CS 415: Homework 2

Due on 10/16/24 at 11:59pm

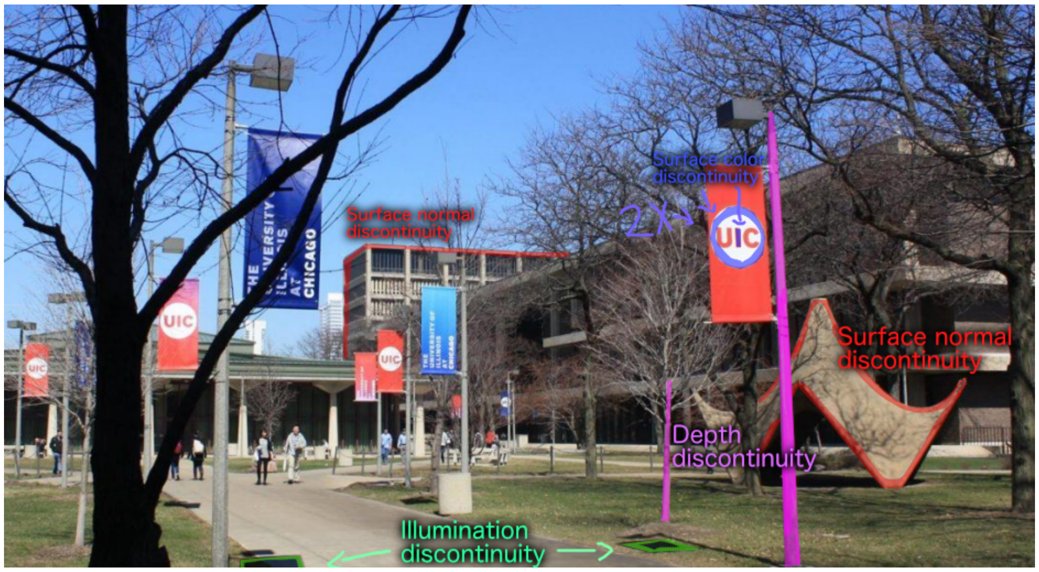
Professor Tang 11-12:15pm

Ryan Magdaleno
rmagd2@uic.edu

Problem 1

1. **What is an edge in an image?**
   A significant local change in the image intensities, a transition between two distinct regions in an image. Edges make up shapes, regions, and other features of objects in a image, they are critical to image processing tasks.

2. **We learned four different factors that could cause edges in an image. For each of these factors, please find and label two segments of edges caused by it in the image below.**


Surface color discontinuity
2X
Surface normal discontinuity
Surface normal discontinuity
Depth discontinuity
Illumination discontinuity

3. **What is the relation between the direction of image gradients and that of edges.**
   The gradient is a vector/direction for a position in an image where there is the steepest increase in intensity. The edge direction is the boundary between two regions of different intensities. Their relation is that the edge direction is perpendicular to the gradient direction, that is, running along the boundary where the change in intensity happens.

Problem 2

Given a 5x5 image below, calculate the gradient of each pixel in the 3x3 central region via finite difference. For the same region, calculate the gradient magnitude and gradient direction for each pixel. You can keep the square root and any trigonometric functions in the answer.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 2 & 1 \\ 1 & 2 & 2 & 1 & 1 \\ 1 & 2 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Pixel $(2, 2)$:

$$\begin{aligned} G_x &= [3,2] - F[2,2] = 0 - 1 = -1 \\ G_y &= [2,3] - F[2,2] = 2 - 1 = 1 \\ \nabla f &= [-1, 1] \\ ||\nabla f|| &= \sqrt{(-1)^2 + (1)^2} = \sqrt{2} \\ \theta &= \arctan\left(\frac{1}{-1}\right) = -\frac{\pi}{4} \end{aligned}$$

Pixel $(2, 3)$:

$$\begin{aligned} G_x &= [3,3] - F[2,3] = 2 - 2 = 0 \\ G_y &= [2,4] - F[2,3] = 2 - 2 = 0 \\ \nabla f &= [0, 0] \\ ||\nabla f|| &= \sqrt{(0)^2 + (0)^2} = 0 \\ \theta &= \arctan\left(\frac{0}{0}\right) = 0 \end{aligned}$$

Pixel $(2, 4)$:

$$\begin{aligned} G_x &= [3,4] - F[2,4] = 1 - 2 = -1 \\ G_y &= [2,5] - F[2,4] = 1 - 2 = -1 \\ \nabla f &= [-1, -1] \\ ||\nabla f|| &= \sqrt{(-1)^2 + (-1)^2} = \sqrt{2} \\ \theta &= \arctan\left(\frac{-1}{-1}\right) = \frac{\pi}{4} \end{aligned}$$

Pixel $(3, 2)$:

$$\begin{aligned} G_x &= [4,2] - F[3,2] = 1 - 0 = 1 \\ G_y &= [3,3] - F[3,2] = 2 - 0 = 2 \\ \nabla f &= [1, 2] \\ ||\nabla f|| &= \sqrt{(1)^2 + (2)^2} = \sqrt{5} \\ \theta &= \arctan\left(\frac{2}{1}\right) = \arctan(2) \end{aligned}$$

Problem 2 continued on next page...

Pixel $(3,3)$:

$$\begin{aligned}G_x &= [4,3] - F[3,3] = 1 - 2 = -1\\ G_y &= [3,4] - F[3,3] = 1 - 2 = -1\\ \nabla f &= [-1,-1]\\ ||\nabla f|| &= \sqrt{(-1)^2 + (-1)^2} = \sqrt{2}\\ \theta &= \arctan\left(\frac{-1}{-1}\right) = \frac{\pi}{4}\end{aligned}$$

Pixel $(3,4)$:

$$\begin{aligned}G_x &= [4,4] - F[3,4] = 0 - 1 = -1\\ G_y &= [3,5] - F[3,4] = 1 - 1 = 0\\ \nabla f &= [-1,0]\\ ||\nabla f|| &= \sqrt{(-1)^2 + (0)^2} = 1\\ \theta &= \arctan\left(\frac{0}{-1}\right) = 0\end{aligned}$$

Pixel $(4,2)$:

$$\begin{aligned}G_x &= [5,2] - F[4,2] = 1 - 2 = -1\\ G_y &= [4,3] - F[4,2] = 1 - 2 = -1\\ \nabla f &= [-1,-1]\\ ||\nabla f|| &= \sqrt{(-1)^2 + (-1)^2} = \sqrt{2}\\ \theta &= \arctan\left(\frac{-1}{-1}\right) = \frac{\pi}{4}\end{aligned}$$

Pixel $(4,3)$:

$$\begin{aligned}G_x &= [5,3] - F[4,3] = 1 - 1 = 0\\ G_y &= [4,4] - F[4,3] = 0 - 1 = -1\\ \nabla f &= [0,-1]\\ ||\nabla f|| &= \sqrt{(0)^2 + (-1)^2} = 1\\ \theta &= \arctan\left(\frac{-1}{0}\right) = 0\end{aligned}$$

Pixel $(4,4)$:

$$\begin{aligned}G_x &= [5,4] - F[4,4] = 1 - 0 = 1\\ G_y &= [4,5] - F[4,4] = 1 - 0 = 1\\ \nabla f &= [1,1]\\ ||\nabla f|| &= \sqrt{(1)^2 + (1)^2} = \sqrt{2}\\ \theta &= \arctan\left(\frac{1}{1}\right) = \frac{\pi}{4}\end{aligned}$$

Problem 3

Why is the normal form of a line a better choice than the slope intercept form in Hough transform for line detection?

Solution

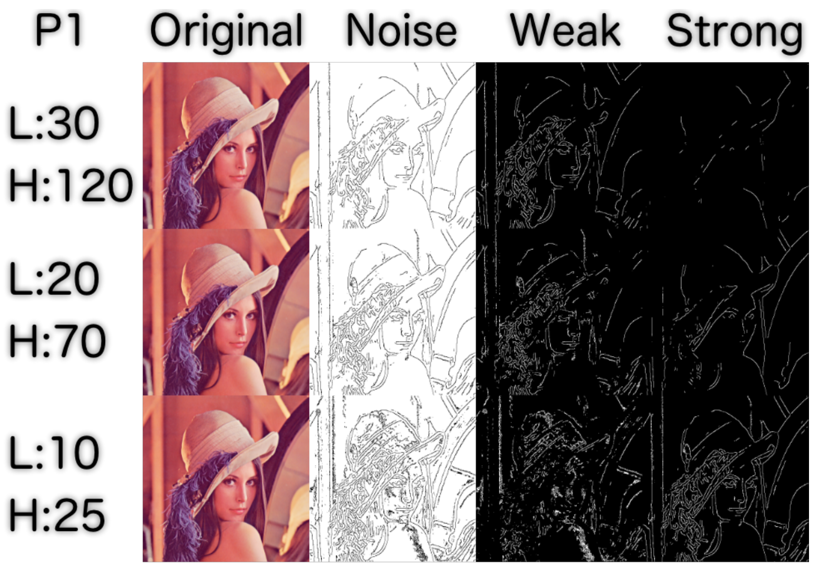
The slope intercept form of a line makes use of slopes and this can cause for infinite slopes when dealing with vertical lines, essentially making it very difficult for us to discretize in the Hough Transform accumulator space.

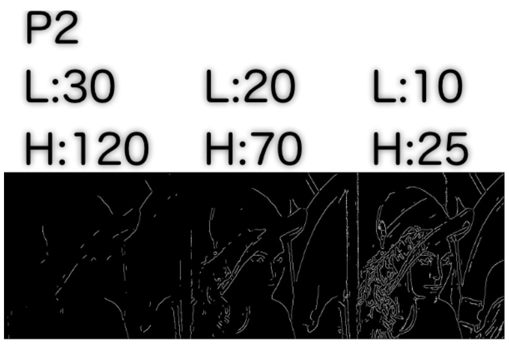
Using normal form however you make use of ρ and θ, their ranges are bounded unlike slope intercept form. θ ranges from 0 to 360°, this let's us get all orientations of a line. ρ ranges from 0 to some ρ_{max}, a finite accumulator array size. ρ's bounding allows us to represent the distance of the line from the origin or some point.

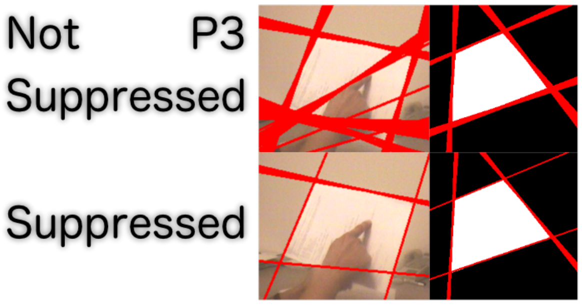
Overall the normal form of a line is better because it can handle all lines, better discretization in Hough space, and avoids infinite slopes for vertical lines, also bounded parameter ranges.

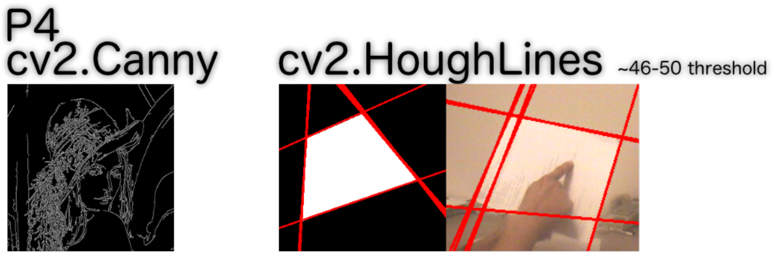
Problem 4: Program Output + Source Code

Code is below image and in bundled main.py, my threshold values can be found in the code.









```python
#  main.py
''' ------------------------------------------------------------------------------------
 >> Assignment details and provided code are created and owned by Wang Tei.
 >> University of Illinois at Chicago - CS 415, Fall 2024
 >> ------------------------------------------------------------------------------------
 >> File   :: main.py
 >> Course :: CS 415 (42844), FA24
 >> Author :: Ryan Magdaleno (rmagd2)
 >> System :: Windows 10 w/ Python 3.11.3
 - -                             - -
 >> File overview ::
 >> This program implements the Canny Edge Detector algorithm. Smoothing is done using
 >> cv2's gaussian blur function. We calculate the gradient magnitudes and directions
 >> of the image using our own applied sobel x/y kernels. We then supress any non-
 >> maximums. We then do hysteresis thresholding and aquire the strong, weak, and noise
 >> maps. Finally, we link the weak edges to strong edges and create our canny edge
 >> detected output image. This program also solves some given problems found in
 >> the bundled pdf, specifically involving the Hough Transforms algorithm.
 - -          References         - -
 >> https://en.wikipedia.org/wiki/Canny_edge_detector
 >> https://en.wikipedia.org/wiki/Hough_transform
 - -                             - -
 >> Usage:
 >> ret_str := main("img/lena.png", test_thresholds[0], Options.P2)
 >> Meaning: Running problem 2 config on "img/lena.png" and use threshold 0 found in
 >> tuple test_thresholds.
 - -                             - -
 >> Extra Note:
 >> Store the input images in a folder called "img", else remove from paths here.
------------------------------------------------------------------------------------ '''
# Module Imports ::
from typing import Optional, Tuple
from collections import namedtuple
from math import cos, sin, sqrt
from enum import Enum
import numpy as np
import cv2 as cv

# A named tuple for use in later functions :: - -                                   - -
Thresholds = namedtuple("Thresholds", ("low", "high"))

# Options for func main :: - -                                                      - -
class Options(Enum):
    P1 = 1
    P2 = 2
    P3 = 3
    P4 = 4
```

```python
# Compute the image's gradient magnitudes and directions using sobel x/y kernels ::    - -
def compute_gradient(img: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
    # Apply sobel filters for x and y directions:
    sobel_x = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
    sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
    gradient_x = cv.filter2D(img, cv.CV_64F, sobel_x)
    gradient_y = cv.filter2D(img, cv.CV_64F, sobel_y)

    # Get the image gradient magnitudes and directions:
    magnitude = np.sqrt(gradient_x**2 + gradient_y**2)
    magnitude = np.uint8(np.clip(magnitude, 0.0, 255.0))
    direction = np.arctan2(gradient_y, gradient_x)
    direction *= 180 / np.pi
    return magnitude, direction

# Using the gradient and magnitude matrices, supress non maxima :: - -                - -
def non_maximum_suppression(magnitude: np.ndarray, direction: np.ndarray) -> np.ndarray:
    height, width = magnitude.shape
    res = np.zeros(magnitude.shape)
    direction[direction < 0] += 180  # (-180, 180) -> (0, 180)

    for y in range(1, height - 1):
        for x in range(1, width - 1):
            current_direction = direction[y, x]
            current_magnitude = magnitude[y, x]
            if (0 <= current_direction < 22.5) or (157.5 <= current_direction <= 180):
                p = magnitude[y, x - 1]
                r = magnitude[y, x + 1]

            elif 22.5 <= current_direction < 67.5:
                p = magnitude[y + 1, x + 1]
                r = magnitude[y - 1, x - 1]

            elif 67.5 <= current_direction < 112.5:
                p = magnitude[y - 1, x]
                r = magnitude[y + 1, x]

            else:
                p = magnitude[y - 1, x + 1]
                r = magnitude[y + 1, x - 1]

            if current_magnitude >= p and current_magnitude >= r:
                res[y, x] = current_magnitude
    return res
```

Problem 4 continued on next page. . .

```python
# Generate the weak and strong edges based on given thresholds :: - -                       - -
def hysteresis_thresholding(img: np.ndarray, thresholds: Thresholds) \
    -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    # Apply hysteresis thresholding onto the nms matrix:
    strong_edges = np.zeros_like(img, dtype=np.uint8)
    weak_edges   = np.zeros_like(img, dtype=np.uint8)
    noise        = np.zeros_like(img, dtype=np.uint8)

    # Threshold using conditionals on np.ndarrays:
    strong_edges[img >= thresholds.high] = 255
    weak_edges[(img >= thresholds.low) & (img < thresholds.high)] = 255
    noise[img < thresholds.low] = 255

    return strong_edges, weak_edges, noise

# Connect weak edges to strong edges by check each pixel around said weak pixel ::     - -
def edge_linking(strong_edges: np.ndarray, weak_edges: np.ndarray) -> np.ndarray:
    # 8 possible directions for connectivity:
    directions = ((1, 1), (1, 0), (1, -1), (0, 1), (0, -1), (-1, 1), (-1, 0), (-1, -1))

    # Iterate over weak edges and connect them to strong edges:
    ret = np.copy(strong_edges)
    height, width = weak_edges.shape
    for y in range(1, height - 1):
        for x in range(1, width - 1):
            if weak_edges[y, x] > 0: # Found a weak edge at weak[y, x]
                for dy, dx in directions:
                    # Case - Found a strong edge in current direction:
                    if strong_edges[y + dy, x + dx] > 0:
                        ret[y, x] = 255
                        break
    return ret
```

Problem 4 continued on next page. . .

```python
# Given function that performs Hough Tranformation on a edge map :: - -           - -
def hough_transform(edge_map: np.ndarray) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    theta_values = np.deg2rad(np.arange(-90.0, 90.0))
    height, width = edge_map.shape
    diagonal_length = int(round(sqrt(width * width + height * height)))
    rho_values = np.linspace(-diagonal_length, diagonal_length, diagonal_length * 2 + 1)

    accumulator = np.zeros((len(rho_values), len(theta_values)), dtype=int)
    y_coordinates, x_coordinates = np.nonzero(edge_map)

    for edge_idx in range(len(x_coordinates)):
        x = x_coordinates[edge_idx]
        y = y_coordinates[edge_idx]
        for theta_idx in range(len(theta_values)):
            theta = theta_values[theta_idx]
            rho = int(round(x * np.cos(theta) + y * np.sin(theta)))
            accumulator[rho + diagonal_length, theta_idx] += 1

    print(f"\n  {edge_idx + 1} out of {len(x_coordinates)} edges have voted.")
    return accumulator, theta_values, rho_values

# Supress values below a given threshold, helps remove excess hough lines :: - -     - -
def suppress_non_maxima(accumulator: np.ndarray, threshold: int = 46): # paper thres: 50
    ret = np.copy(accumulator)
    height, width = accumulator.shape

    for y in range(1, height - 1):
        for x in range(1, width - 1):
            if accumulator[y, x] < threshold:
                ret[y, x] = 0
                continue
            # Check local 3x3 grid for current pixel:
            region = accumulator[y - 1:y + 2, x - 1:x + 2]
            if accumulator[y, x] < np.max(region):
                ret[y, x] = 0

    return ret
```

Problem 4 continued on next page. . .

```
# P3: Help remove lines by suppressing detections that are not the local maxima ::     - -
def problem_three() -> Optional[str]:
    target_img = cv.imread("img/paper.bmp")
    gray_img   = cv.cvtColor(target_img, cv.COLOR_BGR2GRAY)
    edge_map   = cv.Canny(gray_img, 70, 150)

    accumulator, theta_values, rho_values = hough_transform(edge_map)
    lines = np.argwhere(suppress_non_maxima(accumulator) > 46) # paper thres: 50

    _, width = gray_img.shape
    for line in lines:
        rho = rho_values[line[0]]
        theta = theta_values[line[1]]
        slope = -np.cos(theta)/np.sin(theta)
        intercept = rho/np.sin(theta)
        x1, x2 = 0, width
        y1 = int(slope*x1 + intercept)
        y2 = int(slope*x2 + intercept)
        cv.line(target_img, (x1, y1), (x2, y2), (0, 0, 255), 2)

    cv.imwrite("img/output-v2-cv2-Hough-paper.bmp", target_img)
    return ""

# P4: Use cv2's Canny and HoughLine functions :: - -                                    - -
def problem_four(thres: Thresholds) -> Optional[str]:
    lena_img = cv.imread("img/lena.png", cv.IMREAD_GRAYSCALE)
    lena_img = cv.Canny(lena_img, thres.low, thres.high)

    shape_img = cv.imread("img/shape.bmp")
    gray = cv.cvtColor(shape_img, cv.COLOR_BGR2GRAY)
    edges = cv.Canny(gray, thres.low, thres.high)

    for line in cv.HoughLines(edges, 2, np.pi / 180, 80): # paper params: 2, pi/180, 90
        rho, theta = line[0]
        a = np.cos(theta)
        b = np.sin(theta)
        x_0, y_0 = a * rho, b * rho
        x_1 = int(x_0 + 2000 * (-b)) # 2000: make sure full line, extend if needed.
        y_1 = int(y_0 + 2000 * (a))
        x_2 = int(x_0 - 2000 * (-b))
        y_2 = int(y_0 - 2000 * (a))
        cv.line(shape_img, (x_1, y_1), (x_2, y_2), (0, 0, 255), 2)

    cv.imwrite("img/output-cv2-Canny-lena.png",  lena_img)
    cv.imwrite("img/output-cv2-Hough-shape.bmp", shape_img)
    return ""
```

Problem 4 continued on next page. . .

```
# Using the opt paramter, run specific functions and problems :: - -                                - -
def main(file_name: str, thres: Thresholds, opt: int) -> Optional[str]:
    # Check if problem 3/4:
    if opt == Options.P3:
        return problem_three()
    if opt == Options.P4:
        return problem_four(thres)

    if (img := cv.imread(file_name, cv.IMREAD_GRAYSCALE)) is None:
        return "Image load failed."
    print(f"\n  Thres   : ({thres.low=}, {thres.high=})"
          f"\n  Stage 0: Load image \"{file_name}\" into memory.")

    # Smooth the image with a gaussian blur:
    img = cv.GaussianBlur(img, ksize=(9, 9), sigmaX=3)
    print("  Stage 1: Gaussian blur applied.")

    # Get the image gradient magnitudes and directions:
    g_magnitude, g_direction = compute_gradient(img)
    print("  Stage 2: Image gradient magnitudes and directions calculated.")

    # Apply non-maximum suppression:
    img = non_maximum_suppression(g_magnitude, g_direction)
    print("  Stage 3: Non-maximum suppression applied.")

    # Apply Hysteresis Thresholding onto nms'd image:
    strong_edges, weak_edges, noise = hysteresis_thresholding(img, thres)
    print("  Stage 4: Applied Hysteresis Thresholding onto nms'd image.")

    if opt == Options.P1:
        cv.imwrite("img/output-strong-edges.png", strong_edges)
        cv.imwrite("img/output-weak-edges.png", weak_edges)
        cv.imwrite("img/output-noise.png", noise)
        return ""

    # Link the weak edges to strong edges:
    img = edge_linking(strong_edges, weak_edges)
    print("  Stage 5: Link the weak edges to strong edges")
    thres_str = f"{str(thres.low).zfill(3)}-{str(thres.high).zfill(3)}"
    cv.imwrite(f"img/output-linked-edges-{thres_str}.png", img)
    return ""

# Program entrypoint :: - -                                                                         - -
if __name__ == "__main__":
    test_thresholds = (Thresholds(30, 120), Thresholds(20, 70), Thresholds(10, 25))
    if ret_str := main("img/lena.png", test_thresholds[0], Options.P2):
        print(f"  Error :: {ret_str}")
```