COMP2005 Laboratory Sheet 6: Edge Detection and Hough Transform

1. Edge Detection

OpenCV provides many types of edge detection algorithms and functions, including the ones discussed in the lectures (e.g., Sobel, Roberts). Each edge detection method provides slightly different results, and the key to successful edge detection is a **good choice of parameters** rather than the method choice.

Note: The images should be converted to grayscale before performing any of the following edge detection methods. Make sure to consider all the parameters of each edge detection function: what it is and its effect on the resulting image.

1st Derivative: Sobel

Sobel edge detection works by identifying areas with significant intensity changes as edges. This is done by calculating the gradient in both the x and y directions individually and then combining them to calculate the magnitude of the gradient at that specific pixel. Pixels with a high magnitude are then classified as an edge.

- Read in the *Opera house.jpeg* image from Moodle.
- By using the *Sobel* function, tweak the parameters and show the results of applying the Sobel operator in the x direction, y direction and the combination of both x and y. (Hint: Use the formula in the lecture to compute the gradient and remember to normalise the result.)
- Compare the differences in the resulting image. (Note: The resulting image will have both positive and negative values. Use NumPy's *absolute* function when displaying the image to ensure only positive values remain.)

2nd Derivative: Laplacian of Gaussian (LoG)

Laplacian of Gaussian (LoG) edge detection involves two steps: Gaussian smoothing and Laplacian filtering. Gaussian smoothing reduces the noise in the image, and the Laplacian filter identifies areas with significant intensity changes. LoG then identifies the zero crossings (value changes from positive to negative and vice versa) in the resulting image as edges.

- Using the *Opera house.jpeg* image, apply Gaussian blur and then perform LoG edge detection on the blurred image using *Laplacian*.
- Play around with the Gaussian blur and *Laplacian* parameters and compare the effects on the resulting image.

Canny

Canny edge detection can be said to be an extension of Sobel edge detection. The first step is to reduce the noise in the image by using Gaussian smoothing before calculating the gradient using Sobel filters in both the x and y. To obtain thin edges, non-maximum suppression is performed by saving only the local maximums of the gradient in the gradient direction.

Lastly, hysteresis thresholding selects the edges by using two threshold values (min and max). Edges above the max threshold are edges, edges below the min threshold are non-edges, and for edges in between the two thresholds, it is dependent on their connectivity to an edge/non-edge.

- Use the *Canny* function on the *Opera house.jpeg* image, and by varying the parameters, extract:
    a. the structure of the opera house roof
    b. the waves on the water
- Identify parameters in the *Canny* function that will:
    a. extract the boundaries of the cars (but not their shadows) from the *Highway.jpeg* image.
    b. identify all the flowers in the *Tulip.jpeg* image.
- Were you able to achieve all the tasks? If not, why?

Compare the results of the three operators (Sobel, LoG and Canny) and identify the differences. Which edge detection method do you think performed the best and why?

2. Hough Transform

Hough Transform is a technique used to detect shapes (e.g., lines, circles) which can be represented by mathematical equations. This is done by converting the image space into a parameter space and conducting a “voting” procedure on the points which are part of the edges in the image.

- Read the *sudoku.jpg* image from Moodle and convert it to binary. You may either apply thresholding or Canny edge detection on the image to achieve this. However, Canny would be better to highlight the edges which are areas of interest.
- Apply the Hough Line Transform on the edge-detected image using *HoughLines.*
- To find the optimal parameters, we must first be able to see the lines detected. The *HoughLines* function returns an array of (ρ, θ) values, which we will use to draw the lines on the image. You may read more about this here: https://docs.opencv.org/4.x/d6/d10/tutorial_py_houghlines.html.
- Once the coordinate of the detected line is calculated, draw the line on a **copy of the original coloured image** using *line*.
- With the lines detected now visible, adjust the *threshold* parameter in *HoughLines* to get all the grid lines of the sudoku puzzle. Were all the grid lines detected? If not, why?

3. Expected Results

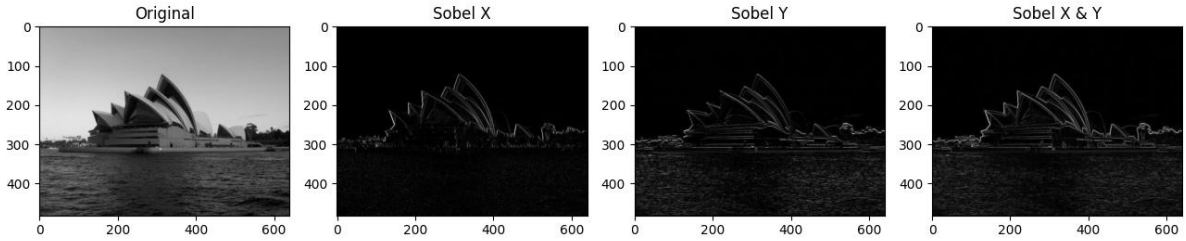

Figure 1: Sobel Edge Detection

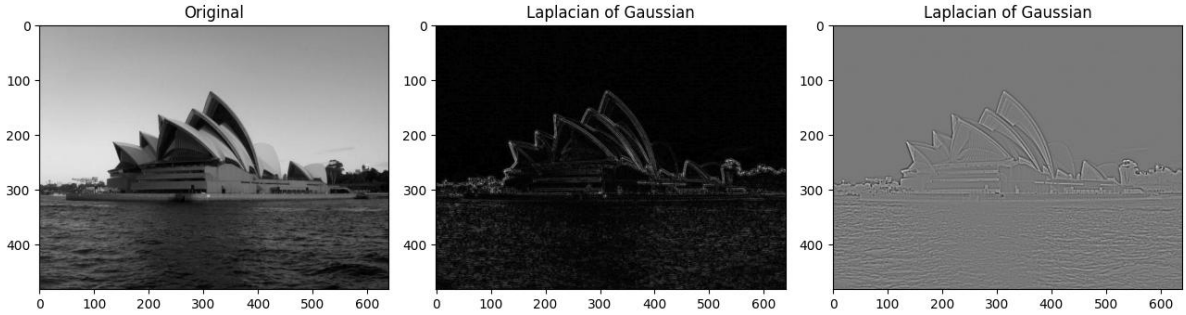

Figure 2: Laplacian of Gaussian

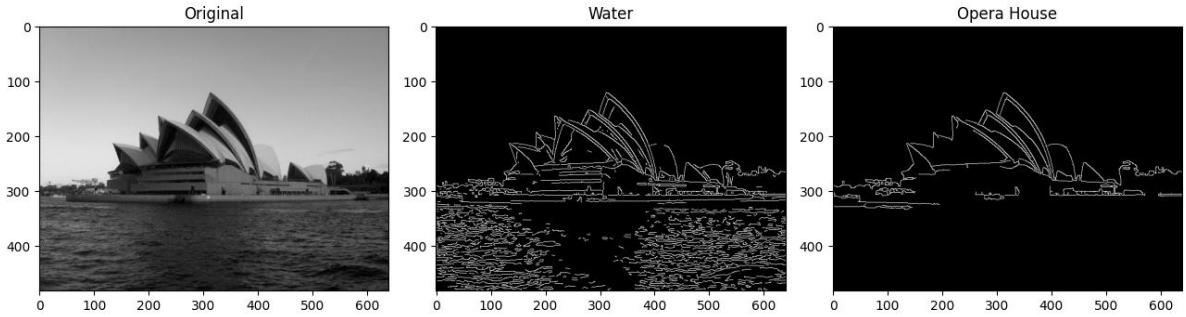

Figure 3: Canny Edge Detection

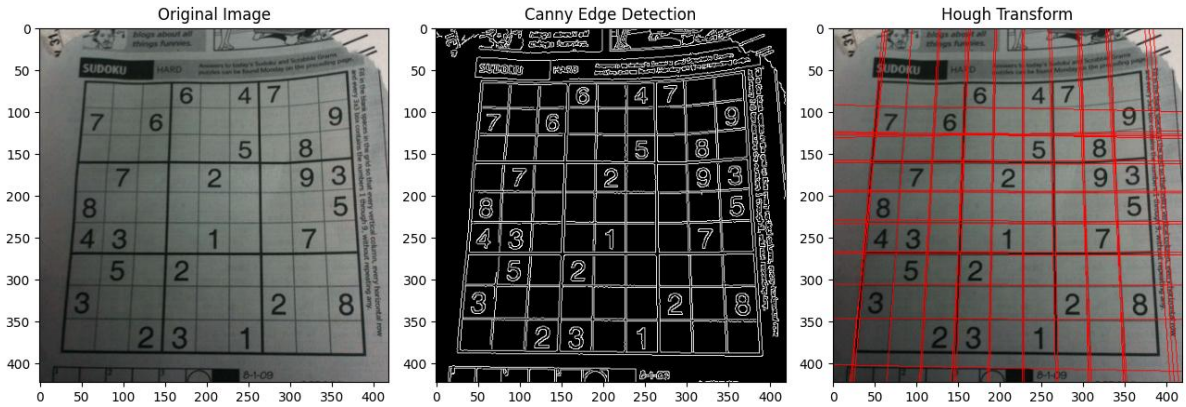

Figure 4: Hough Line Transform