Programming Assignment 1 Write-up

Ethan Wong-Chassine

January 27, 2025

Q2. Theory Questions: Hough Line Parametrization

1.

The line equation:

$$\rho = x\cos\theta + y\sin\theta$$

Let $\vec{u} = (\cos\theta, \sin\theta)$ be the unit vector in the direction of angle θ Applying the dot product of two vectors formula to $\vec{V} = (x, y)$ and $\vec{u} = (\cos\theta, \sin\theta)$:

$$x\cos\theta + y\sin\theta = \vec{V} \cdot \vec{u}$$

Using the formula for the dot product of two vectors

$$\vec{V} \cdot \vec{u} = \|\vec{V}\|\|\vec{u}\|\cos(\alpha)$$

where α is the angle between the two vectors. Here, $\alpha = \theta - \phi$, where ϕ is the angle of the vector $\vec{V}$. Since $\|\vec{u}\| = 1$:

$$x\cos\theta + y\sin\theta = \|\vec{V}\|\cos(\theta - \phi)$$

The magnitude $\|\vec{V}\|$ is:

$$\|\vec{V}\| = \sqrt{x^2 + y^2}$$

Therefore:

$$p = \|\vec{V}\|\cos(\theta - \phi)$$

$$p = \sqrt{x^2 + y^2}\cos(\theta - \phi)$$

where $\|\vec{V}\| = \sqrt{x^2 + y^2}$ is the magnitude of $\vec{V}$, and ϕ is the phase angle of $\vec{V}$, defined by $\tan(\phi) = \frac{y}{x}$.

Amplitude:

$$A = \sqrt{x^2 + y^2}$$

Phase:

$$\phi = \tan^{-1}\left(\frac{y}{x}\right)$$

Therefore, each image point (x, y) in the Image-space maps to a sinusoidal curve in the Hough space, where amplitude is determined by the distance from the origin and phase is determined by the angle of the point from the x-axis.

2.

The slope-intercept form (m, c) does not work for vertical lines (where $m \to \infty$). This also does not allow for bounds to be set on the accumulator array. Parametrizing (ρ, θ) avoids the issue where all lines are represented uniformly. From $x \cos\theta + y \sin\theta = \rho$, m and c are:

$$m = -\cot\theta = -\frac{\cos\theta}{\sin\theta}, \quad c = \frac{\rho}{\sin\theta}.$$

3.

The maximum value of ρ occurs when the line passes through the farthest diagonal corners of the image:

$$\rho_{\max} = \sqrt{W^2 + H^2}.$$

The range of θ: $\theta \in [0, 2\pi)$

4.

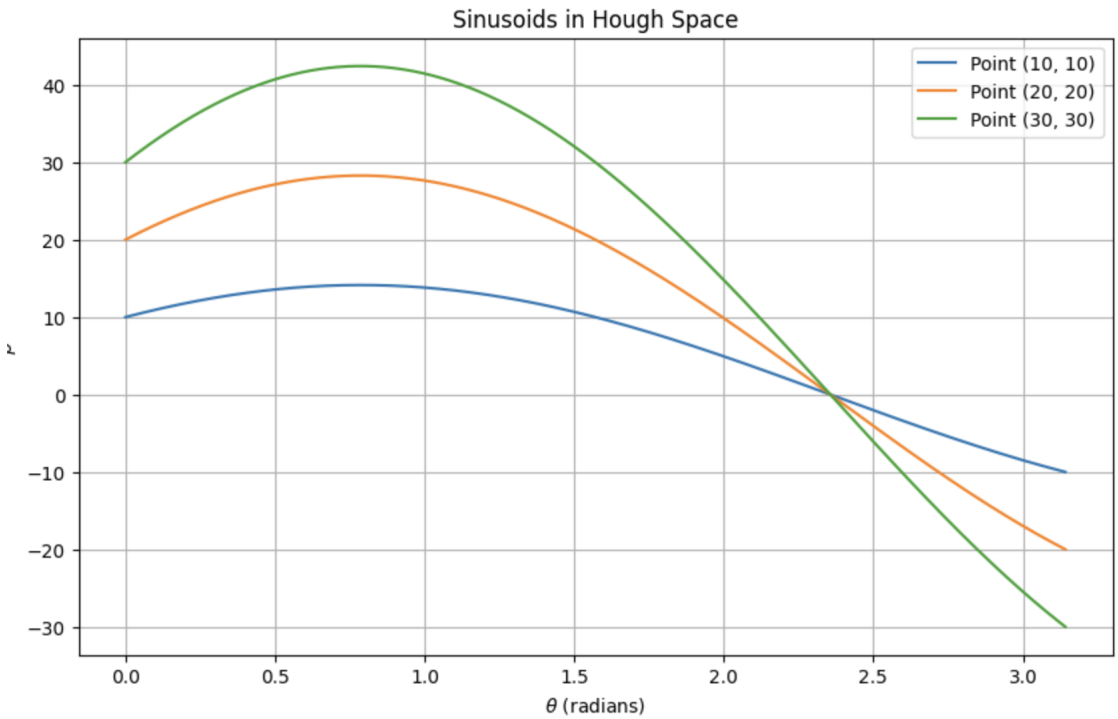


Figure 1: m=1, c=0

Q4. Experiments: Write-up

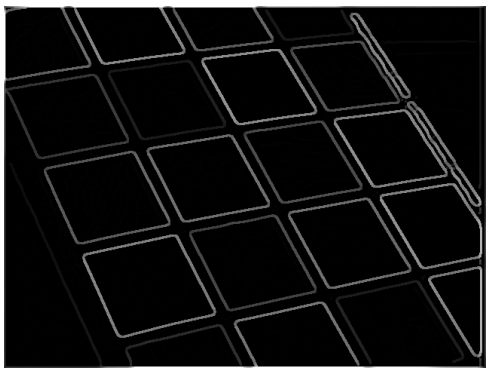
myImageFilter

I found the runtime of this function to be a major issue, which led to a strange solution – using np.strides. Early on in the assignment, I kept running into issues when applying kernels iteratively. I was able to find resources (linked inline in code) that create sliding windows of the image to which I can apply the kernel in moderately fast runtime. There

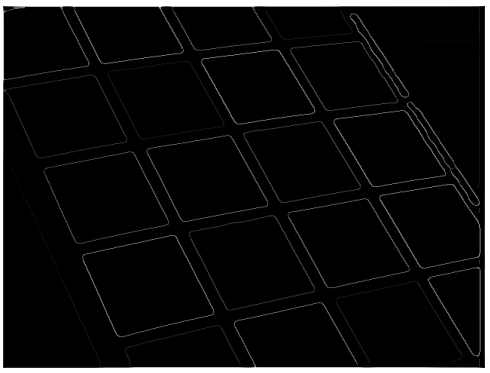
were some warnings on the np documentation, but I did not encounter any obvious issues, and the solution was too valuable to pass over.

myEdgeFilter

This function probably gave me the most trouble and most likely contributes to the runtime the most. I started by using Gaussian blur as instructed but the NMS was probably the most difficult step - I tried using kernels seen in the commented code from Office Hours but regressed to my nested loop solution that manually gathers neighbor data iteratively. Note that this solution does work quickly. I then dilated the image but also eroded the enlarged edges. Eroding decreased my runtime drastically most likely because the edge lines were thinned which was less data to handle (see images)



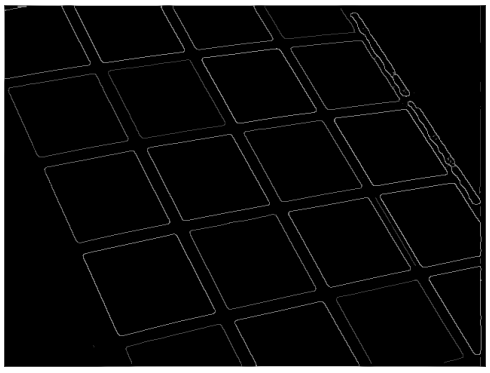
((a)) Dilated image



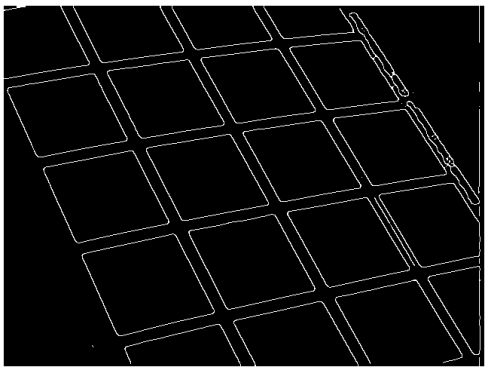
((b)) Eroded image

Figure 2

I then manually applied another threshold operation to clean smaller lines that might have been detected. This was an essential step in the edge filter function as lots of noise would appear otherwise in the output image.



((a)) Edge Detection image



((b)) Threshold image

Figure 3: Results of myEdgeFilter

myHoughTransform

This section also gave me a lot of trouble – I could not get the correct points/curves to appear on the graph in the correct places with the given ranges for rho and theta. I

originally set rhoScale to [-max distance, max distance] and thetaScale to $[0, \pi]$. I realized that I was not selecting the correct rho_index when I was iterating through the edge points. This is where $np.argmin$ was used to find the correct value within the $rhoScale$ array, and also where I ignore negative values.

Figure 4: img01 Hough Space

myHoughLines

I followed the normalization and dilation from the instructions and then went on to select the $nLines$ through sorting and argmax. The top $nLines$ can be found from the Hough accumulator (the highest peaks in the accumulator). The indexes of the lines are then found and returned.

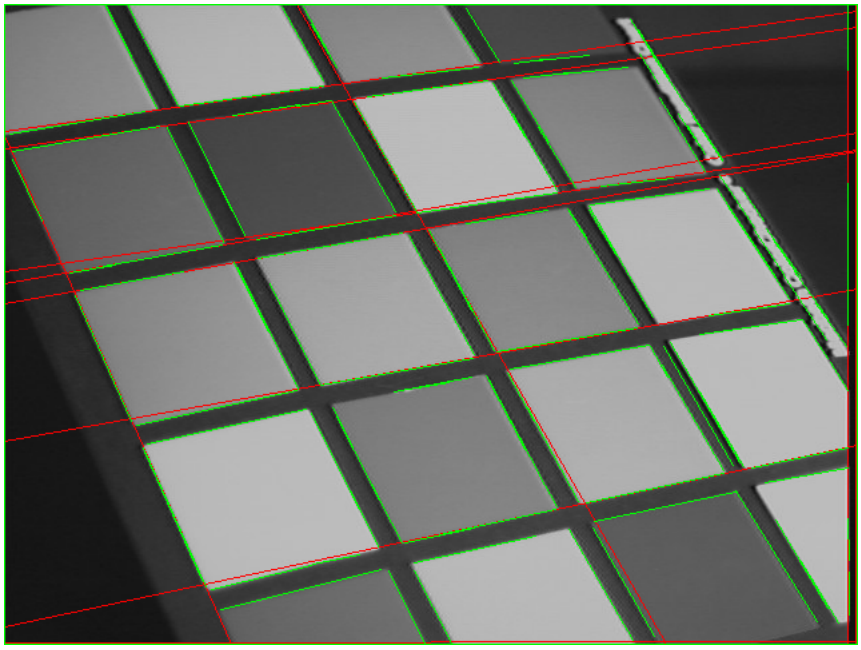


Figure 5: img01 Detected Lines

Changing Parameters

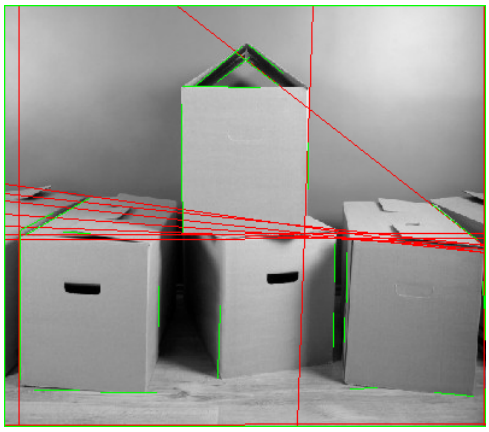
- Sigma: I found this to increase the Gaussian blur of the images the greater the degree of the constant - this would lead to less defined edges and less accurate lines depending on the degree. Increasing sigma is a good idea for images with larger clear lines like img04. It might also be good for img06 because the default parameters select the text's most significant lines, ignoring the desired straight lines on the box edges. Increasing sigma might not be a good idea for img01 because it blurs the squares that are less differentiable from the background, and therefore are not detected as edges. Overall increasing is better, but not too much as to totally blur edges.

((a)) Default Sigma ((b)) High Sigma Value (sigma=10)

Figure 6: Changing Sigma on img06

- Threshold: this variable is the cutoff for the edge intensity values. The higher the value, the stronger (more defined) edges will be "selected", the lower, the more

values will be selected. Lowering the threshold also sometimes increased runtime for me because that meant the collection of valid edges in the edge and threshold images were much larger/more complex. Decreasing the threshold parameter did not have a large impact on my images because of my manual threshold filter in *myEdgeFilter* but increasing it helped detect the more defined lines within the images without adjusting the sigma. In general, higher thresholds are better, to a degree.

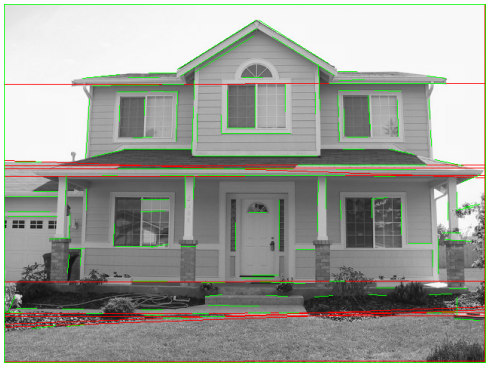


((a)) Default Threshold

((b)) High Threshold (threshold=0.3)

Figure 7: Changing Threshold on img05

- rhoRes: Defines the resolution when observing the Hough space when discretizing the Hough image to in throughlines. Smaller values increase the frequency of observation of the Hough space and, therefore, increase computational complexity + time while decreasing has the inverse effect. The effect on the image is
- thetaRes: Similar to rhoRes, lowering this value would increase the computational complexity (and time). This specific parameter adjusts the angles observed when translating Hough space to a discrete space when looking for edges in the image. Finetuning these two were less straight forward but the increase in thetaRes allowed more discrete angles of lines to be created in the Image-space from the Hough space. Generally, higher values are better but can hugely increase runtime.



((a)) Default thetaRes, rhoRes

((b)) Res * 2

Figure 8: Changes rhoRes and thetaRes on img07

- nLines: It is obvious that this just detects more lines but also note that this should be increased with rhoRes and thetaRes because the increase in resolution will translate to more lines being found when converting from Hough to Image space.