Week 4: Lab 3: Hough Transform

Q1:

Canny edge detection: Firstly, Gaussian filtering is used to reduce image noise, then Sobel operator is applied to calculate the gradient, and edges are determined and optimized by non-maximum suppression and double threshold detection.

Hough transform: The edge points are transformed into the parameter space (θ-ρ space) **w = x * cos(θ) + y * sin(θ)**, and the number of lines corresponding to each parameter is counted in the accumulator array. Significant lines are identified by detecting peaks in the accumulator array.

Line Detection: The peaks extracted from the Hough transform correspond to lines in the image, and finally these lines are plotted on the original image according to their parameters.

Algorithm

Canny edge detection:

Firstly, Canny edge detector is used to identify the edges of cluttera2.jpg image. The Canny algorithm effectively finds the location of edges in the image, which is the basis for Hough transform to identify straight lines.

Set the Hough transform parameters:

```
tested_angles = np.linspace(-np.pi / 2, np.pi / 2, 360, endpoint=False)
```

Next, an array of **360** values is defined to represent the Angle range (-90 to 90 degrees) that will be used during the Hough transform.

Perform the Hough transform:

```
h, theta, d = hough_line(img_edges, theta=tested_angles)
```

Compute the Hough transform of the edge image **img_edges** using the **hough_line** function. The function returns an array of accumulators **h**, and the corresponding arrays of angles **theta** and distances **d**. Each element in the accumulator array h corresponds to the number of edge points on a certain line in the image.

Hough Transform peak detection:

```
accum, angles, dists = hough_line_peaks(h,theta,d,threshold=1,num_peaks=10)
```

The **hough_line_peaks** function is used to detect peaks from the accumulator array h of the Hough transform. These peaks represent significant lines in the image. The function sets a minimum threshold of **threshold=1** and a maximum number of peaks of **num_peaks=10**.

Compute the visual bounds:

```
angle_step = 0.5 * np.diff(theta).mean()
d_step = 0.5 * np.diff(d).mean()
bounds = [np.rad2deg(theta[0] - angle_step),
          np.rad2deg(theta[-1] + angle_step),
          d[-1] + d_step, d[0] - d_step]
```

To better visualize the peaks, the algorithm computes the display boundary of the Hough space.

Draw the detected line:

```
for i in range(0,len(angles)):
    (x0, y0) = dists[i] * np.array([np.cos(angles[i]), np.sin(angles[i])])
    ax[2].axline((x0, y0),slope=np.tan(angles[i] + np.pi/2), color="red", linewidth=3)
```

The algorithm traverses each line detected by the Hough transform. The polar coordinates **(dists[i], angles[i])** of each line are converted to **Cartesian coordinates (x, y)** using the formulas **x = r * cos(θ) and y = r * sin(θ)**. **dists[i]** for each line represents the distance and **angles[i]** represents the Angle.

Finally, these lines are drawn on the original image, usually distinguished by different colors or line types.

Q2:

Increase the number of spikes(num_peaks=40):

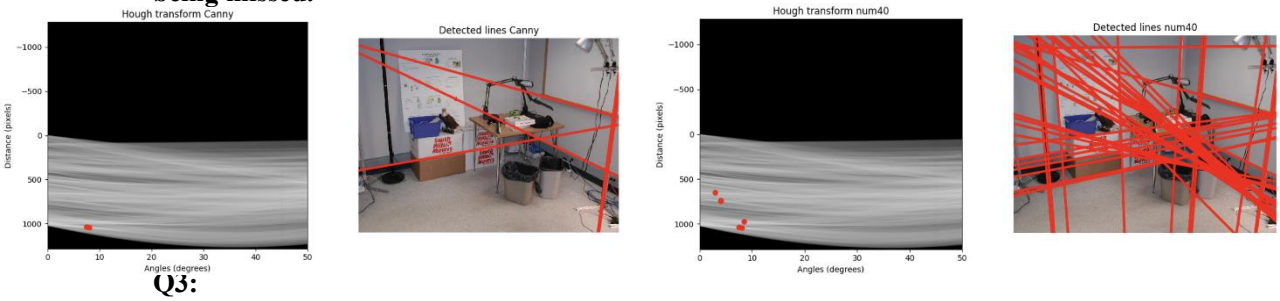
More lines are detected: Increasing the number of spikes means that the algorithm tries to identify more lines from the accumulator array of the Hough transform. This is useful for finding less salient or finer lines in an image.

May contain non-ideal lines: At the same time, this may also cause the algorithm to identify some less important or even wrong lines, especially in images with **more noise or complex lines**.

Reduce the number of spikes(num_peaks=5):

Only the most significant lines are detected: Reducing the number of spikes causes the algorithm to identify only a few of the most significant lines. This helps to focus on the most important line features in the image and reduce interference.

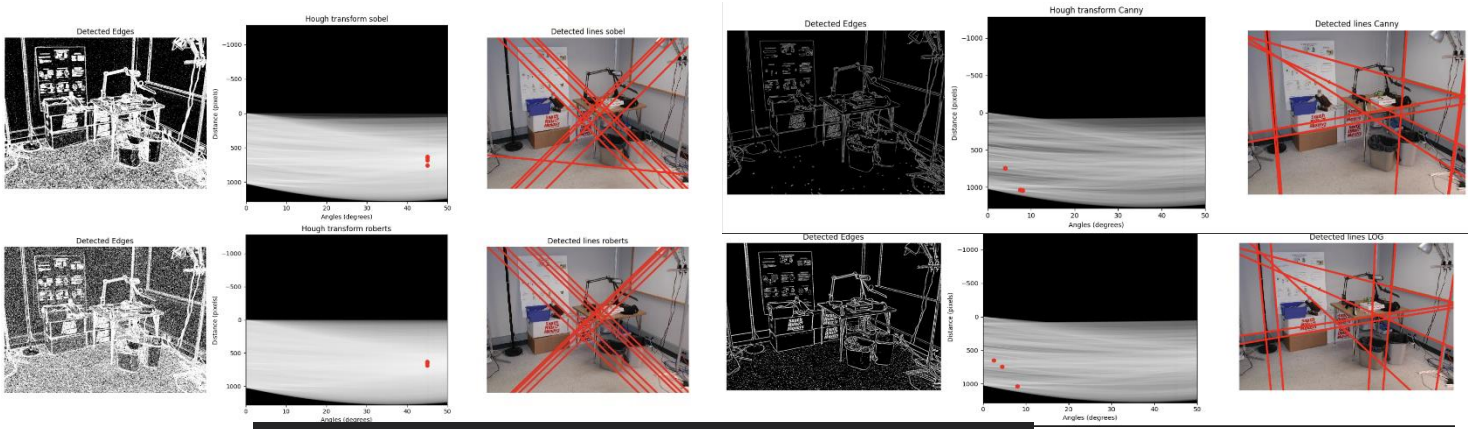
Important lines can be missed: However, this can also lead to some subtle but i**mportant lines being missed.**



Q3:

After using the **Sobel** and **Roberts** edge detectors, it was found that they performed poorly, one reason being that the image **contained significant noise**, to which these operators are quite sensitive. Upon using the **LoG (Laplacian of Gaussian)** detector, it was observed that this operator could represent edges well. One reason for this is that LoG is particularly suited for noisy images as it combines Gaussian smoothing with second-order derivative edge detection. In comparison with the Canny operator, it was found that LoG achieved better **peak separation** in the Hough transform. In the detection of **actual lines**, the LoG marked **more edges** than Canny.

```
#LOG
img_smooth = gaussian(img_grey,sigma=1)
img_edges2=  laplace(img_smooth)
thresh = 0.01
img_edges = img_edges > thresh
```



Q4:

```
lines = probabilistic_hough_line(img_edges, threshold=10, line_length=5, line_gap=3)
```

In **PHT**, the algorithm first randomly selects edge points in the image as **starting points** of potential line segments. It then searches for other edge points near this point to try to **end the segment**. This step consists of **checking whether a line exists between these two points**, whether this line passes through other edge points, and whether the minimum length requirement is satisfied. In addition, the PHT algorithm allows a **certain gap between two edge points**, helping to connect those edges that visually belong to the same line but are interrupted due to noise or image quality issues. Whenever a possible line segment is found, the algorithm accumulates **votes** for this segment in the parameter space. If a segment receives more votes than a set **threshold**, it is considered significant and selected as the final segment. By setting different thresholds, we can **adjust the strictness of the algorithm to detect line segments**, and then control the **quality** of the detected line segments. Finally, the algorithm will end the segment finding and verification process based on certain termination criteria, such as reaching a sufficient number of segments or performing a **predetermined number of iterations.**