Canny Edge Detector

PROJECT - 1

Team:

- Atharva Bhagwat [acb9244]
- Shubham Gundawar [ssg9763]

Steps for Canny Edge Detector

1) Read image in grayscale
2) Gaussian smoothing
3) Gradient calculation, magnitude calculation, gradient angle calculation
4) Non-maxima suppression
5) Thresholding: use simple thresholding and produce three binary edge maps by using three thresholds chosen at the 25th, 50th and 75th percentiles of the gradient magnitudes after non-maxima suppression

Note: Input image and output image should be of same size. Replace undefined values with 0.

Installing packages: opencv-python, numpy

pip3 install -r requirements.txt

Usage

python3 main.py <path_to_img_file>
e.g**:** python3 main.py input_images/House.bmp

Result Location

After execution a ‘results’ folder will be created containing subfolders with output images. Subfolder name will be the same as the input image filename.

Source Code

```
"""
Steps:
1) Read image in grayscale
2) Gaussian smoothing
3) Gradient calculation, magnitude calculation, gradient angle calculation
4) Non-maxima suppression
5) Thresholding: use simple thresholding and produce three binary edge maps by
using three thresholds chosen at the 25th, 50th and 75th percentiles of the
gradient magnitudes after non-maxima suppression

Notes: Input image and output image should be of same size. Replace undefined values
with 0.

To save:
1) Normalized  image result after Gaussian smoothing.
2) 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**
3) Normalized gradient magnitude image.
4) Normalized gradient magnitude image after non-maxima suppression.
5) Binary edge maps using simple thresholding for thresholds chosen at the 25th, 50th
and 75th percentiles.

*******************************************************************************

Dependency Installation: pip3 install -r requirements.txt

Usage: python3 main.py <path__to_img_file>

Example: python3 main.py input_images/House.bmp

After execution, 'results' folder will be created containing sub-folders with results
of input images.

*******************************************************************************

Libraries Used:
- os: Create folder structure for output images
- cv2: Read input image, write output images
- argparse: Parse arguments (Input image file path)
```

```
- numpy: ndarray handling, magnitude calculation (sqrt(a^2+b^2)), gradient angle
calculation, setting undefined pixel values to 0
"""

import os
import cv2
import argparse
import numpy as np

class CannyEdgeDetector():
   def __init__(self, img_path):
       """Initialise variables, constants

       Args:
           img_filename (str): Image filename (.bmp)
       """
       self.img_path = img_path
       self.img_filename = self.get_img_filename()
       self.output_folder = os.path.join('results', self.img_filename.split('.')[0])
       self.pad = 0
       self.img = None
       self.smooth_img = None
       self.gradient_x = None
       self.gradient_y = None
       self.gradient_angle = None
       self.gradient_magnitude = None
       self.quantized_angle = None
       self.magnitude_nms = None
       self.edgemap_t25 = None
       self.edgemap_t50 = None
       self.edgemap_t75 = None

       # Gaussian filter as per the question
       self.GAUSSIAN_FILTER = 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]]
                                )
```

```
        # Sum of entries in the gaussian filter
        self.GAUSSIAN_NORMALIZATION_FACTOR = 140
        # Maximum possible value for gradients
        self.GRADIENT_NORMALIZATION_FACTOR = 3
        # Maximum possible value for gradient magnitude
        self.MAGNITUDE_NORMALIZATION_FACTOR = 3*(2**0.5)

        # Prewitt's horizontal gradient operator
        self.PREWITT_X = np.array(
                            [[-1,0,1],
                            [-1,0,1],
                            [-1,0,1]]
                        )
        # Prewitt's vertical gradient operator
        self.PREWITT_Y = np.array(
                            [[1,1,1],
                            [0,0,0],
                            [-1,-1,-1]]
                        )

        # Sector map to quantize gradient angles depending on multiple of 22.5
        self.SECTORS = {
            0:0,
            1:1,
            2:1,
            3:2,
            4:2,
            5:3,
            6:3,
            7:0,
            8:0
        }

        # Map to calculate neighbors depending on the sector
        self.NEIGHBORS = {
            0:{'l':(0,-1),'r':(0,1)},
            1:{'l':(1,-1),'r':(-1,1)},
            2:{'l':(-1,0),'r':(1,0)},
            3:{'l':(-1,-1),'r':(1,1)}
        }
```

```
        # Call to driver function. Performs canny edge detection on input image.
        self.driver()

    def is_dir(self, directory):
        """Helper function to create directories if they don't exist

        Args:
        directory (str): Directory path
        """
        if not os.path.isdir(directory):
            os.makedirs(directory)
            print(f'Creating {directory}...')

    def get_img_filename(self):
        """Helper function to extract filename from image path, filename will be used
to save result images

        Returns:
            str: Image filename
        """
        img_path_split = self.img_path.split('/')
        img_path_split.reverse()
        return img_path_split[0]

    def read_img(self):
        """Function to read image stored at img_path in grayscale
        """
        self.img = cv2.imread(self.img_path, 0)

    def write_img(self, filename, file):
        """Saves image at out_path

        Args:
            filename (str): Filename to store images
            file (ndarray): Image numpy array
        """
        out_path = os.path.join(self.output_folder,filename)
        cv2.imwrite(out_path, file)
        print(f'{out_path} saved...')

    def update_padding(self, val):
        """Helper function which keeps track of padding values (undefined values)
```

```
        Args:
            val (int): Value is equal to len(filter)//2
        """
        self.pad += val

    def convolution(self, x, y):
        """Implementation of convolution operation

        Args:
            x (ndarray): First numpy array
            y (ndarray): Second numpy array

        Returns:
            ndarray: Resultant of x*y; * -> convolution
        """
        x_shape = x.shape
        y_shape = y.shape
        output_shape = (x_shape[0]-y_shape[0]+1, x_shape[1]-y_shape[1]+1)
        output = np.zeros(output_shape)
        for itr_x in range(output_shape[0]):
            for itr_y in range(output_shape[1]):
                output[itr_x][itr_y] = (x[itr_x:itr_x+y_shape[0],
itr_y:itr_y+y_shape[1]]*y).sum()
        return output

    def angle_calc(self):
        """Calculates gradient angle: tan inv (Gy/Gx)
        """
        self.gradient_angle = np.zeros(self.gradient_magnitude.shape)
        self.gradient_angle = np.rad2deg(np.arctan2(self.gradient_y, self.gradient_x))
        # If angle is negative add 180 to make it positive
        self.gradient_angle[self.gradient_angle < 0] += 180

    def get_sector(self, angle):
        """Returns sector value (0-3)
            Logic: As the sector wheel is same along the 0-180 degree line,
            we reduce the range from 0-360 to 0-180 for the gradient angle
            We achieve this by subtracting it by 180 if it is greater than 180
            We then divide this value by 22.5(as the sector is divided into smaller
sectors of 22.5 degrees)
```

```
            This value gives us the multiple which will further give us the sector
using dictionary
            Usage of dictionary reduces the complexity of the problem

        Args:
            angle (float): Gradient angle for a pixel location

        Returns:
            int: Sector value (0-3)
        """
        if angle > 180:
            angle -= 180
        angle = int(angle//22.5)
        return self.SECTORS[angle]

    def quantize_angle(self):
        """Function iterates over the gradient angle array and calculates sector
            for every pixel location.
        """
        self.quantized_angle = np.zeros(self.gradient_magnitude.shape)
        for itr_x in range(self.quantized_angle.shape[0]):
            for itr_y in range(self.quantized_angle.shape[1]):
                self.quantized_angle[itr_x][itr_y] =
self.get_sector(self.gradient_angle[itr_x][itr_y])

    def nms_compare(self, ind_x, ind_y, sector):
        """Function calculates neighbor coordinates and checks if the center pixel is
maximum out of the neighbors
            If the center pixel is the maximum, then function returns the magnitude of
the pixel
            Else returns 0

        Args:
            ind_x (int): X coordinate of the center pixel
            ind_y (int): Y coordinate of the center pixel
            sector (int): Quantized gradient angle

        Returns:
            int: Gradient magnitude or 0
        """
        # Calculation of neighbors depending on the sector
        # NEIGHBOR dictionary is used for calculating indexes of the ndarray
```

```
        neighbor_l =
{'x':ind_x+self.NEIGHBORS.get(sector).get('l')[0],'y':ind_y+self.NEIGHBORS.get(sector)
.get('l')[1]}
        neighbor_r =
{'x':ind_x+self.NEIGHBORS.get(sector).get('r')[0],'y':ind_y+self.NEIGHBORS.get(sector)
.get('r')[1]}
        if (self.gradient_magnitude[ind_x][ind_y] >
self.gradient_magnitude[neighbor_l.get('x')][neighbor_l.get('y')]) and
(self.gradient_magnitude[ind_x][ind_y] >
self.gradient_magnitude[neighbor_r.get('x')][neighbor_r.get('y')]):
            return self.gradient_magnitude[ind_x][ind_y]
        else:
            return 0

    def apply_threshold(self, x, threshold_25, threshold_50, threshold_75):
        """Applies simple thresholding operation
            If magnitude < threshold, set the edgemap pixel value to 0
            Else set the edgemap pixel value to 255 (white pixel)

        Args:
            x (ndarray): Gradient magnitude after non-maxima suppression
            threshold_25 (float): 25th percentile value from gradient magnitudes after
nms (excluding magnitudes equal to 0)
            threshold_50 (float): 50th percentile value from gradient magnitudes after
nms (excluding magnitudes equal to 0)
            threshold_75 (float): 75th percentile value from gradient magnitudes after
nms (excluding magnitudes equal to 0)

        Returns:
            ndarray: Edgemap created after applying simple threshold operation with
threshold_25
            ndarray: Edgemap created after applying simple threshold operation with
threshold_50
            ndarray: Edgemap created after applying simple threshold operation with
threshold_75
        """
        edgemap_t25 = np.zeros(x.shape)
        edgemap_t50 = np.zeros(x.shape)
        edgemap_t75 = np.zeros(x.shape)
        for itr_x in range(x.shape[0]):
            for itr_y in range(x.shape[1]):
                if x[itr_x][itr_y] <= threshold_25:
```

```
                    edgemap_t25[itr_x][itr_y] = 0
                else:
                    edgemap_t25[itr_x][itr_y] = 255

                if x[itr_x][itr_y] <= threshold_50:
                    edgemap_t50[itr_x][itr_y] = 0
                else:
                    edgemap_t50[itr_x][itr_y] = 255

                if x[itr_x][itr_y] <= threshold_75:
                    edgemap_t75[itr_x][itr_y] = 0
                else:
                    edgemap_t75[itr_x][itr_y] = 255
        return edgemap_t25, edgemap_t50, edgemap_t75

    def gaussian_smoothing(self):
        """Gaussian smoothing operation: IMAGE * GAUSSIAN_FILTER; * -> convolution
        """
        self.smooth_img = self.convolution(self.img,
self.GAUSSIAN_FILTER)/self.GAUSSIAN_NORMALIZATION_FACTOR
        self.update_padding(len(self.GAUSSIAN_FILTER)//2)
        self.write_img('smooth_'+self.img_filename, np.pad(self.smooth_img, self.pad))

    def gradient_calc(self):
        """Calculate horizontal and vertical gradients using prewitt operator: IMAGE *
PREWITT_X and IMAGE * PREWITT_y; * -> convolution
        """
        self.gradient_x = self.convolution(self.smooth_img, self.PREWITT_X)
        self.gradient_y = self.convolution(self.smooth_img, self.PREWITT_Y)
        self.update_padding(len(self.PREWITT_X)//2)

        self.gradient_magnitude = np.hypot(self.gradient_x, self.gradient_y)

        self.angle_calc()

        # Normalizing gradients and gradient magnitude to set the range to [0, 255]
        # Normalization formula: abs(x)/maximum possible value
        # Maximum possible value for gradients = 3
        # Maximum possible value for gradient magnitude = 3*sqrt(2)
        self.gradient_x = abs(self.gradient_x)/self.GRADIENT_NORMALIZATION_FACTOR
        self.gradient_y = abs(self.gradient_y)/self.GRADIENT_NORMALIZATION_FACTOR
```

```
        self.gradient_magnitude =
self.gradient_magnitude/self.MAGNITUDE_NORMALIZATION_FACTOR
        self.write_img('horizontal_'+self.img_filename, np.pad(self.gradient_x,
self.pad))
        self.write_img('vertical_'+self.img_filename, np.pad(self.gradient_y,
self.pad))
        self.write_img('magnitude_'+self.img_filename, np.pad(self.gradient_magnitude,
self.pad))


    def nms(self):
        """Non-maxima suppression function
            Steps:
                1) Quantize gradient angles
                2) Iterate over gradient magnitude and compare center pixel with
neighbors
        """
        self.quantize_angle()
        self.magnitude_nms = np.zeros(self.gradient_magnitude.shape)
        for itr_x in range(1, self.magnitude_nms.shape[0]-1):
            for itr_y in range(1, self.magnitude_nms.shape[1]-1):
                self.magnitude_nms[itr_x][itr_y] = self.nms_compare(itr_x, itr_y,
self.quantized_angle[itr_x][itr_y])
        self.magnitude_nms = np.pad(self.magnitude_nms, self.pad)
        self.write_img('nms_magnitude_'+self.img_filename, self.magnitude_nms)

    def thresholding(self):
        """Simple thresholding operation
            We calculate 3 thresholds:
                1) T_25 = 25th percentile of magnitude after nms (excluding magnitude
equal to 0)
                2) T_50 = 50th percentile of magnitude after nms (excluding magnitude
equal to 0)
                3) T_75 = 75th percentile of magnitude after nms (excluding magnitude
equal to 0)
        """
        magnitude_vals = [value for value in self.magnitude_nms.flatten() if value !=
0]
        threshold_25, threshold_50, threshold_75 = np.percentile(magnitude_vals,[25,
50, 75])
```

```
        self.edgemap_t25, self.edgemap_t50, self.edgemap_t75  =
self.apply_threshold(self.magnitude_nms, threshold_25, threshold_50, threshold_75)

        self.write_img('edgemap_t25_'+self.img_filename, self.edgemap_t25)
        self.write_img('edgemap_t50_'+self.img_filename, self.edgemap_t50)
        self.write_img('edgemap_t75_'+self.img_filename, self.edgemap_t75)

    def driver(self):
        """Calls canny edge detection functions in order:
            - Read image
            - Gaussian smoothing
            - Gradient and gradient magnitude calculation using prewitt's operator
            - Non-maxima suppression
            - Thresholding
        """
        self.is_dir(self.output_folder)
        self.read_img()
        self.gaussian_smoothing()
        self.gradient_calc()
        self.nms()
        self.thresholding()


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Canny Edge Detector.')
    parser.add_argument('img_filename', type=str, help='image filename')
    args = parser.parse_args()
    obj = CannyEdgeDetector(args.img_filename)
```

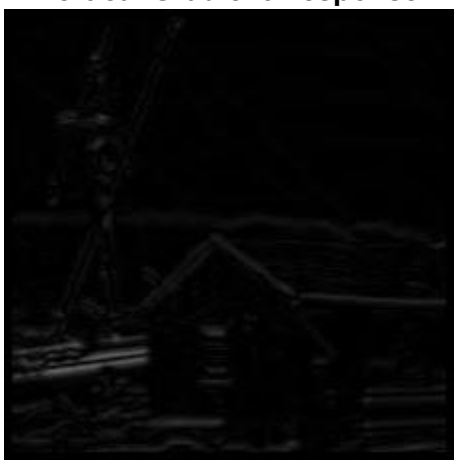
Results for House.bmp

Original Image

Gaussian Smoothing

Horizontal Gradient Response

Vertical Gradient Response



Gradient Magnitude

Non Maxima Suppression Response

Thresholding Response [25th percentile]

Thresholding Response [50th percentile]

Thresholding Response [75th percentile]

Results for test_patterns.bmp

Original Image

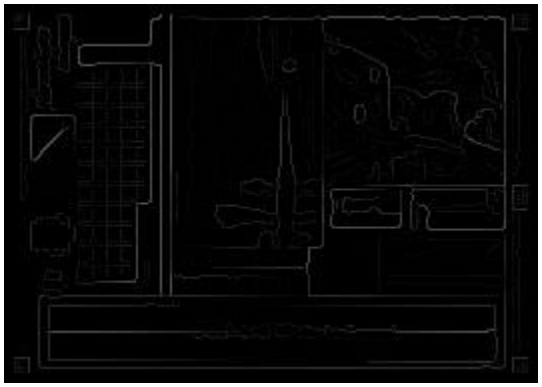
Gaussian Smoothing

Horizontal Gradient Response

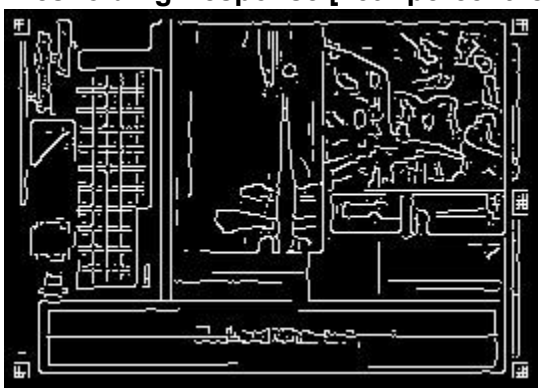
Vertical Gradient Response

Gradient Magnitude

Non Maxima Suppression



Thresholding Response [25th percentile]



Thresholding Response [50th percentile]

Thresholding Response [75th percentile]