Name: Reyyan Qureshi - 169033850
Course: CP467

Task 1 & 2: Implementing Image Filters

Task a: Averaging Smoothing Filter

- **Explanation:** The averaging filter smooths the image by taking the average of the neighbor pixels within the 3x3 kernel size that was given. The result has the fine details smoothened out but the image is still slightly blurry.

Task b: Gaussian Smoothing Filter

- **Explanation:** The Gaussian filter smooths the image by applying a Gaussian function to the neighbors of each pixel. It preserves more structure than the averaging filter because the weights decrease smoothly as the pixels distance from the center.

Task c: Sobel Sharpening Filter

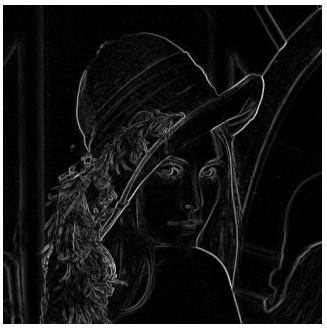
- **Explanation:** The Sobel filter highlights edges by emphasizing regions with gradients that have a high intensity. A pair of filters is applied to detect the horizontal and vertical edges and then it combines the results.

Comparison Between Task 1 and Task 2 Results

The scratch implementations and the built-in OpenCV filters produced very similar outputs, but the differences were only noticeable when looked at upclose. OpenCV's functions are definitely much more optimized, which means there is more accurate edge detection and smoother filtering. The OpenCV's Gaussian filter retained slightly more details around edges while still applying a decent blur. My Sobel filter produced edges that were thicker, probably because of the limitations I had in manually implementing the gradient calculation. I experimented with different kernel sizes for the Sobel filter, but even with larger kernels, OpenCV's optimized functions were still superior.


Average Smoothing


Gaussian Smoothing


Sobel Sharpening

Task 3: Edge Detection with Built-in OpenCV Functions

Task 3a: Marr-Hildreth Edge Detector

- **Explanation:** This detector applied a Laplacian of Gaussian (LoG) filter to detect edges by looking for zero crossings after smoothing the image with the Gaussian filter.
- **Results:** The Marr-Hildreth edge detector produced smoother and more connected edges, but it was more sensitive to noise.

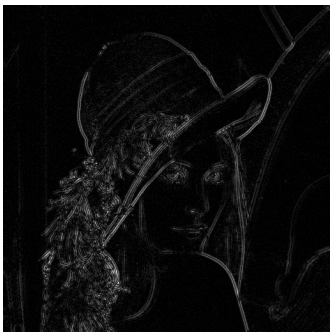
Task 3b: Canny Edge Detector

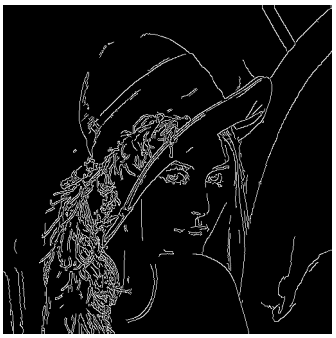
- **Explanation:** The Canny edge detector includes gradient computation, non-maximum suppression, and thresholding with edge tracking by hysteresis. It is more adaptable to noise and produces thinner edges.

- **Results:** The Canny edge detector produced sharper and more defined edges. It also handled noise better than the Marr-Hildreth method.

Comparison Between Marr-Hildreth and Canny

The key difference I saw between these two methods is in edge sharpness and noise sensitivity. As said before, the Marr-Hildreth detector produced more continuous edges but some of the edges were a little bit thicker because of over smoothing. In comparison, Canny produced thinner and more precise edges. I experimented with various sigma values in Marr-Hildreth and it showed that Canny was better at preserving detailed edges while controlling noise. The thresholding allowed for better control over the edges.


Marry-Hilderth edging


Canny edging

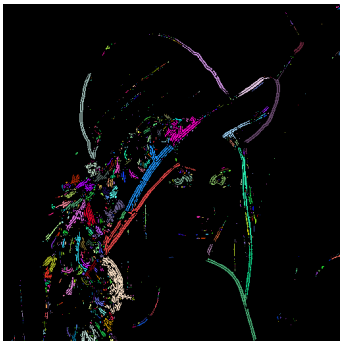
Task 4: Connected Component Labeling from Scratch

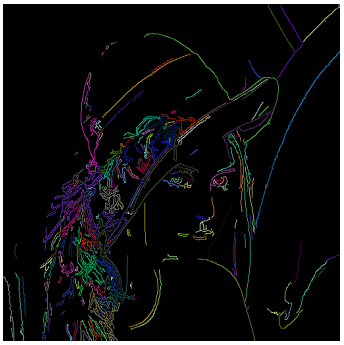
Results and Comparison

- **Marr-Hildreth:** In the Marr-Hildreth edge map, the connected component labeling produced larger and more connected regions mainly because of the smooth nature of the edges. I noticed that small details were often grouped into larger regions, which showed the smoother, more continuous edges produced. After experimenting with different thresholds in the labeling process, I found that too high of a threshold resulted in the loss of smaller components, while a lower threshold captured a bit more detail but also included more noise in the process.

- **Canny:** The Canny edge map was more detailed and finer, resulting in a larger number of smaller connected components. By experimenting with the labeling threshold, I noticed that the Canny detector was more precise when grouping fine edges, but any minor adjustments to the threshold would impact the number of detected components.

Conclusion: While the Marr-Hildreth detector produced more connected components, the Canny detector provided a more accurate representation of finer image features.


Marry-Hilderth edging


Canny edging

Bonus Task: Efficiency Comparison

Task	Implementation	OpenCV	Difference
T1: Averaging Filter	0.583014	0.000821	0.582193
T1: Gaussian Filter	0.721593	0.000127	0.721466
T1: Sober Filter	1.246944	0.000988	1.245956
T3: Marr-Hildreth	0.009442	0.000627	0.0088155
T3: Canny	0.001272	0.00396	0.000876

OpenCV has faster computation times mainly because of better optimizations. When writing from scratch, my implementations were computationally longer because of larger kernels. To improve the performance of my scratch implementations, I could potentially explore parallel processing or functions within NumPy.