16-385 Computer Vision, Spring 2025

Programming Assignment 1
Image Filtering and Hough Transform

Belle Connaught (bconnaug)

Due Date: Mon February 10, 2025 23:59 (ET)

Theory Questions

Q2.1 Median Filter

In the lecture, you learned about filters commonly used in image processing, such as Gaussian and box filters. Another widely used filter is the median filter, which replaces each pixel with the median value of the pixels in its neighborhood (e.g., a 3×3 window).

1. Is the median filter a convolution operation? Justify your answer in one or two brief sentences.

   The median filter is not a convolution operation. This is because convolution operations are linear and involve applying a kernel to an image, but the median filter is non-linear and involves replacing each kernel with the median value of pixels in its local neighborhood.

2. What are some advantages and disadvantages of the median filter compared to filters like Gaussian and box filters?

   Since it selects pixel values based on the local neighborhood, some advantages of the median filter include its ability to preserve edges and lines, as well as being highly effective at removing salt-and-pepper noise from images. Inherently, it also avoids bias towards higher or lower intensity values and is relatively simpler (when compared to Gaussian/box filters) since it requires no parameters beyond window size.

   Unlike Gaussian or box filters, which use simple operations like weighted sums and/or averages, the median filter requires sorting the values of n pixels in a given neighborhood to find median values (with a typical complexity of $O(n\log(n))$. Thus, a disadvantage of the median filter is its computational cost. It is also less effective at filtering out Gaussian noise because the latter consists of small, continuous variations that do not create outliers that the median filter is designed to remove.

3. Suppose you apply a Gaussian filter to an image followed by a median filter. Would the result be the same if you reversed the order? Explain why or why not.

   If the order were reversed, the result would not be the same:

   Applying a Gaussian filter first reduces Gaussian noise and small variations in the image (giving it a more uniform, smooth appearance). The median filter would then remove any outliers/salt-and-pepper noise.

   Applying the median filter first would still remove outliers and salt-and-pepper noise, but subtle variations from Gaussian noise would remain unsmoothed. When the Gaussian filter is applied afterward, it would smooth everything—including fine details and edges that were preserved by the median filter—potentially leading to a loss of clarity and sharpness in the resulting image.

Q2.2 Convolutions

Imagine you have a 1000×1000 image I and a 5×5 box filter F. In one case, you convolve the filter F with the image I (treating the filter as the kernel). In another case, you convolve the image I with the filter F (treating the image as the kernel).

1. Do these two operations produce the same result? Justify your answer in one or two brief sentences.

   These operations would produce the same result since convolution is commutative. Consequently, convolving I with F produces the same result as convolving F with I.

2. In practice, would there be any reason to choose one approach over the other?

   In practice, choosing to convolve F with I (treating the filter as the kernel) is better in terms of performance/efficiency. Since the filter is small, optimized techniques like fast Fourier transform (FFT) can be applied to speed up computation. Additionally, using the filter as the kernel improves cache efficiency because the small filter is repeatedly accessed while the image is processed in an orderly manner, reducing cache misses and improving overall performance.

   On the other hand, treating the image as the kernel is significantly more computationally expensive: instead of sliding a small filter over the image, this approach requires sliding the large image over the small filter, so every position in the output involves summing over all pixels of the original image rather than just a small region. This results in *tens of thousands* of times more multiplications, as each output pixel requires an entire image-sized computation rather than just a small 5×5 patch.

Q2.3 Hough Transform Line Parameterization

1. Show that if you use the line equation $\rho = x\cos\theta + y\sin\theta$, each image point (x, y) results in a sinusoid in (ρ, θ) Hough space. Relate the amplitude and phase of the sinusoid to the point (x, y).

   Incorporating the distance of a point with coordinates (x,y) from the origin, $\sqrt{x^2+y^2}$, we have:

   $$\rho = \sqrt{x^2+y^2} * \left(\frac{x\cos\theta}{\sqrt{x^2+y^2}} + \frac{y\sin\theta}{\sqrt{x^2+y^2}} \right) \tag{1}$$

   According to the sum of angles formula for sine,
   $\sin(\phi+\theta) = \sin\phi\cos\theta + \cos\phi\sin\theta$ with

   $$\sin(\phi) = \frac{x}{\sqrt{x^2+y^2}}, \quad \cos(\phi) = \frac{y}{\sqrt{x^2+y^2}}$$

   So, Equation (1) now becomes:

   $$\rho = \sqrt{x^2+y^2} * \sin(\phi+\theta)$$

   Let $A = \sqrt{x^2+y^2}$. Then $\rho = A\sin(\phi+\theta)$.
   Therefore, we have:

   $$\boxed{A = \sqrt{x^2+y^2}} \quad \boxed{\phi = \tan^{-1}(\frac{y}{x})}$$

   For amplitude A and phase shift ϕ.

2. Why do we parametrize the line in terms (ρ, θ) instead of the slope and intercept (m, c)? Express the slope and intercept in terms of (ρ, θ).

   Parameterizing the line using (ρ, θ) allows for more robust Hough Transform representations than if we were to use the (m, c) parametrization. For example, in the slope-intercept form $y = mx + c$, the slope m becomes undefined for vertical lines (since $m = \frac{\Delta y}{\Delta x}$, and $\Delta x = 0$). The (ρ, θ) parametrization avoids this issue by representing all lines—including vertical ones—consistently.

   For instance, a vertical line at $x = a$ with $a \in \mathbb{R}$ is represented in (ρ, θ) form with $\theta = \pm\frac{\pi}{2}$, where $\sin\theta = \pm 1$. After substituting θ into $\rho = x\cos\theta + y\sin\theta$, we simply have $\rho = a$, providing a clear representation of the line.

   Considering the line equation $\rho = x\cos\theta + y\sin\theta$ we isolate y as such:

   $$\rho = x\cos\theta + y\sin\theta$$
   $$y\sin\theta = -x\cos\theta + \rho$$
   $$y = -\frac{\cos\theta}{\sin\theta}x + \frac{\rho}{\sin\theta}$$

Comparing terms from the slope-intercept formula $y = mx + c$, we have:

$$\boxed{m = -\frac{\cos\theta}{\sin\theta}} \quad \text{and} \quad \boxed{c = \frac{\rho}{\sin\theta}}$$

3. Assuming that the image points (x, y) are in an image of width W and height H, that is, $x \in [1, W]$, $y \in [1, H]$, what is the maximum absolute value of ρ, and what is the range for θ?

   Since the image has width W and height H, the maximum absolute value of ρ is $\boxed{\sqrt{W^2 + H^2}}$.
   The range for θ is $\boxed{\theta \in [0, 2\pi)}$.

4. For point (10,10) and points (20,20) and (30,30) in the image, plot the corresponding sinusoid waves in Hough space, and visualize how their intersection point defines the line. What is (m, c) for this line? Please use Python to plot the curves and report the result in your write-up.

   Using points (10, 10) and (20, 20) and the slope formula:

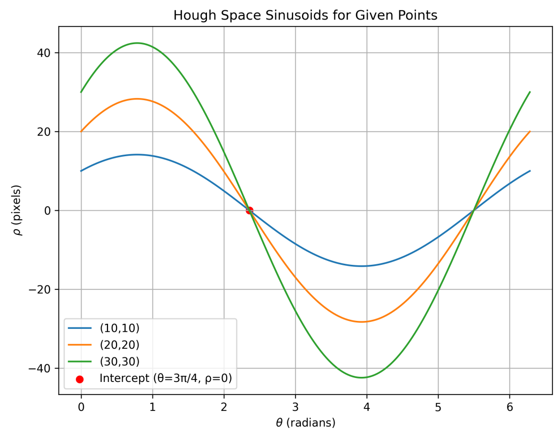
$$m = \frac{\Delta y}{\Delta x}$$
$$m = \frac{20 - 10}{20 - 10}$$
$$m = 1$$

   Using point (30, 30) and the slope-intercept formula:

$$y = mx + c$$
$$30 = (1)(30) + c$$
$$c = 0$$

   So (m, c) for this line is $\boxed{(1, 0)}$.

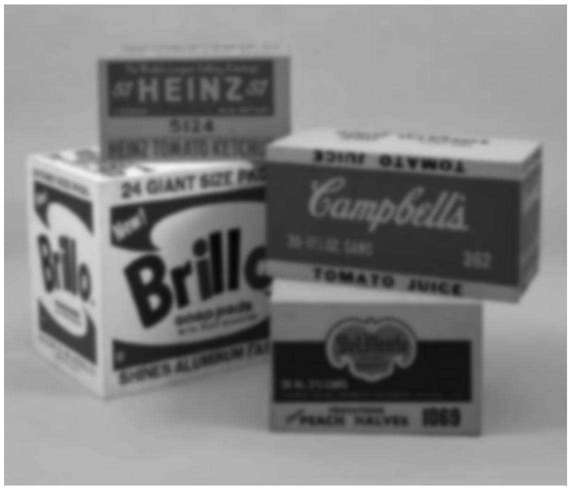
   <u>Curve plot with $0 \leq \theta \leq \pi$:</u>



Implementation

Q3.1 Convolution

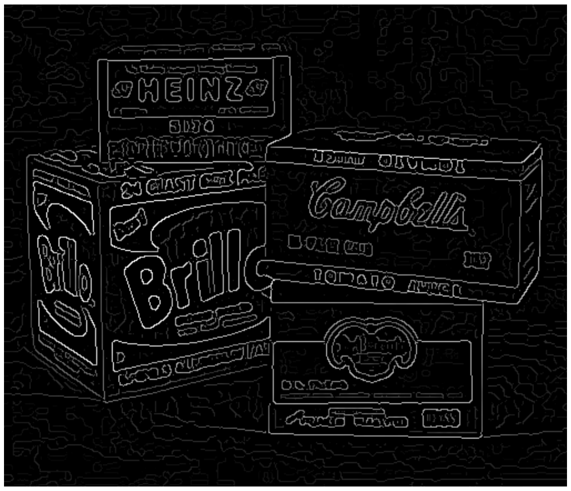
img06.png



myImageFilter(img06)

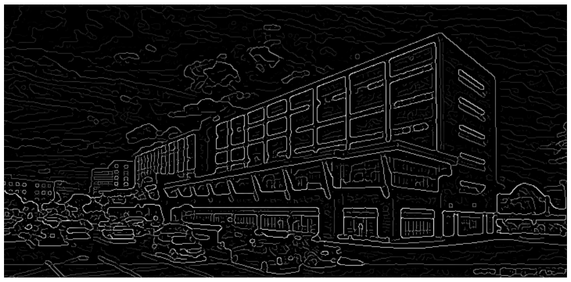
Q3.2 Edge Detection

img06.png



myEdgeFilter(img06, sigma=2)

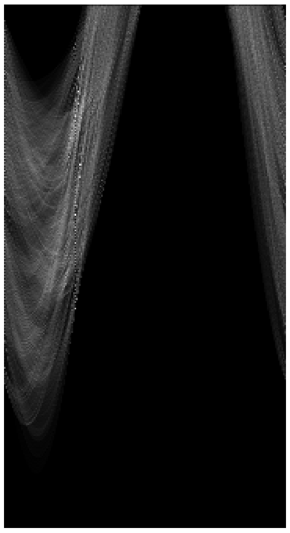
img09.png



myEdgeFilter(img09, sigma=2)

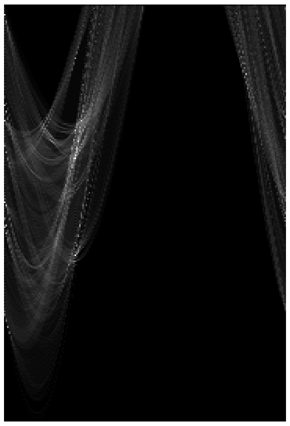
Q3.3 The Hough Transform

img06.png



myHoughTransform(img06)

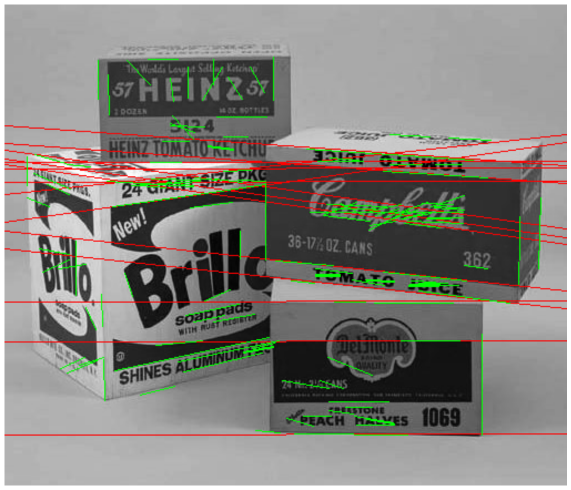
img05.png



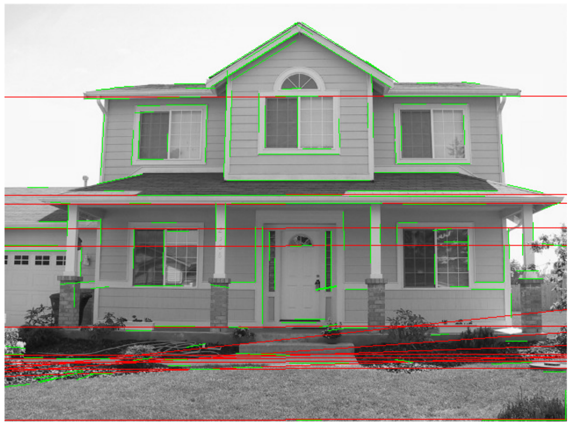
myHoughTransform(img05)

Q3.4 Finding Lines

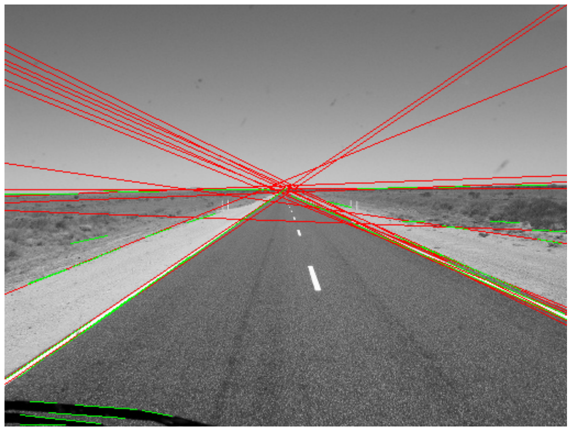
img06.png


myHoughLines(img06, nLines=15)

img07.png


myHoughLines(img07, nLines=15)

img04.png


myHoughLines(img04, nLines=15)

Experiments

Q4.1 Write-up

As indicated by sample images above, I found that the effectiveness of my code varied between images.

For example, I believe the `myImageFilter` function was effective on all images. We created and applied a Gaussian smoothing filter with $\sigma = 2$ to the input image—which we expect to have a 'blurring' effect on the resulting image, since noise/finer details are reduced—and the resulting images did indeed appear to be a blurry version of their respective input images.

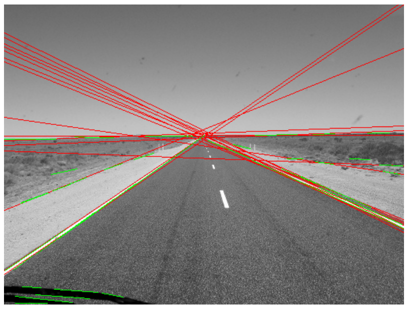
The `myEdgeFilter` and `myHoughTransform` functions also seemed to work reasonably well on all images. Note that the latter function, however, is the most computationally expensive. Its worst-case time complexity is $O(nmt)$ (where n is the number of rows in the image, m is the number of columns, and t is the length of `thetaScale`), since we have to consider and compute ρ for every pixel in the image.

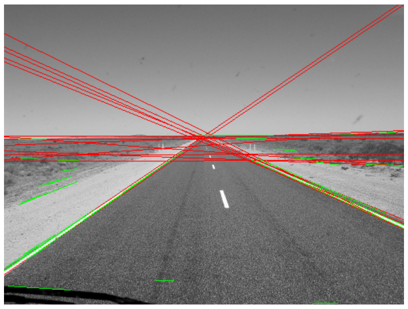
However, I found the greatest discrepancy in effectiveness to be revealed when running `myHoughLines`. Of course, since we can vary the maximum number of lines with the `nLines` input argument, not all possible lines found by the function would be visible. Still, the function seemed to perform notably worse on `img04` (pictured above), for example, as a few red lines from my implementation were drawn in unexpected areas. On the other hand, images like `img06` and `img07` (also above) seemed to produce great results, with lines appearing as expected: aligned with those produced by OpenCV's `HoughLinesP` function.

I suspected that the discrepancy stemmed mainly from the value of σ in `myEdgeFilter`, as higher values of σ smooth/blur its input image more, and smaller values preserve more noise and fine detail. Perhaps the loss of said detail caused `myHoughLines` to have more trouble detecting edges (and, therefore, drawing the unexpected resulting lines). I know that changing the values of `myHoughTransform`'s `rhoScale` and `thetaScale` arguments would toggle the distance and angular resolution of the Hough accumulator, respectively. However, since it takes in the image threshold, which is created using the output of `myEdgeFilter`, I thought changing just the value of σ in the latter would be both simple and impactful.

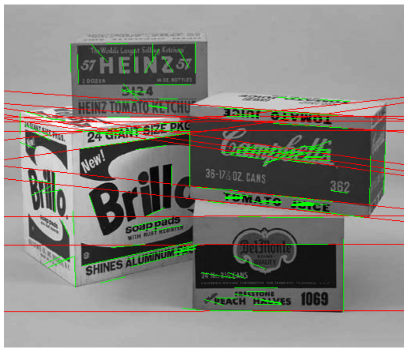
As indicated by the images below, toggling the value of σ did not necessarily improve results for all images. Decreasing the value of sigma (preserving more detail in the input image) revealed new lines that were detected by `myHoughLines` and `HoughLinesP`. Hence, some stray lines were eliminated from the `img04` output, but some lines I thought should be included were lost in `img06`'s. Clearly, finding an optimal set of parameters is non-trivial, as the input images have widely varying clarity, content/subject matter, texture, etc.

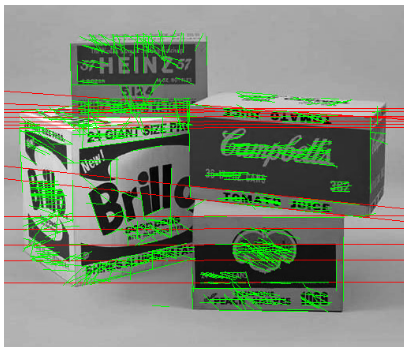
`img04.png`


`myHoughLines`, $\sigma = 1$


`myHoughLines`, $\sigma = 1.5$

`img06.png`


`myHoughLines`, $\sigma = 1$


`myHoughLines`, $\sigma = 0.5$