Answers to questions in
Lab 2: Edge detection & Hough transform

Name: _____Cheng-Hsun Chang_____ Program: _____

Instructions: Complete the lab according to the instructions in the notes and respond to the questions stated below. Keep the answers short and focus on what is essential. Illustrate with figures only when explicitly requested.

Good luck!

Question 1: What do you expect the results to look like and why? Compare the size of *dxtools* with the size of *tools*. Why are these sizes different?

Answers:

dxtools highlights vertical edges by emphasizing intensity changes along rows, while dytools highlights horizontal edges by emphasizing intensity changes along columns. These gradients reveal areas of significant change, such as edges, and suppress regions of constant intensity.

The sizes of dxtools and dytools are smaller than tools because the valid mode in the convolution operation only computes values for regions where the kernel fully overlaps the image. For a $k \times k$ kernel, the output size is reduced to $(M - k + 1) \times (N - k + 1)$, where M and N are the original image's dimensions.

For a Sobel operator with $k = 3$, the output size is $(M - 2) \times (N - 2)$.

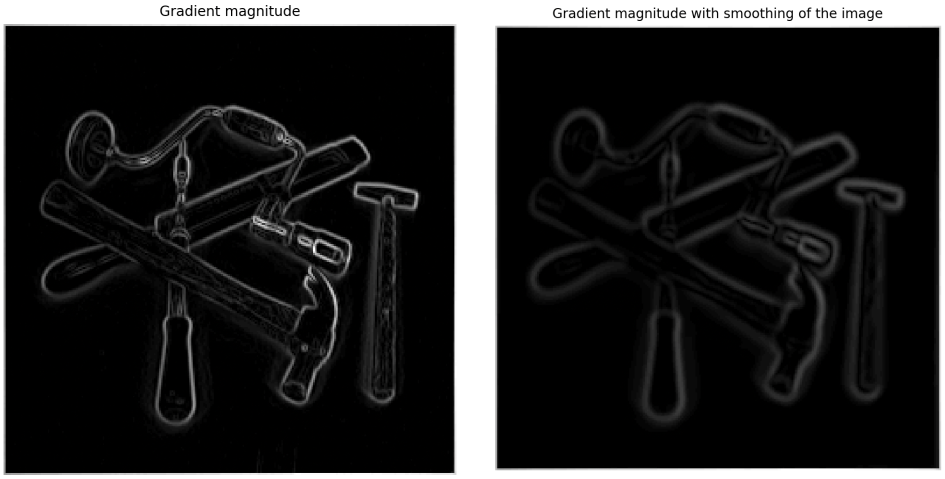
Question 2: Is it easy to find a threshold that results in thin edges? Explain why or why not!

Answers:
Finding a single threshold that results in thin edges is not straightforward. The difficulty arises because gradient magnitudes vary significantly across the image, depending on the intensity transitions at different regions.

High thresholds effectively highlight strong edges but may miss weaker edges, while low thresholds capture weaker edges but can introduce noise or thicker edges due to small gradient fluctuations.

Question 3: Does smoothing the image help to find edges?

Answers:



Yes, smoothing the image helps to find edges by reducing noise and suppressing small intensity variations that could produce spurious gradients.

Applying Gaussian smoothing before computing the gradient magnitude makes the edges more consistent and reduces noise. This makes it easier to apply thresholds that highlight significant edges while avoiding noise artifacts. However, excessive smoothing can blur fine details, causing some edges to disappear entirely, so the degree of smoothing must be carefully balanced.

Question 4: What can you observe? Provide explanation based on the generated images.

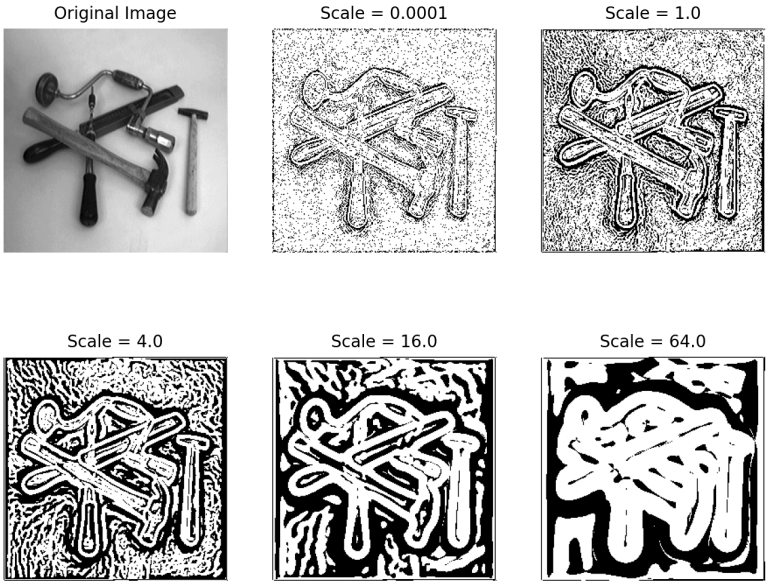
Answers:

1. At small scales, fine details and noise dominate the image.
2. As the scale increases, noise is reduced, and significant edges like the roof and windows become clearer.
3. At very large scales, fine details are suppressed, but significant edges become distorted and lose sharpness due to excessive smoothing.

This demonstrates the trade-off between capturing detail and emphasizing broader structures, with larger scales risking edge distortion.

Question 5: Assemble the results of the experiment above into an illustrative collage with the *subplot* command. Which are your observations and conclusions?

Answers:



1. At very small scales, the result captures fine details but is highly sensitive to noise, leading to cluttered edges.
2. As the scale increases, noise is suppressed, and the detection focuses more on significant edges, producing clearer and smoother results.
3. At very large scales, fine details are lost, and the detected edges become coarser and distorted.

The sign condition helps to refine edge detection by ensuring that only regions meeting the third-order derivative condition (e.g., changes in concavity) are emphasized, reducing false-positive edges and noise. This illustrates the importance of scale selection in balancing detail and abstraction for robust edge detection.

Question 6: How can you use the response from *Lvv* to detect edges, and how can you improve the result by using *Lvvv*?

Answers:

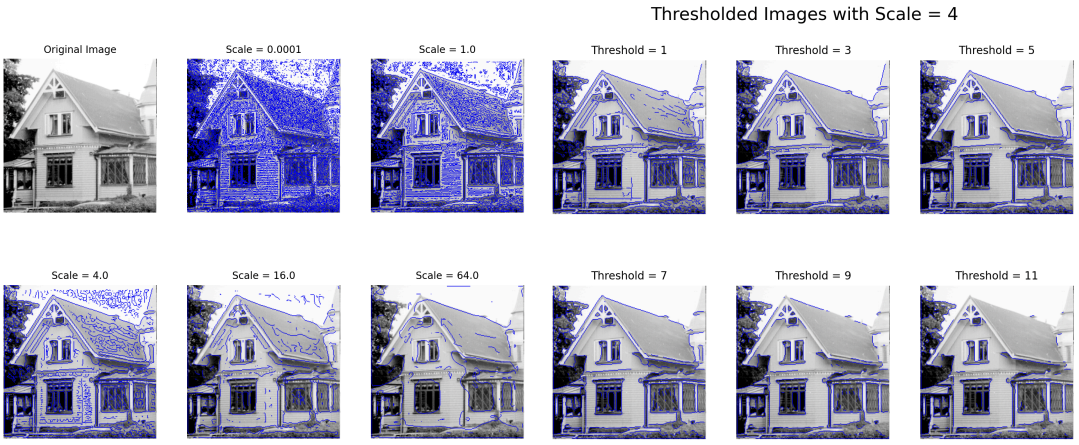
The response from *Lvv* detects edges by identifying zero-crossings in the second derivative, highlighting areas of rapid intensity changes. However, *Lvv* alone can lead to false detections caused by noise or insignificant structures.

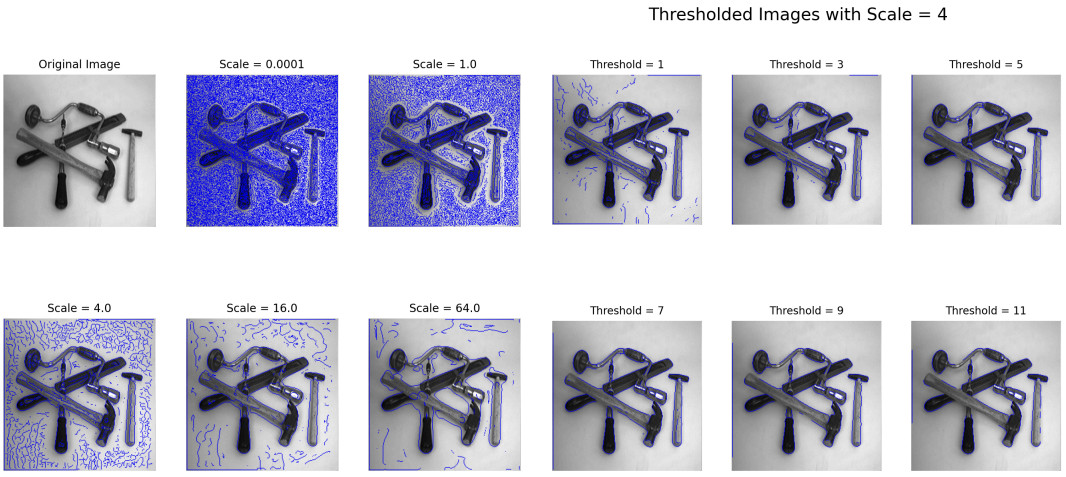
By incorporating *Lvvv*, the edge detection process is refined, as the third-order derivative condition ensures that only edges corresponding to meaningful changes in curvature are retained. This suppresses spurious edges and enhances the detection of significant structures, especially at larger scales.

The combination of *Lvv* and *Lvvv* provides a more robust and accurate edge detection method after filtering out noise and focusing on essential features.

Question 7: Present your best results obtained with *extractedge* for *house* and *tools*.

Answers:





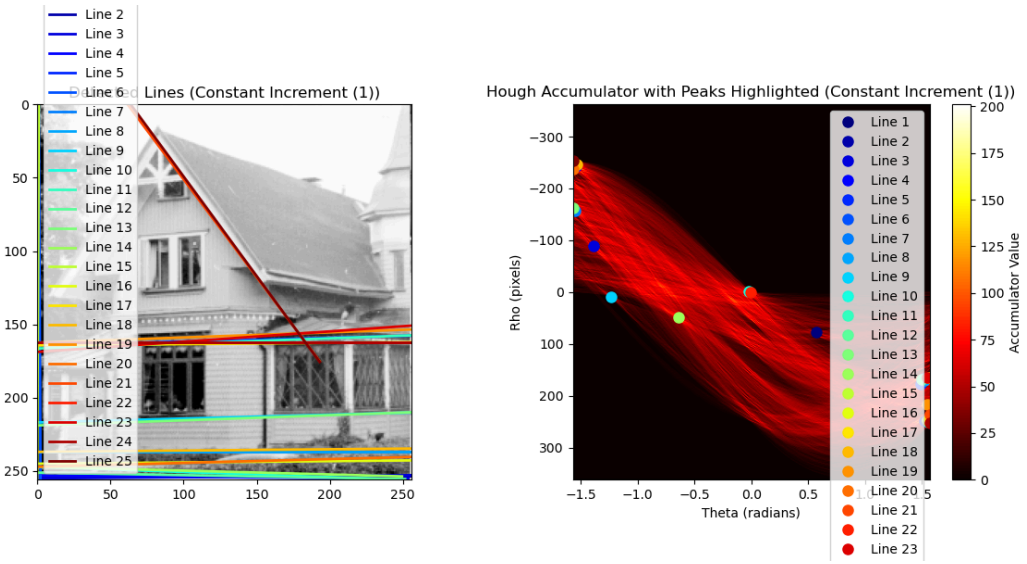
Best results

House with scale=4 and threshold=7

Tools with scale=4 and threshold=7

Question 8: Identify the correspondences between the strongest peaks in the accumulator and line segments in the output image. Doing so convince yourself that the implementation is correct. Summarize the results of in one or more figures.

Answers:

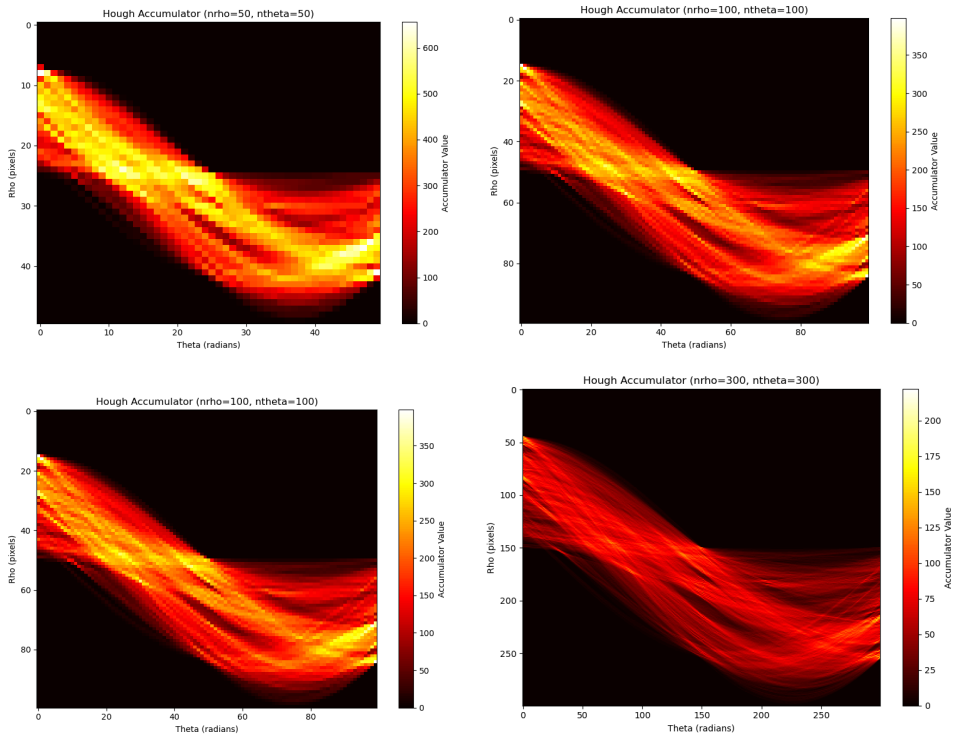


Each strong peak corresponds exactly to an edge feature in the image.
The algorithm is correct in detecting straight-line features.

Question 9: How do the results and computational time depend on the number of cells in the accumulator?

Answers:
When the number of accumulator cells is small, the resolution is insufficient, resulting in multiple possible straight lines clustered in the same cell. When the number of cells is too high, the accumulator space is subdivided too finely and the contribution of each straight line is spread over multiple cells.

Peak detection requires finding local maxima in the accumulator space, and the computational complexity is proportional to nrho×ntheta




```
nrho=50, ntheta=50, time=1.75s
nrho=100, ntheta=100, time=3.49s
nrho=180, ntheta=180, time=6.27s
nrho=300, ntheta=300, time=10.36s
```

Question 10: How do you propose to do this? Try out a function that you would suggest and see if it improves the results. Does it?

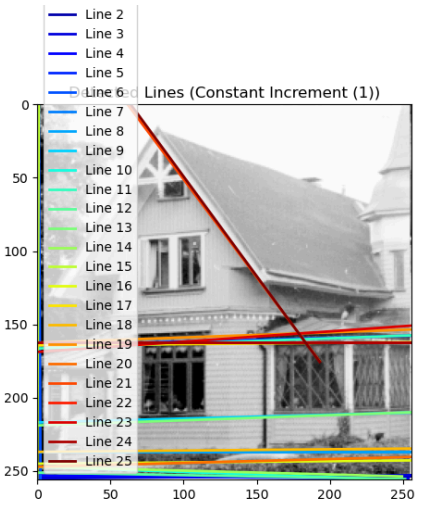
```python
def linear_increment(grad_value):
    return grad_value

def squared_increment(grad_value):
    return grad_value ** 2

def log_increment(grad_value):
    return np.log(1 + grad_value)
```

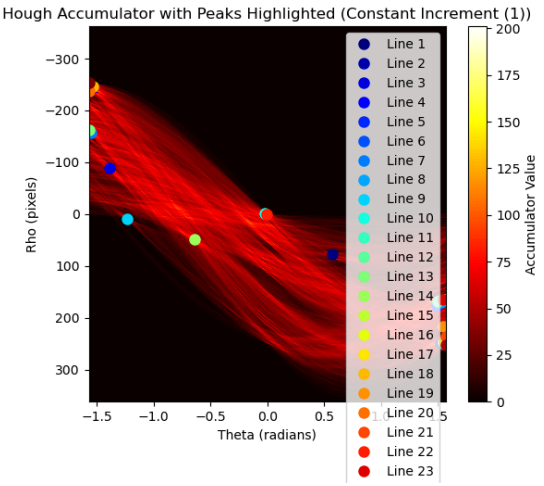
Answers:
Constant Increment: Treats all edge points equally, with smooth cumulative results, but may miss weaker edges; less sensitive to line segments of stronger edges.

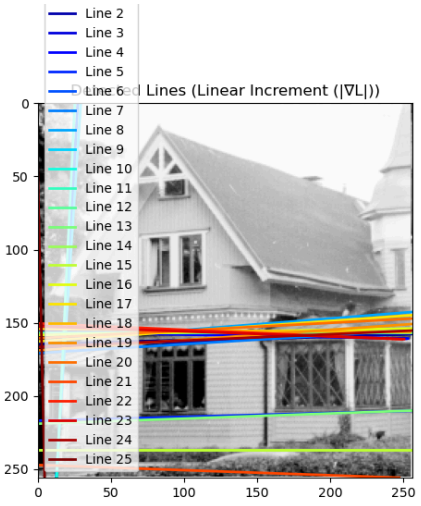
Linear increment: More emphasis on edge points with strong gradient magnitudes, more focused cumulative results, effective in extracting distinct straight line segments; reduced contribution to weak edges.

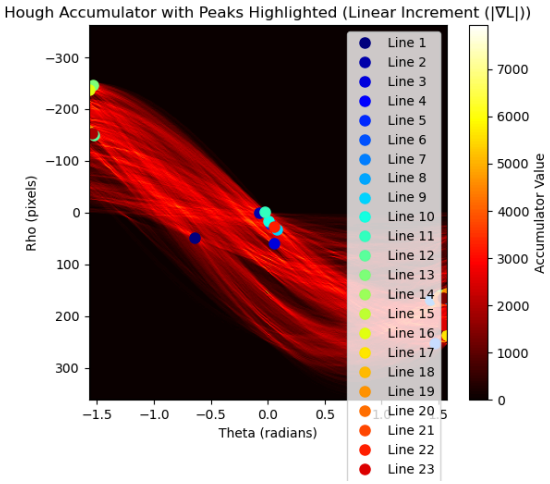
Squared increment: The effect of strong gradient magnitudes is further amplified, with effects similar to those of the linear increment, but with a greater bias toward detecting strong edge straight lines; may ignore weak edges more.

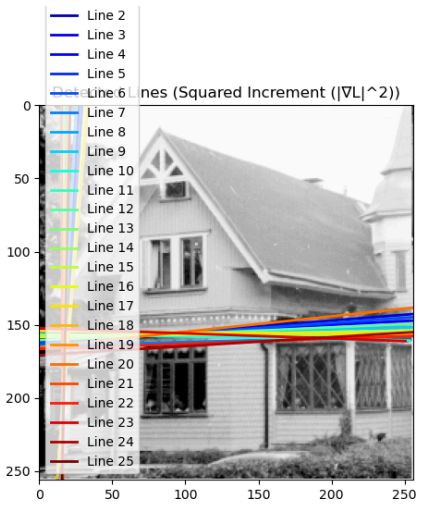
Logarithmic increment: Balances the contribution of strong and weak gradient points, suitable for scenes where the image contains a greater mix of strong and weak edges; the accumulator has a wider response range and the results are easily adjustable.



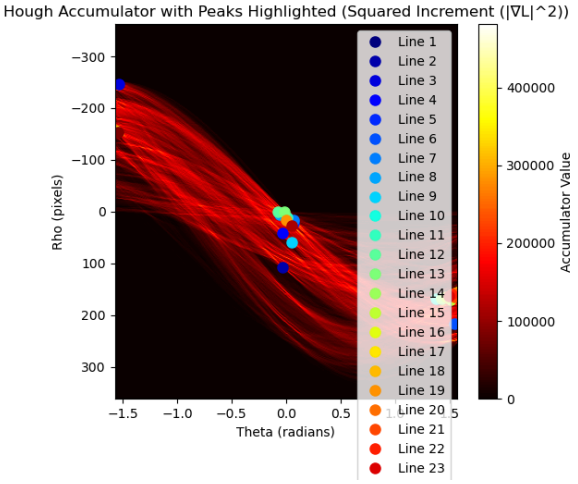




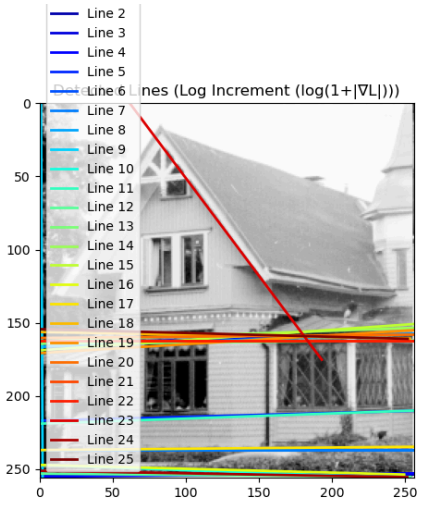




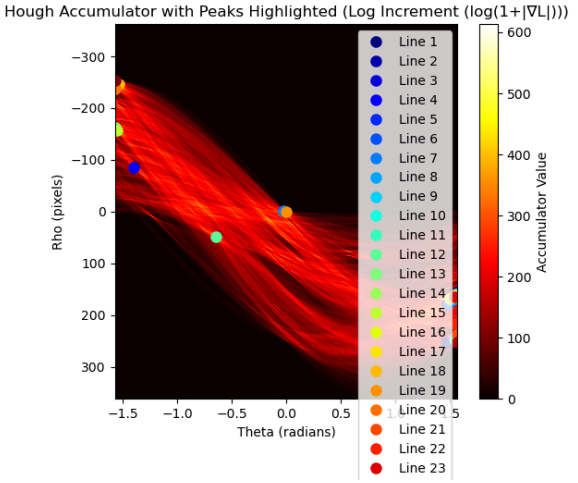
Detected Lines (Squared Increment (|∇L|^2))



Hough Accumulator with Peaks Highlighted (Squared Increment (|∇L|^2))



Detected Lines (Log Increment (log(1+|∇L|)))



Hough Accumulator with Peaks Highlighted (Log Increment (log(1+|∇L|)))