**Project 1 - Canny Edge Detector**

**Project group members: 1. Aniket Bote (N12824308)**

**2. Sindhu Harish (N19806874)**

**Instructions to run the code:**

1. Download and install python 3.8.6.
2. Clone the project and navigate to the project directory named ‘canny-edge-detector’.
3. Create a virtual environment (OPTIONAL).
   * pip install virtualenv
   * virtualenv venv
4. Install the required libraries after activating the virtual environment.
   * pip install -r requirements.txt
5. Run the below command
   * python canny\_edge\_detector.py --input\_folder input --output\_folder output
6. The input\_folder is from where the input images are read, and output\_folder is where the output images are going to get saved.
7. In the given project structure input\_folder is named as input and output\_folder is named as output respectively.

**Source Code:**

1. **The main file where the code execution starts:- canny\_edge\_detector.py**

# Import the required libraries

import argparse

import glob

import os

import shutil

import cv2

from gaussian\_smoothing import perform\_gaussian\_smoothing

from gradient\_operation import perform\_gradient\_operation

from non\_maxima\_suppression import perform\_non\_maxima\_suppression

from thresholding import perform\_thresholding

parser = argparse.ArgumentParser()

parser.add\_argument(

    '--input\_folder',

    type=str,

    default='input',

    required=False,

    help='input folder with images'

)

parser.add\_argument(

    '--output\_folder',

    type=str,

    default='output',

    required=False,

    help='output folder to save processed images'

)

args = parser.parse\_args()

if os.path.exists(args.output\_folder):

    shutil.rmtree(args.output\_folder)

os.makedirs(args.output\_folder)

print("Reading images from input folder")

images = glob.glob(os.path.join(args.input\_folder, '\*.bmp'))

for image\_name in images:

    output\_image\_name = image\_name.split('\\')[1].split('.bmp')[0]

    img = cv2.imread(image\_name, cv2.IMREAD\_GRAYSCALE)

print("Performing gaussian smoothing for image: " + output\_image\_name)gaussian\_smooth\_image = perform\_gaussian\_smoothing(args, output\_image\_name, img)

print("Performing gradient smoothing for image: " + output\_image\_name)

M, THETA = perform\_gradient\_operation(args, output\_image\_name, gaussian\_smooth\_image)

print("Performing non-maxima suppression for image: " + output\_image\_name)

NMS = perform\_non\_maxima\_suppression(args, output\_image\_name, M, THETA)

print("Performing thresholding for image: " + output\_image\_name, '\n')

T1, T2, T3 = perform\_thresholding(args, output\_image\_name, NMS)

1. **Source Code for Gaussian Smoothing :- gaussian\_smoothing.py**

import os

import cv2

import numpy as np

from utils import Operator, apply\_discrete\_convolution

def perform\_gaussian\_smoothing(args, image\_name, image):

    '''

    Args:

        image : An image to on which smoothing will appear

    Returns:

        smoothened image : Smoothened image

    '''

    # Apply discreet convolution with gaussian mask

    image = apply\_discrete\_convolution(image, Operator.gaussian\_mask)

    # Normalize the image

    image = image / np.sum(Operator.gaussian\_mask)

    #write image into the output folder after normalization

    cv2.imwrite(os.path.join(args.output\_folder, image\_name + '\_gaussian\_smooth\_normalized.bmp'), image)

    # Return the smoothened image

    return image

1. **Source Code for Gradient Operation:- gradient\_operation.py**

import os

import cv2

import numpy as np

from utils import Operator, apply\_discrete\_convolution

def perform\_gradient\_operation(args, image\_name, image):

    '''

    Args:

        image : An image on which gradient operation will happen

    Returns:

        Magnitude : Magnitude of the gradient

        Theta     : Gradient Angle

    '''

    # Compute horizontal gradients

    dfdx = apply\_discrete\_convolution(image, Operator.gx)

    #Copy Image to Output folder after horizontal gradient

    cv2.imwrite(os.path.join(args.output\_folder, image\_name + '\_Gx\_normalized.bmp'), dfdx)

    # Compute vertical gradients

    dfdy = apply\_discrete\_convolution(image, Operator.gy)

    #Copy Image to Output folder after vertical gradient

    cv2.imwrite(os.path.join(args.output\_folder, image\_name + '\_Gy\_normalized.bmp'), dfdy)

    # Compute magnitude of the gradient

    m = np.sqrt(np.square(dfdx) + np.square(dfdy))

    # Normalize gradient magnitude

    m = np.absolute(m) / 3

    #Copy Image to Output folder with gradient magnitude value

    cv2.imwrite(os.path.join(args.output\_folder, image\_name + '\_gradient\_magnitude\_normalized.bmp'), m)

    # Compute gradient angle

    theta = np.degrees(np.arctan2(dfdy, dfdx))

    return m, theta

1. **Source Code for non-maxima-supression:- non\_maxima\_supression.py**

import os

import numpy as np

import cv2

from utils import get\_positive\_angle, Sector

def perform\_non\_maxima\_suppression(args, image\_name, magnitude, gradient\_angle):

    '''

    Args:

        magnitude : Magnitude of the gradient

        gradient\_angle : Gradient angle

    Returns:

        Magnitude : Magnitude array after non-maxima supression

    '''

    # Compute positive angles

    positive\_gradient\_angle = get\_positive\_angle(gradient\_angle)

    # Get magnitude array shape

    m\_arr, n\_arr = magnitude.shape

    # reference pixel location during start of the process

    rpi\_m, rpi\_n = 1,1

    # Build output array

    output\_arr = np.ones((m\_arr , n\_arr)) \* np.nan

    for i in range(m\_arr - 2):

        for j in range(n\_arr - 2):

            # Compute output pixel location for output array

            op\_m, op\_n = i + rpi\_m, j + rpi\_n

            # Get 3 x 3 magnitude slice

            arr\_slice = magnitude[i:i+3, j:j+3]

            # Get 3 x 3 angle slice

            angle\_slice = positive\_gradient\_angle[i:i+3, j:j+3]

# If undefined value at reference pixel in magnitude or angle put zero in output pixel location

if np.isnan(arr\_slice[rpi\_m][rpi\_n]) or np.isnan(angle\_slice[rpi\_m][rpi\_n]):

                output\_arr[op\_m][op\_n] = 0

      else:

                # Get the sector value

    sector = Sector().get\_sector(angle\_slice[rpi\_m][rpi\_n])

      if sector == 0:

# If undefined value at any of sector neighbour put zero in output pixel location

      if np.isnan(arr\_slice[rpi\_m][rpi\_n+1]) or np.isnan(arr\_slice[rpi\_m][rpi\_n-1]):

                output\_arr[op\_m][op\_n] = 0

# If reference pixel is greater than its sector neighbours put reference pixel value at output location

  elif arr\_slice[rpi\_m][rpi\_n] > arr\_slice[rpi\_m][rpi\_n+1] and arr\_slice[rpi\_m][rpi\_n] > arr\_slice[rpi\_m][rpi\_n-1]:

output\_arr[op\_m][op\_n] = arr\_slice[rpi\_m][rpi\_n]

# If reference pixel value is less than its sector neighbours put zero in output pixel location

   else:

        output\_arr[op\_m][op\_n] = 0

   elif sector == 1:

# If undefined value at any of sector neighbour put zero in output pixel location

    if np.isnan(arr\_slice[rpi\_m-1][rpi\_n+1]) or np.isnan(arr\_slice[rpi\_m+1][rpi\_n-1]):

                        output\_arr[op\_m][op\_n] = 0

# If reference pixel is greater than its sector neighbours put reference pixel value at output location

   elif arr\_slice[rpi\_m][rpi\_n] > arr\_slice[rpi\_m-1][rpi\_n+1] and arr\_slice[rpi\_m][rpi\_n] > arr\_slice[rpi\_m+1][rpi\_n-1]:

                        output\_arr[op\_m][op\_n] = arr\_slice[rpi\_m][rpi\_n]

# If reference pixel value is less than its sector neighbours put zero in output pixel location

   else:

                        output\_arr[op\_m][op\_n] = 0

   elif sector == 2:

# If undefined value at any of sector neighbour put zero in output pixel location

if np.isnan(arr\_slice[rpi\_m-1][rpi\_n]) or np.isnan(arr\_slice[rpi\_m+1][rpi\_n]):

                        output\_arr[op\_m][op\_n] = 0

# If reference pixel is greater than its sector neighbours put reference pixel value at output location

   elif arr\_slice[rpi\_m][rpi\_n] > arr\_slice[rpi\_m-1][rpi\_n] and arr\_slice[rpi\_m][rpi\_n] > arr\_slice[rpi\_m+1][rpi\_n]:

                        output\_arr[op\_m][op\_n] = arr\_slice[rpi\_m][rpi\_n]

# If reference pixel value is less than its sector neighbours put zero in output pixel location

    else:

                        output\_arr[op\_m][op\_n] = 0

    elif sector == 3:

# If undefined value at any of sector neighbour put zero in output pixel location

     if np.isnan(arr\_slice[rpi\_m-1][rpi\_n-1]) or np.isnan(arr\_slice[rpi\_m+1][rpi\_n+1]):

                        output\_arr[op\_m][op\_n] = 0

# If reference pixel is greater than its sector neighbours put reference pixel value at output location

  elif arr\_slice[rpi\_m][rpi\_n] > arr\_slice[rpi\_m-1][rpi\_n-1] and arr\_slice[rpi\_m][rpi\_n] > arr\_slice[rpi\_m+1][rpi\_n+1]:

                        output\_arr[op\_m][op\_n] = arr\_slice[rpi\_m][rpi\_n]

# If reference pixel value is less than its sector neighbours put zero in output pixel location

   else:

                        output\_arr[op\_m][op\_n] = 0

# If sector value is other 0,1,2,3 raise an error.(Not going to happen its there for correctness)

   else:

    raise f"Undefined sector: {sector}"

    cv2.imwrite(os.path.join(args.output\_folder, image\_name + '\_non\_maxima\_supression.bmp'), output\_arr)

    return output\_arr

1. **Source Code for Thresholding:- Thresholding.py**

import os

import numpy as np

import cv2

def perform\_thresholding(args, image\_name, image):

    '''

    Args:

        image: Non maxima suppressed

    Returns:

        img1 : Image after applying threshold t1

        img2 : Image after applying threshold t2

        img3 : Image after applying threshold t3

    '''

# Store all the values of image after non- maxima suppression which are greater than zero into array

    image\_arr = image[image>0].ravel()

    # Get 25th percentile of the array

    t1 = np.percentile(image\_arr,25)

    image\_1 = (image > t1).astype("int32")

cv2.imwrite(os.path.join(args.output\_folder, image\_name + f'\_threshold\_t1\_{np.round(t1, 2)}.bmp'), image\_1 \* 255)

# Multiplying the image with 255 for contrast

    # Get 50th percentile of the array

    t2 = np.percentile(image\_arr,50)

    image\_2 = (image > t2).astype("int32")

cv2.imwrite(os.path.join(args.output\_folder, image\_name + f'\_threshold\_t2\_{np.round(t2, 2)}.bmp'), image\_2 \* 255)

    # Get 75th percentile of the array

    t3 = np.percentile(image\_arr,75)

    image\_3 = (image > t3).astype("int32")

cv2.imwrite(os.path.join(args.output\_folder, image\_name + f'\_threshold\_t3\_{np.round(t3, 2)}.bmp'), image\_3 \* 255)

    # Apply threshold to the image and convert it into integer array

    return image\_1, image\_2, image\_3

**6. Common Functions are included as Utility Class :- utility.py**

import numpy as np

# A class to store all operators

class Operator:

    # Prewitt operator for Gx

    gx = np.array([

        [-1,0,1],

        [-1,0,1],

        [-1,0,1]])

    # Prewitt operator for Gy

    gy = np.array([

        [1,1,1],

        [0,0,0],

        [-1,-1,-1]])

    # Gaussian mask

    gaussian\_mask = np.array([

        [1,1,2,2,2,1,1],

        [1,2,2,4,2,2,1],

        [2,2,4,8,4,2,2],

        [2,4,8,16,8,4,2],

        [2,2,4,8,4,2,2],

        [1,2,2,4,2,2,1],

        [1,1,2,2,2,1,1]])

# A class to store sector angle definitions and method to provide sector based on angle

class Sector():

    def \_\_init\_\_(self):

        # Dictionary with {sector: sector range}

        self.sector = {0: [(0, 22.5),(337.5,360),(157.5,202.5)], 1: [(22.5,67.5), (202.5,247.5)], 2:[(67.5,112.5), (247.5, 292.5)], 3:[(112.5, 157.5), (292.5,337.5)]}

    def get\_sector(self, angle):

        for key, val in self.sector.items():

            for l,u in val:

                # check if angle lies in the range if yes return key

                if angle >= l and angle < u:

                    return key

        # If angle is not in any range we return -1. (Not going to happen. Its there for correctness)

        return -1

# A function to apply dicreet convolutions

def apply\_discrete\_convolution(image, mask):

    '''

    Args:

        image : An image to use for convolution

        mask  : An mask to use for convolution

    Returns:

        convolved image: An image after convolution

    '''

    # Get the shape of image and mask

    (m\_image, n\_image), (m\_mask, n\_mask) = image.shape, mask.shape

    # Compute the reference pixel index from where output array will start populating

    rpi\_m, rpi\_n = int(np.floor(m\_mask/2)), int(np.floor(n\_mask/2))

    # Initialize an output array with nan values

    output\_arr = np.ones((m\_image, n\_image)) \* np.nan

    # Iterate through the image

    for i in range(m\_image - m\_mask + 1):

        for j in range(n\_image - n\_mask + 1):

            # Isolate the image slice to apply convolution

            img\_slice = image[i:i+m\_mask, j:j+n\_mask]

            # Apply convolution and store the result in output array in approriate location

            output\_arr[i+rpi\_m][j+rpi\_n] = np.sum(img\_slice \* mask)

    return output\_arr

# A function to convert negative angles to positive angles

def get\_positive\_angle(angle):

    pos\_angle = angle.copy()

    pos\_angle[pos\_angle<0] += 360

    return pos\_angle

**Output Images**

**Image 1**

1. **Normalized image result after Gaussian smoothing**

[input image – house.bmp, output image - House\_gaussian\_smooth\_normalized.bmp]

A picture containing outdoor

Description automatically generated

1. **Normalized horizontal and vertical gradient responses (two separate images.) To generate normalized gradient responses, take the absolute value of the results first and then normalize.**

Gx:

[input image – house.bmp, output image – House\_Gx\_Normalized.bmp]

A picture containing dark, night

Description automatically generated

Gy:

[input image – house.bmp, output image - House\_Gy\_Normalized.bmp]

A picture containing dark

Description automatically generated

1. **Normalized gradient magnitude image.**

[input image – house.bmp, output image - House\_gradient\_magnitude\_normalized.bmp]

A picture containing dark, black, white, night

Description automatically generated

1. **Normalized gradient magnitude image after non-maxima suppression.**

[input image – house.bmp, output image - House\_non\_maxima\_supression.bmp]

Text

Description automatically generated with medium confidence

**(5) Binary edge maps using simple thresholding for thresholds chosen at the 25th, 50th and 75th percentiles**

Thresholding T1:

[input image – house.bmp, output image - House\_threshold\_t1\_2.28.bmp]

Diagram, engineering drawing

Description automatically generated

Thresholding T2:

[input image – house.bmp, output image - House\_threshold\_t2\_5.3.bmp]

A picture containing text, electronics

Description automatically generated

Thresholding T3:

[input image – house.bmp, output image - House\_threshold\_t3\_15.56.bmp]

Diagram, engineering drawing

Description automatically generated

**Image 2**

1. **Normalized image result after Gaussian smoothing**

[input image – Test patterns.bmp, output image - Test patterns\_non\_maxima\_supression.bmp]

A picture containing text

Description automatically generated

1. **Normalized horizontal and vertical gradient responses (two separate images.) To generate normalized gradient responses, take the absolute value of the results first and then normalize.**

**Gx:**

[input image – Test patterns.bmp, output image - Test patterns\_non\_maxima\_supression.bmp]

A picture containing night

Description automatically generated

**Gy:**

[input image – Test patterns.bmp, output image - Test patterns\_non\_maxima\_supression.bmp]

A close up of a piano

Description automatically generated with low confidence

1. **Normalized gradient magnitude image.**

[input image – Test patterns.bmp, output image - Test patterns\_maxima\_supression.bmp]

A close-up of a computer

Description automatically generated with low confidence

1. **Normalized gradient magnitude image after non-maxima suppression.**

[input image – Test patterns.bmp, output image - Test patterns\_non\_maxima\_supression.bmp]

A picture containing text, blackboard, black

Description automatically generated

**(5) Binary edge maps using simple thresholding for thresholds chosen at the 25th, 50th and 75th percentiles**

Thresholding T1:

[input image – Test patterns.bmp, output image - Test patterns\_threshold\_t1\_2.28.bmp]

Diagram, schematic

Description automatically generated

Thresholding T2:

input image – Test patterns.bmp, output image – Test\_patterns\_threshold\_t2\_24.45.bmp]

A screen shot of a computer

Description automatically generated with low confidence

Thresholding T3:

[input image – Test patterns.bmp, output image - Test patterns\_threshold\_t3\_15.56.bmp]

Diagram

Description automatically generated