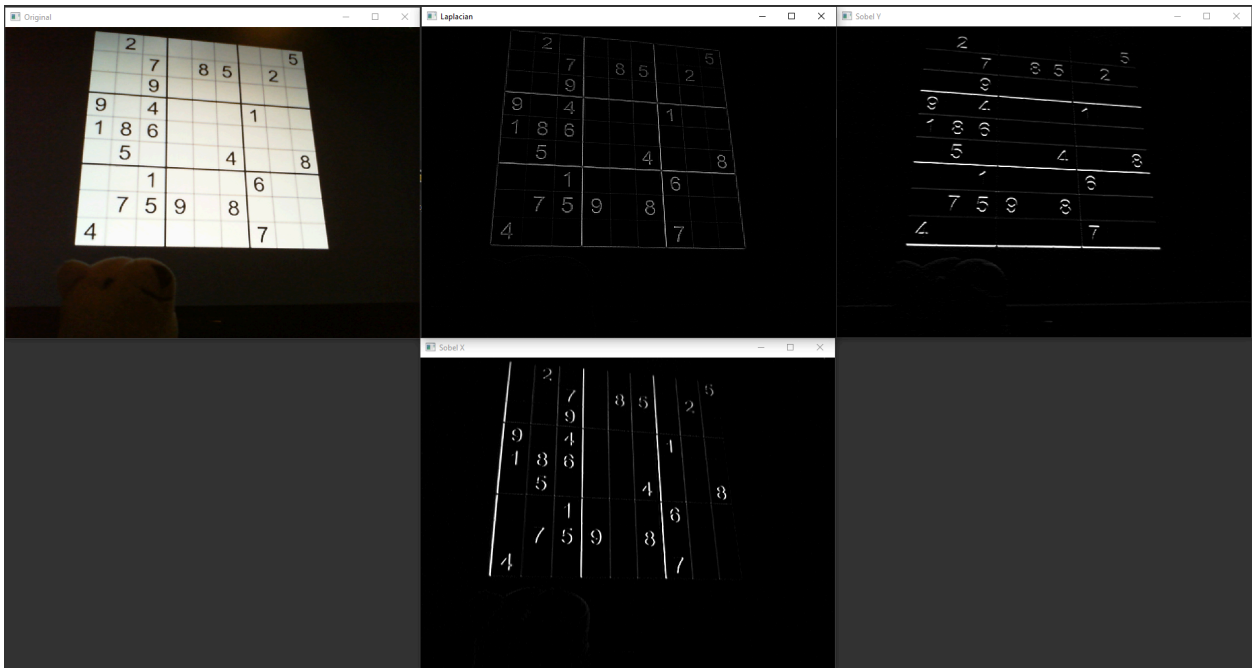

Figure 7: Original Image, Custom Laplacian, Sobel X and Sobel Y comparison

The second window shows the result of applying the custom Laplacian filter which emphasizes areas of rapid intensity change indicative of edges. Unlike the Sobel operator, the Laplacian is isotropic meaning it detects edges in all directions. The image clearly highlights all the boundaries and numbers with strong white lines against a dark background, capturing finer details compared to the original.

The third window demonstrates the effect of the custom Sobel X operator, which is designed to detect horizontal gradients. Therefore, it accentuates vertical edges in the Sudoku grid, as seen by the strong vertical lines and somewhat clearer visibility of the vertical aspects of the numbers.

The final window displays the output of the custom Sobel Y operator, focusing on vertical gradients and thus highlighting horizontal edges. This is evident from the enhanced visibility of the horizontal lines of the Sudoku grid and the tops and bottoms of the numbers. By pressing '4', the camera app successfully toggles to display these four windows, each labeled accordingly.

Part 3:

This part is about applying a few geometric transformations to a predetermined image, ‘Unity-Hall”.

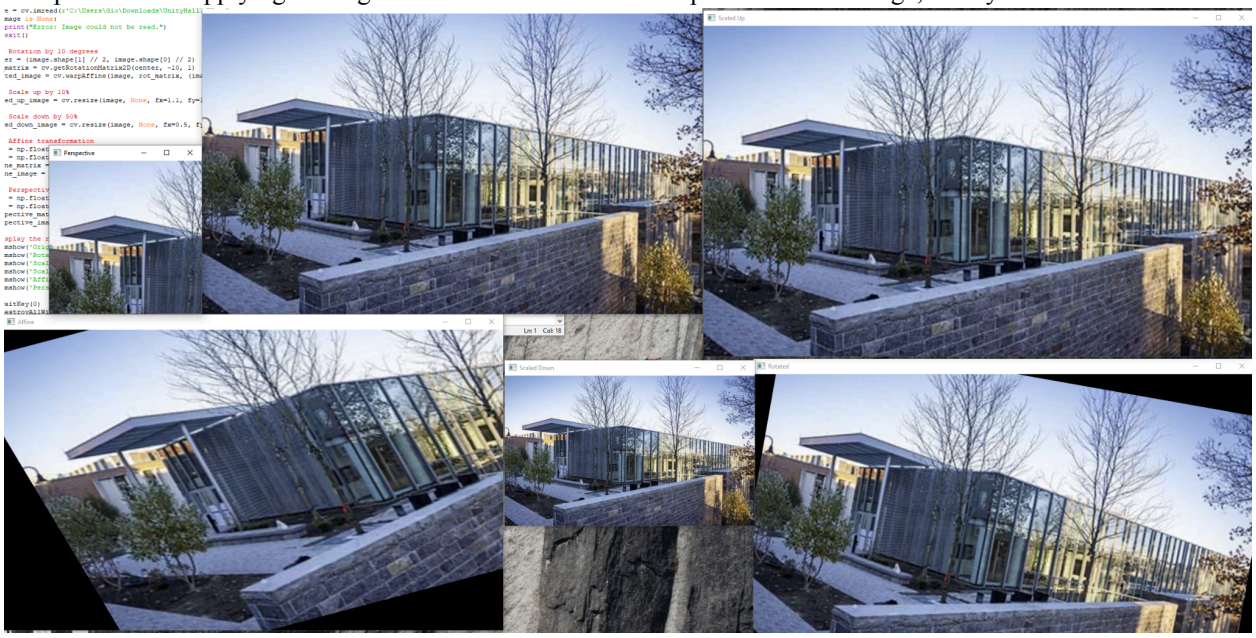

Figure 8: Original Image, Scale Up/Down, Perspective, Affine and Rotate.

The original, scaled-up, and scaled-down images are self-explanatory. The perspective transformation results in a zoomed-in effect because it enlarges a smaller part of the original image to fill a bigger area. This occurs when the designated new points are set closer together than the original points they correspond to, which causes the area they outline to appear larger.

The rotated and affine transformations change the image based on where specific points are placed. The affine transformation changes the image through a setup where three pairs of points dictate how the image will transform.

For example, moving a point from (50, 50) to (10, 100) shifts it leftward and downward. A point at (200, 50) slightly moves up but stays in line horizontally. Another point moves from (50, 200) to (100, 250), going rightward and downward. These shifts make the image seem as if it is rotating or tilting the other way, shaped by the movement of these points against the original layout of the image.

Part 4:

This segment of the code incorporates Harris Corner Detection and SIFT algorithms to the previous images. The primary goal is to show how these transformations impact the detection capabilities within an image.

The apply_harris function processes the image by converting it to grayscale before applying the Harris algorithm, which identifies corner points. These corners are then enhanced and distinctly marked in red to ensure visibility. Similarly, the apply_sift function converts the image to grayscale and employs the SIFT algorithm, known for its scale-invariance, to detect and highlight key points in green.

Once the transformations are applied, both Harris Corner Detection and SIFT are executed on each modified image. The output of these feature detection processes are displayed in separate subplots dedicated to each method. This visualization not only illustrates how geometric changes influence the detection results but also involves converting the images from BGR to RGB to maintain accurate color representation in the visual plots.

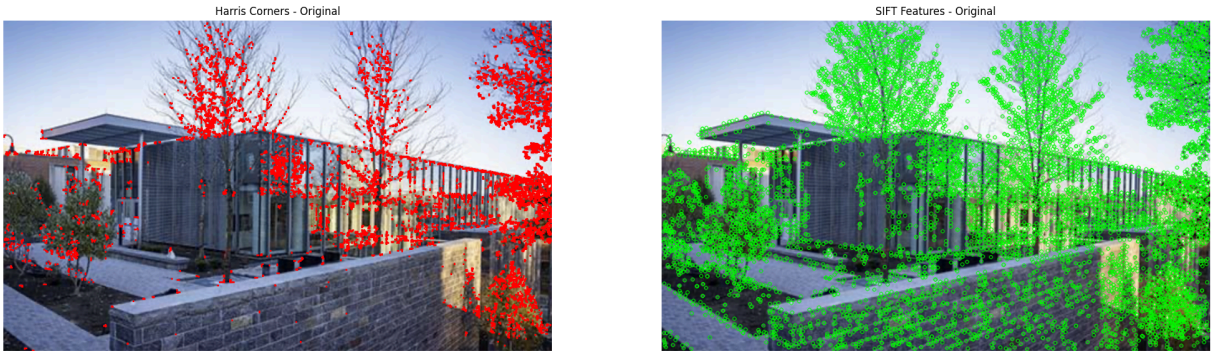

Figure 9: Original Images HCD & SIFT Algorithm Comparison

Harris Corner Detection focuses on identifying corners where the gradient of the image changes significantly in multiple directions. This algorithm is particularly sensitive to junctions where two edges meet, which typically result in high variations in image brightness.

In our context, the Harris algorithm emphasizes the areas in the trees because the patterns and textures within the foliage create many intersections where the gradient changes sharply; these are interpreted as corners by the Harris detector.

On the other hand, SIFT is designed to detect and describe local features in images. The algorithm looks for regions in the image where there are significant changes in all directions, which are robust to changes in scale, noise, and illumination. SIFT not only identifies corners but also more complex features such as blobs or textured areas.

Therefore, in our image, SIFT highlights a broader range of features, not just the corners but also edges, architectural elements, and other textures throughout the scene, including but not limited to the trees.

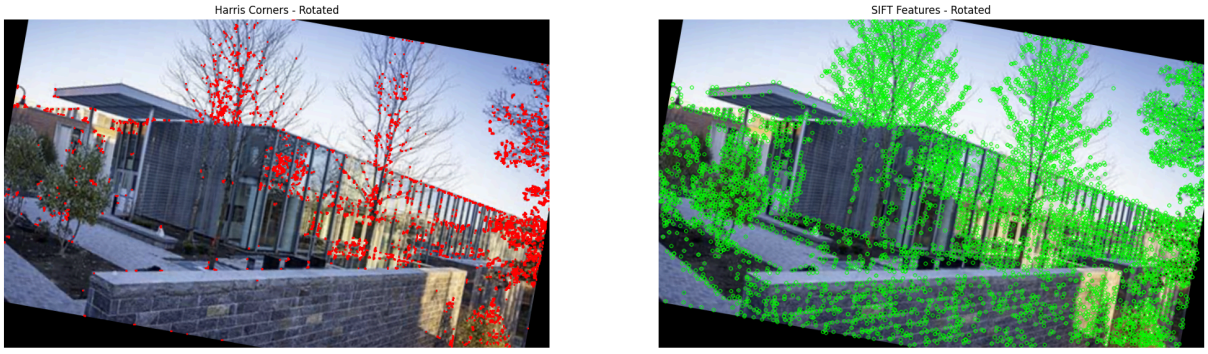

Figure 10: Rotated Images HCD & SIFT Algorithm Comparison

For the Rotate Image case described, the HCD algorithm displays slightly fewer dots compared to Figure 9, while SIFT exhibits a similar number of green dots as seen in the previous case. This observation suggests that the SIFT algorithm may be more resilient to transformations than Harris Corner Detection, indicating greater stability in feature detection under these conditions."

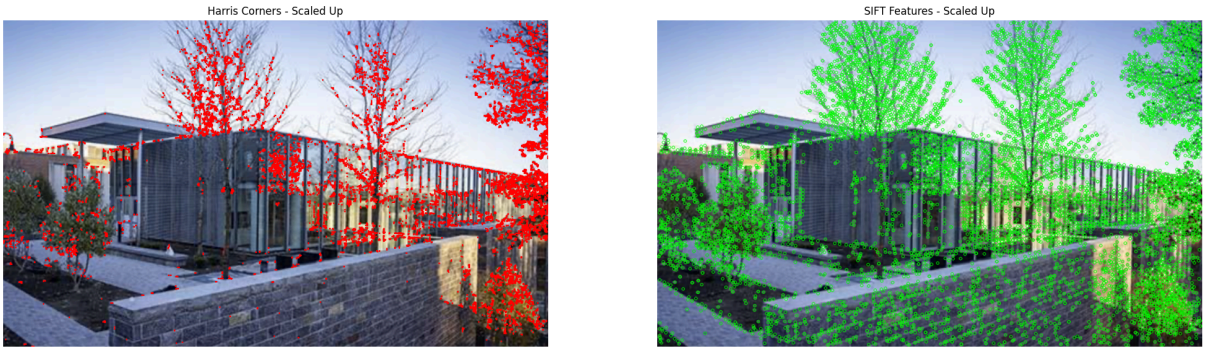

Figure 10: Scaled Up Images HCD & SIFT Algorithm Comparison

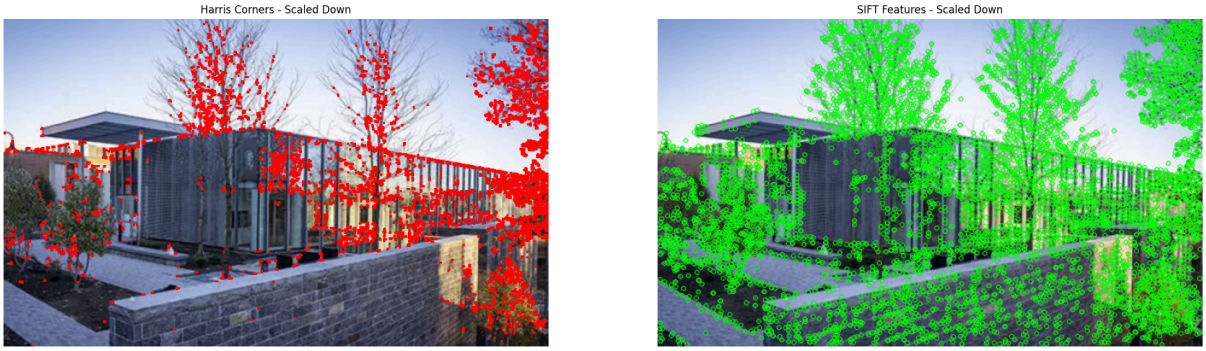

Figure 11: Scaled Down Images HCD & SIFT Algorithm Comparison

At first glance, the number of red and green dots seems similar for both the scaled-up and scaled-down images. However, upon closer inspection, the red dots in the scaled-up images appear more spread out, while in the scaled-down images, the density of both red and green dots increases. This is because the features are closer to each other in the smaller-scaled image, making them appear more concentrated.

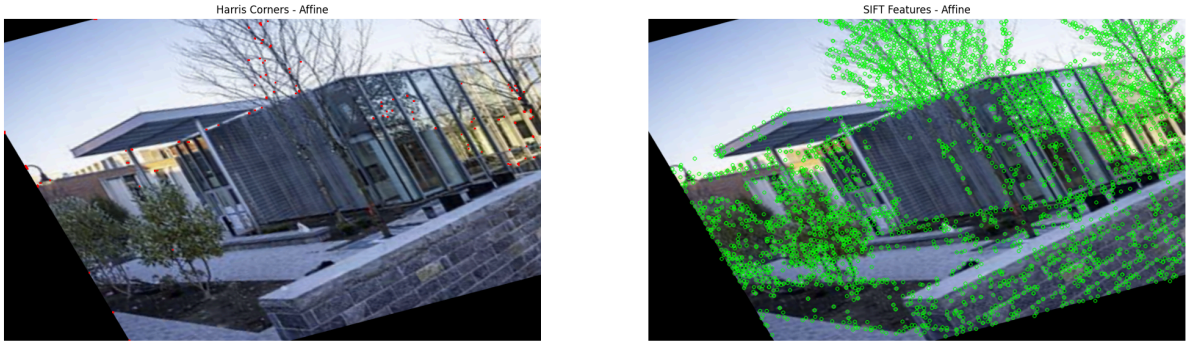

Figure 12: Affine Images HCD & SIFT Algorithm Comparison

The comparison of feature detection results after an affine transformation reveals some notable differences. In the HCD output, significantly fewer red dots are observed compared to previous cases, suggesting that HCD may be more sensitive to the effects of affine transformations, which include scaling, rotating, and shearing.

This sensitivity could lead to fewer corners being detected as the geometry of the image changes. On the other hand, the SIFT algorithm continues to perform consistently, although it too shows a slight decrease in the number of green

dots compared to the original image. This indicates that while SIFT is generally more robust to such transformations, it is not entirely unaffected.

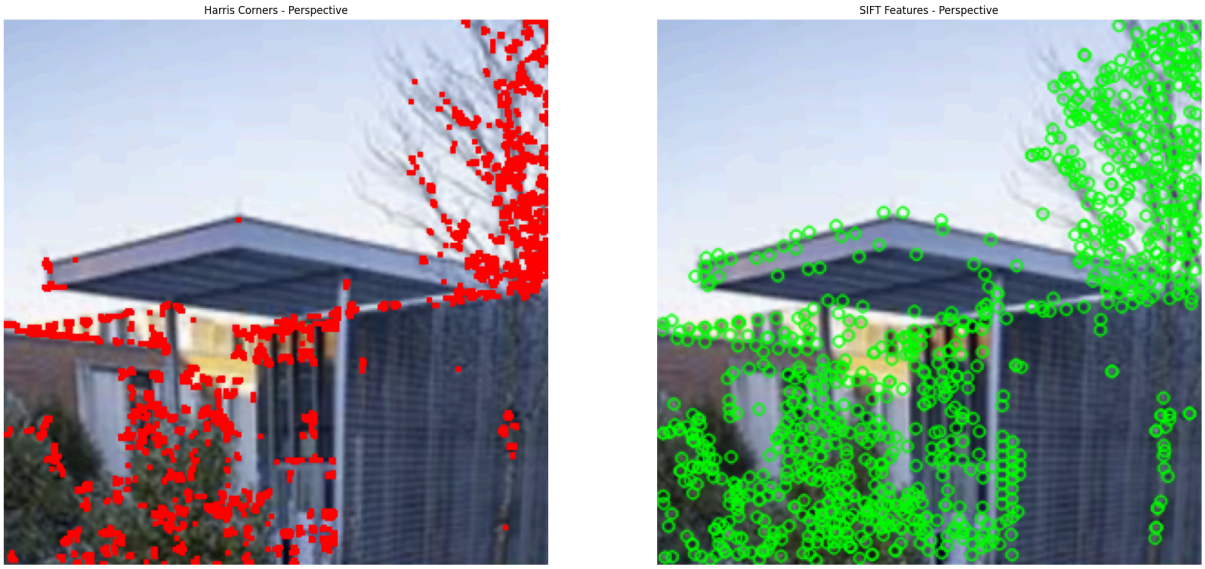

Figure 13: Perspective Images HCD & SIFT Algorithm Comparison

The last comparison involves images with a perspective transformation, which visually resemble a zoomed-in effect as shown above. In this case, both algorithms HCD and SIFT demonstrate similar effectiveness in detecting features within the trees and branches.

However, the SIFT algorithm distinguishes itself by identifying a greater number of features, marked by more green dots compared to the red dots from the HCD. Notably, SIFT also successfully captures additional features along the edges and corners of the building's roof, showcasing its comprehensive feature-tracking capability under the influence of perspective changes.

Conclusion:

The experiments demonstrate that while both algorithms are proficient at detecting significant features in images, they respond differently to transformations due to their inherent design characteristics.
Harris Corner Detection was particularly sensitive to transformations such as affine and perspective changes, showing a reduced number of detected corners in these scenarios. This indicates that HCD may be less suited for applications where images undergo significant geometric distortions but remains highly effective in less altered visual contexts.

Conversely, SIFT displayed resilience across all transformations, consistently detecting a broader range of features beyond mere corners, such as edges and textures. Despite slight reductions in feature counts under extreme transformations, SIFT's robustness makes it an excellent choice for scenarios requiring reliable feature detection across varied conditions and scales.

Perhaps the SIFT algorithm uses more resources compared to the HCD algorithm which is why it performs better. If that’s the case, then the HCD algorithm would probably be better used for constrained devices such as microcontrollers with limited RAM and storage while the SIFT could be more applicable in more robust hardware platforms such as NVIDIA Jetson modules. However, that was not explored in this lab.

Overall, these findings highlight the importance of choosing the appropriate feature detection algorithm based on specific application requirements and expected image transformations.