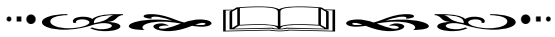
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA CƠ KHÍ

BỘ MÔN CƠ ĐIỆN TỬ



BÁO CÁO BÀI TẬP LỚN

NHẬP MÔN THỊ GIÁC MÁY TÍNH

ĐỀ TÀI:

CANNY EDGE DETECTION ALGORITHM

Giảng viên hướng dẫn: TS. LÊ THANH HẢI

Sinh viên thực hiện	**MSSV**
ĐỖ NGUYỄN ĐĂNG ANH	**1710457**
LÊ CẢNH HOÀNG	**1710095**
ĐỖ HOÀNG VĨNH	**1713998**
NGUYỄN VĂN TRIỆU VỸ	**1714065**

TP. Hồ Chí Minh, Ngày 26 tháng 12 năm 2020

LỜI GIỚI THIỆU

Xin kính chào Thầy và các bạn, đây là bài báo cáo bài tập lớn, mang tính chất tổng hợp những hiểu biết và tìm hiểu của nhóm xung quanh thuật toán Canny. Bài báo cáo được tham khảo nhiều nguồn và lập trình dựa trên ngôn ngữ Python, vì thế còn có nhiều thiếu sót mong Thầy và các bạn đọc nếu thấy bất cập hãy liên hệ và trao đổi với nhóm qua gmail vinhdo0913@gmail.com để có những phản hồi, giải đáp tích cực nhất.

Bài báo cáo gồm 3 phần chính, phần I, ***Canny Edge Detction trong thư viện OpenCv*** – sẽ sơ lược tổng quan về thuật toán cũng như chương trình và kết quả trong thư viện có sẵn viết dựa trên Python này. Phần II, ***Thuật toán Canny Edge Detection*** – đi vào từng bước tạo ra thuật toán tìm cạnh Canny này, ở đây không mất tính tổng quát nhóm sẽ thêm một vài bước để tăng chất lượng xử lý hình ảnh của thuật toán. Phần III, ***Kết quả và nhận xét*** – nhận xét giữa chương trình dùng thư viện OpenCv và dựa theo thuật toán để xây dựng lại không sử dụng hàm trong OpenCv.

Xin chân thành cảm ơn Thầy và các bạn đã hỗ trợ nhóm trong suốt học phần này!

Nhóm trưởng

Đỗ Hoàng Vĩnh

I. CANNY EDGE DETECTION TRONG THƯ VIỆN OPENCV:

Khái quát thuật toán và thư viện sử dụng

Với sự phát triển công nghệ các thuật toán xử lý ảnh ngày càng trở nên thông dụng vì thế các ngôn ngữ cũng dần phát triển theo mảng riêng. Python với OpenCv được nhóm chọn và thực hiện bài tập lớn lần này, đơn thuần vì dễ tiếp cận, mạnh trong việc xử lý số liệu, AI… Và cũng kết nối với rất nhiều thư viện khác kể cả các thư viện về Data Science, Deep Learning, AI, và các thư viện game, GUI(Graphical user interface)…

Đối với OpenCv dựa trên ngôn ngữ Python là một sự tối ưu về mặt chương trình, chương trình sẽ ngắn hơn nhiều so với lập trình trên các nền tảng ngôn ngữ khác như C, C++, C#... nhưng hạn chế vì phải hiểu sâu về Toán, Đại số tuyến tính, thiếu đi GUI – công cụ để trình diện các thao tác nhanh chóng.

Canny Edge Detection Algorithm là thuật toán tìm cạnh cơ bản được giới thiệu trong môn Thị giác máy tính, sau khi trải qua những bước tiền xử lý như chuyển màu sắc, lọc nhiễu thì việc tìm cạnh có trong hình ảnh rất quan trọng vì chúng giúp ta nhận biết được những sự vật, hành động, đặc trưng, đặc điểm của một đối tượng có trong ảnh.

OpenCV thực hiện tất cả các bước trên bằng một hàm duy nhất **cv2.Canny()**. Tham số đầu tiên là ảnh đầu vào. Tham số thứ hai và thứ ba tương ứng với minVal và maxVal. Tham số thứ tư là aperture_size cho biết kích thước của Sobel Kernel, mặc định là 3. Tham số cuối cùng là L2gradient cho biết công thức tính gradient. Nếu L2gradient=True sẽ sử dụng công thức đã được đề cập ở trên, ngược lại nó sẽ được tính theo công thức: Edge_Gradient(G)=|Gx|+|Gy|(G)=|Gx|+|Gy|. Mặc định, L2gradient=False.

```
edges = cv2.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]])
```

Với các tham số:

Tham số	Chức năng chi tiết
image	Ảnh đầu vào
edges	Ảnh đầu ra
threshold1	Giá trị ngưỡng thứ nhất dùng cho qua trình xét ngưỡng cường độ cao, thấp và tính liên tục của cạnh
threshold2	Giá trị ngưỡng thứ 2 dùng cho qua trình xét ngưỡng cường độ cao, thấp và tính liên tục của cạnh
apertureSize	Kích thước focus cho Sobel
L2gradient	Đề cập đến công thức tính cho giải thuật tìm cạnh một cách chính xác

Chương trình đầy đủ kết hợp GUI:

```
import PySimpleGUI as sg
import cv2
import os
import numpy as np

"""
Big project's program that displays a image||(webcam) using Opencv and
appling some very basic image functions
We using this file for checking our function written.
none:       no processing, just show Origin Image
threshold:  simple b/w-threshold on the luma channel, slider sets the
threshold value (BONUS)
canny:      edge finding with canny, sliders set the two threshold values
for the function => edge sensitivity
blur:       simple Gaussian blur, slider sets the sigma, i.e. the amount of
blur smear (BONUS)
hue:        moves the image hue values by the amount selected on the slider
(BONUS)
enhance:    applies local contrast enhancement on the luma channel to make
the image fancier - slider controls fanciness. (BONUS)
"""

def main():
    # Create the GUI with PySmpleGUI's library
    sg.theme('LightBlue')
```

```
    # Define the window's contents
    """
    Text is name of showing project
    """
    layout = [
      [sg.Text('Project of Computer Vison', size=(60, 1),
justification='center')],
      [sg.Image(filename='', key='-IMAGE-')],
      [sg.Radio('None', 'Radio', True, size=(10, 1))],
      [sg.Radio('threshold', 'Radio', size=(10, 1), key='-THRESH-'),
       sg.Slider((0, 255), 128, 1, orientation='h', size=(40, 15), key='-
THRESH SLIDER-')],
      [sg.Radio('canny', 'Radio', size=(10, 1), key='-CANNY-'),
       sg.Slider((0, 255), 128, 1, orientation='h', size=(20, 15), key='-
CANNY SLIDER A-'),
       sg.Slider((0, 255), 128, 1, orientation='h', size=(20, 15), key='-
CANNY SLIDER B-')],
      [sg.Radio('blur', 'Radio', size=(10, 1), key='-BLUR-'),
       sg.Slider((1, 11), 1, 3, orientation='h', size=(40, 15), key='-BLUR
SLIDER-')],
      [sg.Radio('hue', 'Radio', size=(10, 1), key='-HUE-'),
       sg.Slider((0, 225), 0, 1, orientation='h', size=(40, 15), key='-HUE
SLIDER-')],
      [sg.Radio('enhance', 'Radio', size=(10, 1), key='-ENHANCE-'),
       sg.Slider((1, 255), 128, 1, orientation='h', size=(40, 15), key='-
ENHANCE SLIDER-')],
      [sg.Button('Exit', size=(10, 1))],
    ]

    # Create the window and show it without the plot
    width, height = 640, 320
    window = sg.Window('OpenCV Integration', layout, location=(width,
height))
    # Create button "Browse", "OK", "Cancel" for choosing image locate in
your PC
    event_img, values_img = window.Layout([[sg.Input(key='-FILES-'),
sg.FilesBrowse()],[sg.OK(), sg.Cancel()]]).Read()
    #If you want to show webcam, please adjust here.
    # cap = cv2.VideoCapture(0)


    while True:
        event, values = window.read(timeout=20)
        # Statement breaks program!!!
        if event == 'Exit' or event == sg.WIN_CLOSED:
            break
        # If you want to read frame of webcam, please adjust here.
        # ret, frame = cap.read() # it returns 2 params and frame is the
images with default FPS
        # When turning on the webcam, please comment 1 line below.
        frame = cv2.imread(values_img['-FILES-']) # Read image when you
choose on GUI
        """ Task1: Threshold"""
        """ Task2: Canny Edge Detection"""
        """ Task3: Filter with Gaussian Blur"""
        """ Task4: Convert frame/image BGR to HSV """
        """ Task5: Basically, Enhance image"""
        if values['-THRESH-']:
```

```
            frame = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)  #
bgr2grayscale(frame)
            frame = cv2.threshold(frame, values['-THRESH SLIDER-'], 255,
cv2.THRESH_BINARY)[1]
        elif values['-CANNY-']:
            frame = cv2.Canny(frame, values['-CANNY SLIDER A-'], values['-
CANNY SLIDER B-'])
        elif values['-BLUR-']:
            frame = cv2.GaussianBlur(frame, (21, 21), values['-BLUR SLIDER-
'])
        elif values['-HUE-']:
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
            frame[:, :, 0] += int(values['-HUE SLIDER-'])
            frame = cv2.cvtColor(frame, cv2.COLOR_HSV2BGR)
        elif values['-ENHANCE-']:
            enh_val = values['-ENHANCE SLIDER-'] / 40
            clahe = cv2.createCLAHE(clipLimit=enh_val, tileGridSize=(8, 8))
            lab = cv2.cvtColor(frame, cv2.COLOR_BGR2LAB)
            lab[:, :, 0] = clahe.apply(lab[:, :, 0])
            frame = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
        # For updating statement above when it actives. Display in GUI
        imgbytes = cv2.imencode('.png', frame)[1].tobytes()
        window['-IMAGE-'].update(data=imgbytes)
    #Just close the GUI when we click button "Exit" on GUI  or close GUI
tab!!!
    window.close()

main()
```

II. THUẬT TOÁN CANNY EDGE DETECTION:

Để thực hiện trên môi trường Python nhóm lưu ý về các thư viện cần phải install và khai báo khi bắt đầu chương trình lớn.

```
# Import librabry as needed
import numpy as np
from scipy import ndimage
import cv2
import matplotlib.pyplot as plt
```

Bên cạnh đó còn có các thư viện hỗ trợ GUI cũng được dùng cho project này:

```
import PySimpleGUI as sg
```

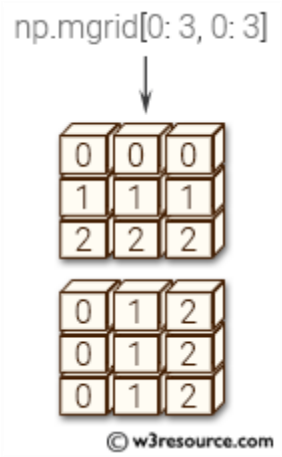
1. Noise reduction (lọc nhiễu)

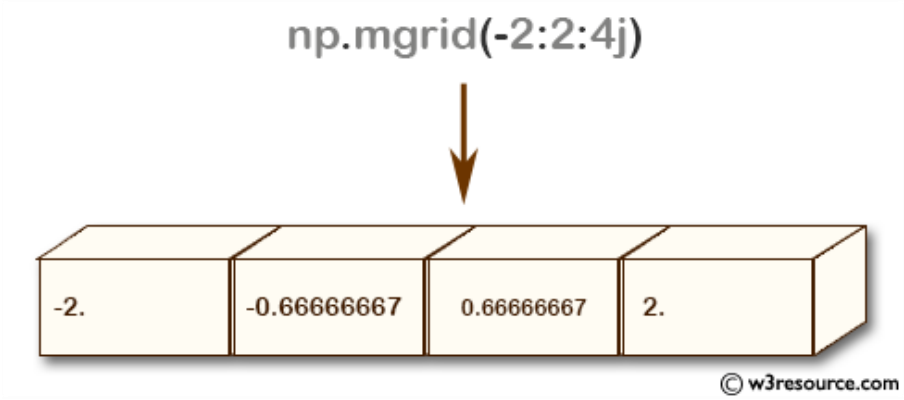
Một bức ảnh luôn tồn tại đáng kể một số nhiễu bên trong. Độ ít hay nhiều sẽ ảnh hưởng đến chất lượng của ảnh và các vấn đề liên quan đến nhận diện, xử lý tín hiệu, thu thập dữ liệu… Việc sử dụng những bộ lọc thông dụng sẽ giúp ta loại đi bớt nhiễu cơ bản, kể tới như: 2D filter, median filter, gaussian filter… Và hầu hết chúng đều phải dựa vào thuật toán “convolution” với kích thước của kernel (3x3, 5x5, 7x7…) thì tuỳ vào

kernel size mà việc làm mờ(lọc nhiễu) sẽ đạt hiệu quả cao. Nghĩ đơn giản rằng Kernel càng nhỏ thì hiệu quả lọc nhiễu/làm mở càng ít.

Dựa theo phương trình cho Gaussian filter kernel of size (2k+1)(2k+1) :

$$H_{ij} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{\left(i-(k+1)\right)^2+\left(j-(k+1)\right)^2}{2\sigma^2}\right); 1 \le i,j \le (2k+1)$$





Hàm trong thư viện numpy liên quan đến array sử dụng trong chương trình

Đây là cách tìm kernel của bô lộc Gauss.

```
def kernel_Gaussian(size, sigma):
    size = int(size)//2
    x, y = np.mgrid[-size:size+1,-size:size+1]
    normal = 1/(2*np.pi*sigma**2)
    g = np.exp(-((x**2 + y**2)/(2*sigma**2)))*normal
    return g
```

Nhưng vì bước làm này có thì sẽ rất hữu ích cho các ảnh bị nhiễu, tuy nhiên nếu như sử dụng thư viện đã tích hợp sẵn hàm `kernel_Gaussian(size, sigma)` để các bước làm rút ngắn lại (vì bước này có khi cần, nhưng cũng có khi không cần thiết bởi lọc nhiễu ở cường độ nào đó nếu lọc quá mạnh sẽ làm mất đi các phần quan trọng của đối tượng trong ảnh).

Vì thế đoạn chương trình sử dụng lọc nhiễu nhóm sẽ sử dụng thư viện Scipy (thư viện chứa những liên quan đến mathematics, science và engineering, tương tự dựa trên Numpy).

```
# 1. Gaussian blur to reduce noise
# We're passed the noise with function in library because the function easy
to write
# and the function in library very useful.!!
im2 = gaussian_filter(im, blur)
```

2. Gradient Calculation

Bước này ta thực hiện nhằm nhận diện được cường độ của cạnh và trực tiếp tính toán gradient giá trị pixels của ảnh. Cạnh của bức ảnh trả về với những cường độ pixel thay đổi. Để nhận diện được cạnh, điều đơn giản nhất là áp dụng bộ lọc để làm nổi bật cường độ thay đổi theo 2 phương: đứng (vertical – y hay còn gọi là cột – column), ngang (horizontal – x hay còn gọi là hang – row).

Ta có thể dựa vào 2 kernel của Sobel fiter cho 2 phương (horizontal và vertical) để giúp ta nhận diện tốt được cường độ thay đổi sau khi áp dụng Sobel.

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

Tính độ lớn $\boldsymbol{G}$ và hệ số góc $\boldsymbol{\theta}$ của gradient trong việc xác định cường độ cạnh và định hướng của cạnh sau xử lý.

$$|G| = \sqrt{I_x^2 + I_y^2},$$
$$\theta(x, y) = arctan\left(\frac{I_y}{I_x}\right)$$

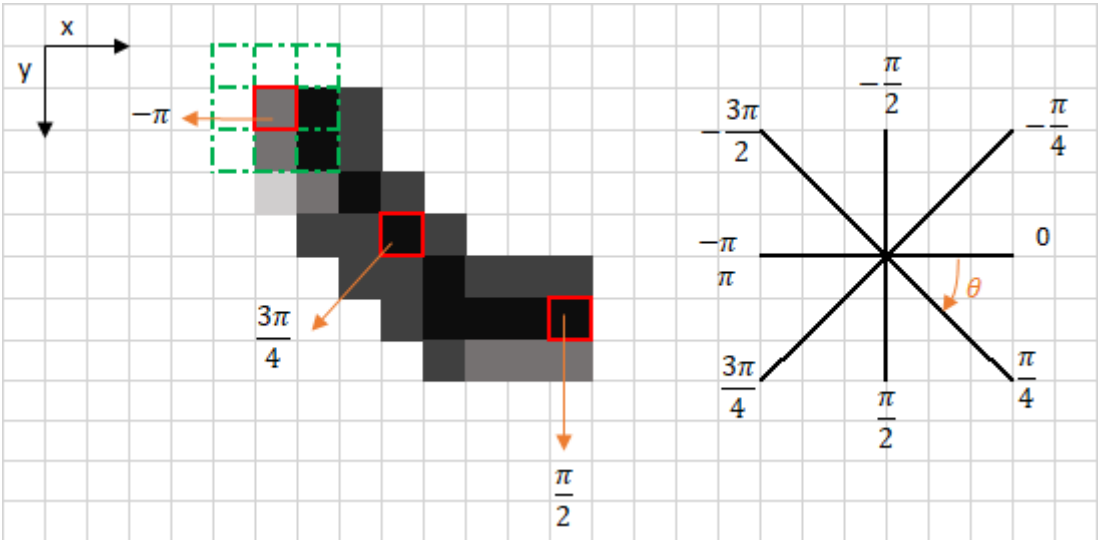
Đoạn chương trình với tác vụ miêu tả trên sử dụng Python với IDE Pycharm:

```
# 2. Use sobel filters to get horizontal and vertical gradients
# convolve using like convolution what we learning in CS201's subject
im3h = convolve(im2, [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
im3v = convolve(im2, [[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

# Get gradient and direction
# np.power(a,0.5) is square root & np.power(a,0.5) is square
grad = np.power(np.power(im3h, 2.0) + np.power(im3v, 2.0), 0.5)
theta = np.arctan2(im3v, im3h)
thetaQ = (np.round(theta * (5.0 / np.pi)) + 5) % 5  # Quantize direction
```

3. Non-Maximum Suppression

Mục tiêu tối ưu nhất, lý tưởng nhất là ảnh sau khi xử lý sẽ trả về những cạnh mỏng (thin edges). Và bước làm mỏng cạnh sẽ được sử dụng như sau:

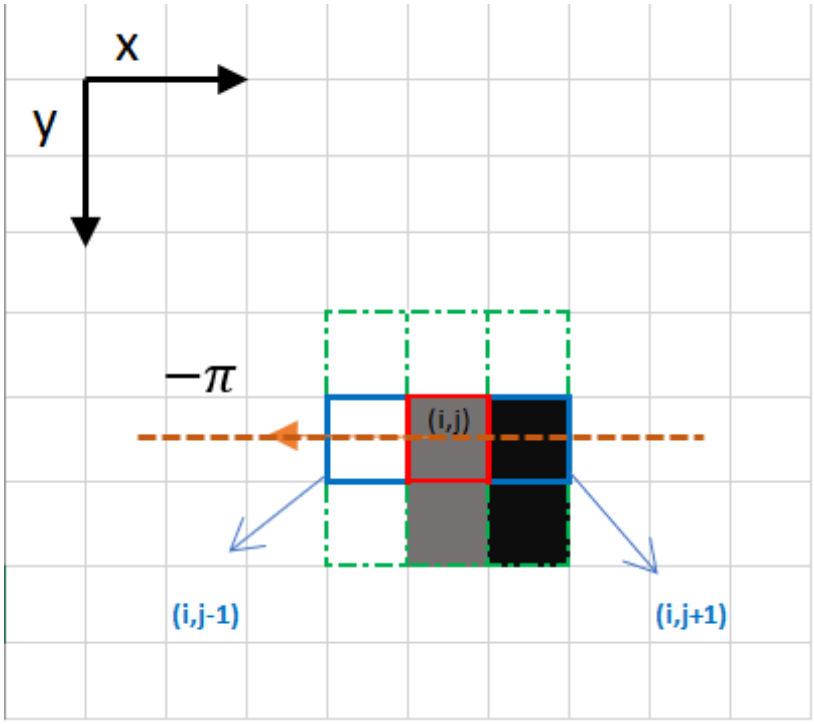
Với nguyên tắc đơn giản, thuật toán sẽ đi qua tất cả các điểm trong ma trận gradient intensity ***G*** và đồng thời tìm các pixels có giá trị lớn nhất (maximum value) theo hướng cạnh (định hướng được xem gần như cạnh do chênh lệch cường độ hai bên lân cận điểm đó).



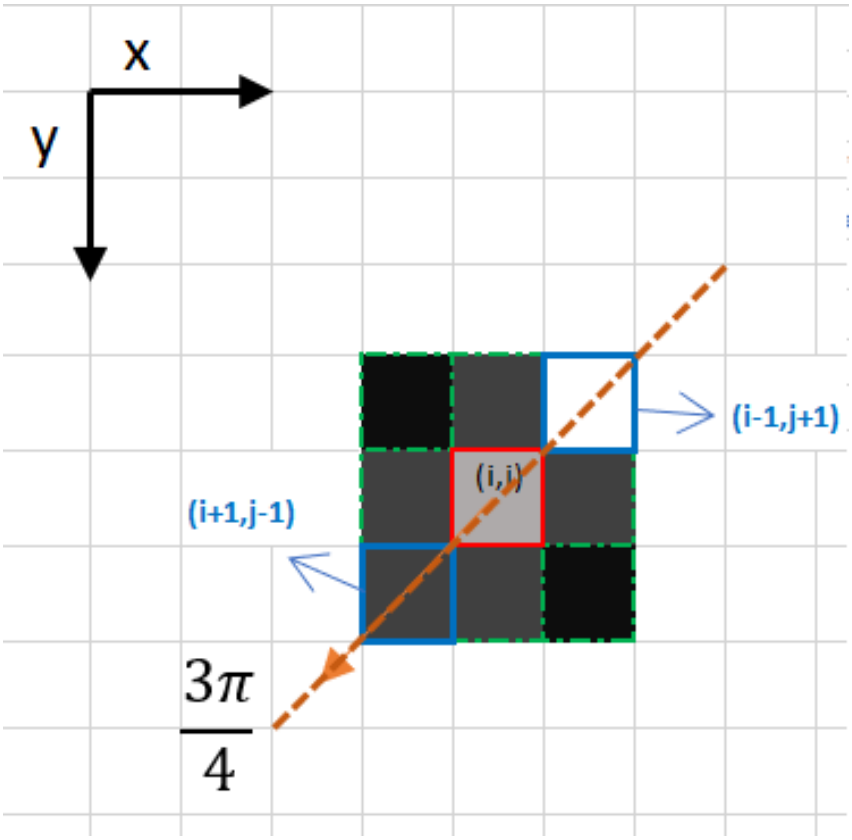
Một ví dụ đơn giản về bước làm mỏng[1]

[1] https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

Vị trí phía trên bên trái của hình chữ nhật đỏ đại diện cho cường độ điểm ảnh của Gradient Intensity matrix (ma trận sự thay đổi về cường độ sáng theo mức sáng). Và hướng của cạnh đại diện bởi mũi tên màu cam góc giới hạn $\pm 180°$ ($\pm\, \pi\, rad$).



Mục đích của bước này để kiểm tra pixels trên một hướng (theo hướng của mũi tên cam) có cường độ sáng lớn hoặc nhỏ hơn vị trí đang xét. Như ảnh trên đang đi kiểm tra điểm trong hình chữ nhật nhỏ đỏ (i,j) so với (i,j-1) và (i,j+1). Nếu như một trong số hai pixels có cường độ lớn hơn cái đang xét thì chỉ giữ lại pixel có cường độ cao hơn. Vì vậy pixel (i,j-1) có cường độ cao (255 – màu trắng). Vì vậy đơn giản là ta đặt cho vị trí pixel (i,j) gán bằng 0. Nếu như không có pixels nào trên hướng thay đổi của cạnh có giá trị cường độ lớn thì giá trị của pixel đang xét giữ lại.



Tương tự, ta cũng xét cho trường hợp này giá trị pixel (i-1,j+1) đan là lớn vì thế pixel (i,j) sẽ xét về 0.

Vì thế để viết được hàm xét độ mỏng của cạnh đơn giản là giá trị đang xét nếu có cường độ cao(xét so với range(0,255)) sẽ được gán bằng giá trị cao nhất là 255. Tóm lại ảnh sau khi ra khỏi sobel fiter qua bộ lọc mỏng cạnh sẽ làm cho việc phát hiện cạnh bớt nhiễu cũng như hình thành được rõ ràng hơn, nhưng mặc khác cũng làm mất một số đặc trưng, điểm quan trọng… Tổng hợp lại ta có các bước thực hiện như sau:

- Tạo một ma trận zero ban đầu cùng kích thước với ma trận sự thay đổi cường độ.
- Nhận định hướng thay đổi của cạnh dựa trên giá trị góc từ ma trận θ.
- Kiểm tra xem nếu pixel trên cùng hướng có cường độ cao hơn so với pixel đang xét.
- Trả về ảnh sau khi sử lý với non-maximum suppression (làm mỏng cạnh).

Đoạn chương trình với tác vụ miêu tả trên sử dụng Python với IDE Pycharm:

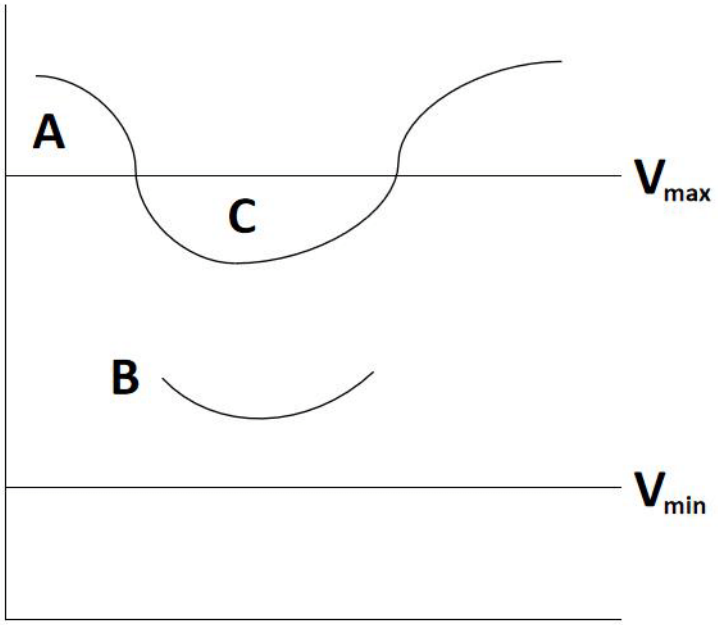
```
# 3. Non-maximum suppression
gradSup = grad.copy()
for r in range(im.shape[0]):
    for c in range(im.shape[1]):
        # Suppress pixels at the image edge
        if r == 0 or r == im.shape[0] - 1 or c == 0 or c == im.shape[1] -
1:
            gradSup[r, c] = 0
            continue
        tq = thetaQ[r, c] % 4
        #           N - North
        # W - West            E - East
        #           S - South
        if tq == 0:  # 0 is E-W (horizontal)
            if grad[r, c] <= grad[r, c - 1] or grad[r, c] <= grad[r, c +
1]:
                gradSup[r, c] = 0
        if tq == 1:  # 1 is NE-SW
            if grad[r, c] <= grad[r - 1, c + 1] or grad[r, c] <= grad[r +
1, c - 1]:
                gradSup[r, c] = 0
        if tq == 2:  # 2 is N-S (vertical)
            if grad[r, c] <= grad[r - 1, c] or grad[r, c] <= grad[r + 1,
c]:
                gradSup[r, c] = 0
        if tq == 3:  # 3 is NW-SE
            if grad[r, c] <= grad[r - 1, c - 1] or grad[r, c] <= grad[r +
1, c + 1]:
                gradSup[r, c] = 0
```

4. Double threshold

Bước này với mục đích nhận diện rằng các vị trí pixels có cường độ mạnh, yếu và không liên quan đến cạnh phát hiện được.

- Pixels mạnh là pixels có cường độ lớn và chắc rằng chúng là có vai trò một phần vào cạnh sau cùng, cường độ lớn dễ thấy và mượt.
- Pixels yếu có cường độ không quá nhỏ cũng không quá lớn chúng vẫn gần như liên tục nhưng đứt quảng vì do cường độ nhỏ nên có thể khó phát hiện.
- Và những pixels còn lại sẽ là không liên quan đến cạnh khi chúng nằm rải rác, thưa thớt trên ảnh cường độ.

Vậy ở bước này ta sử dụng hai giá trị ngưỡng cao (Vmax) và thấp (Vmin) để lọc ra các cường độ cạnh mạnh và yếu kể cả không liên quan.



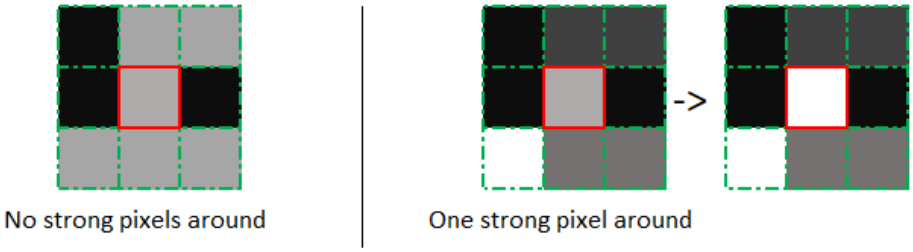
Phân ngưỡng cạnh để xem độ liên tục của chúng

Đoạn chương trình với tác vụ miêu tả trên sử dụng Python với IDE Pycharm:

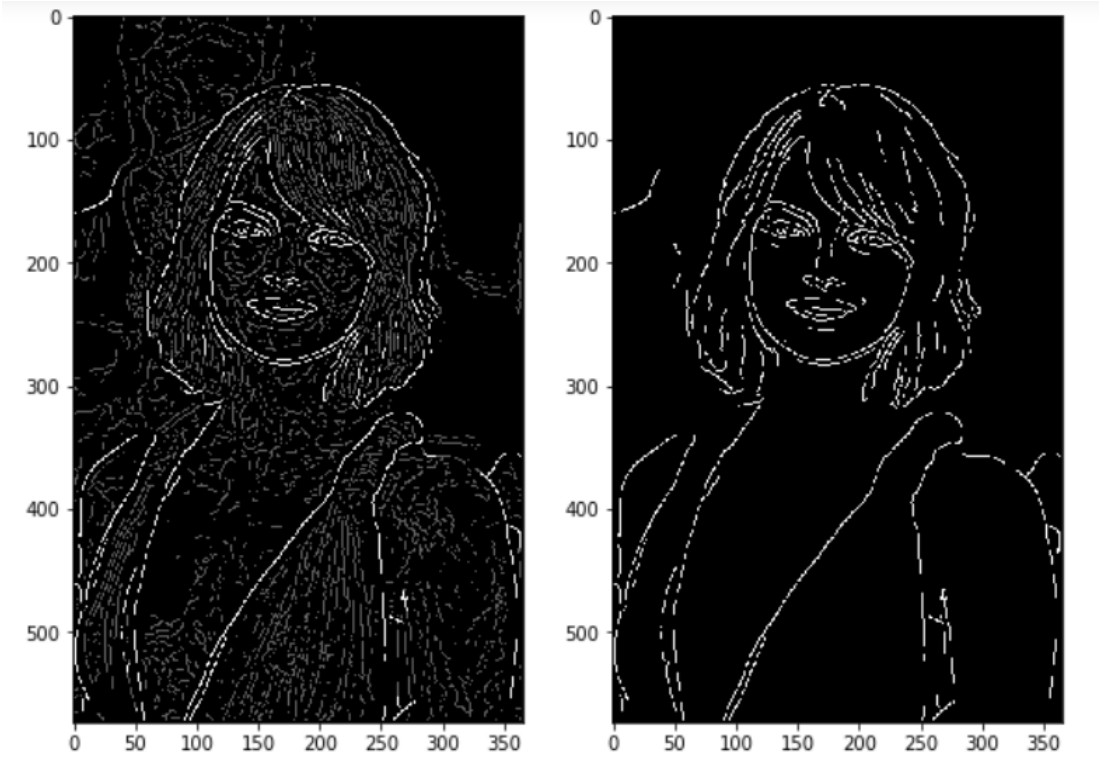
```
# 4. Double threshold
strongEdges = (gradSup > highThreshold)

# Strong has value 2, weak has value 1
thresholdedEdges = np.array(strongEdges, dtype=np.uint8) + (gradSup >
lowThreshold)
```

5. Edge Tracking by Hysteresis

Dựa trên kết quả của Double Threshold, Hysteresis bao gồm chuyển pixels đang xét từ yếu sang mạnh nếu xung quanh chúng có một và chỉ ít nhất một pixel cường độ cao thì lập tức pixel yếu sẽ gán lại cường độ cao như minh hoạ ảnh dưới.



Bước cuối cùng này quyết định cho ta việc cạnh sẽ hình thành liên tục hoặc đứt quãng hoặc có những nhiễu trong đó. Ví dụ minh hoạ bên dưới:



Đoạn chương trình với tác vụ miêu tả trên sử dụng Python với IDE Pycharm:

```
# 5.Tracing edges with hysteresis
# Find weak edge pixels near strong edge pixels
finalEdges = strongEdges.copy()
currentPixels = []
for r in range(1, im.shape[0] - 1):
    for c in range(1, im.shape[1] - 1):
        if thresholdedEdges[r, c] != 1:
            continue  # Not a weak pixel

        # Get 3x3 patch
        localPatch = thresholdedEdges[r - 1:r + 2, c - 1:c + 2]
        patchMax = localPatch.max()
        if patchMax == 2:
            currentPixels.append((r, c))
            finalEdges[r, c] = 1

# Extend strong edges based on current pixels
while len(currentPixels) > 0:
    newPix = []
    for r, c in currentPixels:
        for dr in range(-1, 2):
            for dc in range(-1, 2):
                if dr == 0 and dc == 0: continue
                r2 = r + dr
                c2 = c + dc
```

```
                if thresholdedEdges[r2, c2] == 1 and finalEdges[r2, c2] ==
0:
                    # Copy this weak pixel to final result
                    newPix.append((r2, c2))
                    finalEdges[r2, c2] = 1
    currentPixels = newPix

return finalEdges
```

Chương trình đầy đủ như sau:

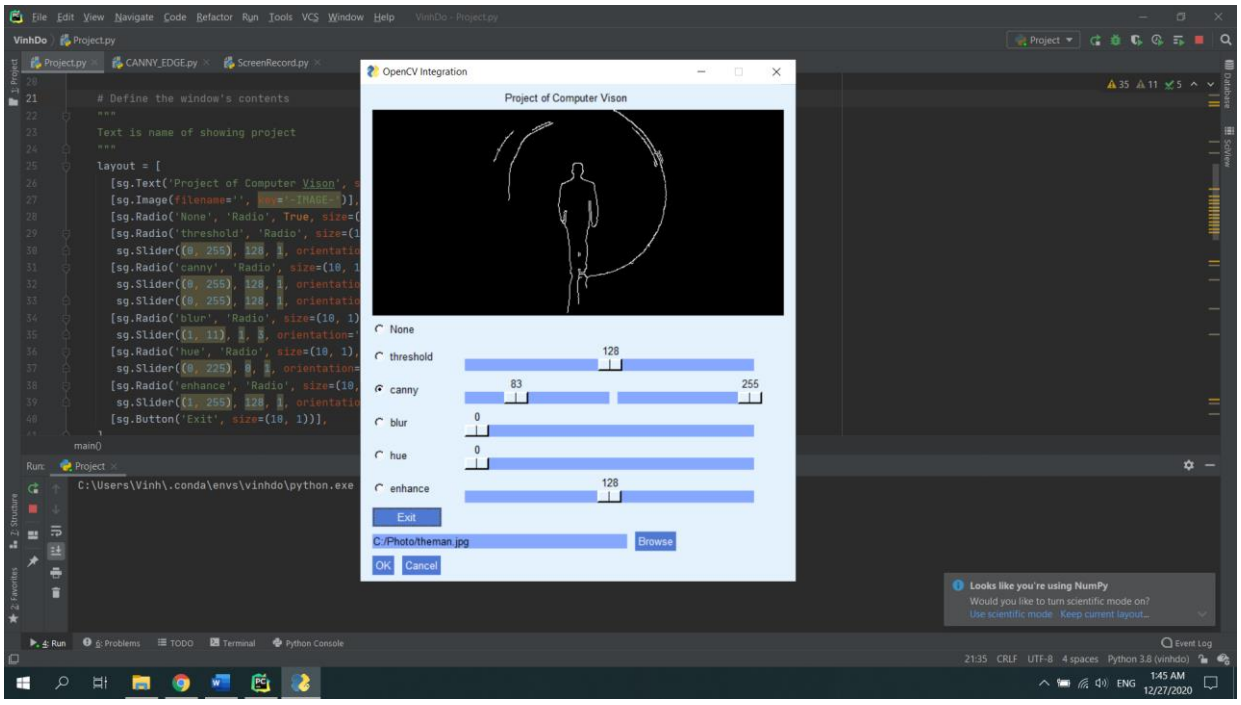
```
"""
This program introduce the Canny edge detection algorithm.
The Canny edge detection algorithm is composed of 5 steps:
1. Noise reduction;
2. Gradient calculation;
3. Non-maximum suppression;
4. Double threshold;
5. Edge Tracking by Hysteresis.
Note: One last important thing to mention, is that the algorithm
is based on grayscale pictures. Therefore, the pre-requisite
is to convert the image to grayscale before following
the above-mentioned steps.
"""

# import libraby as needed
import numpy as np
from scipy.ndimage.filters import convolve, gaussian_filter
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

'''Create Big function Canny Edge Detection with default: blur=1,
highThreshold=91, lowThreshold=31'''

def CannyEdgeDetector(im, blur=1, highThreshold=91, lowThreshold=31):
    im = np.array(im, dtype=float)  # Convert to float to prevent clipping
values

    # 1. Gaussian blur to reduce noise
    # We're passed the noise with function in library because the function
easy to write
    # and the function in library very useful.!!
    im2 = gaussian_filter(im, blur)

    # 2. Use sobel filters to get horizontal and vertical gradients
    # convolve using like convolution what we learning in CS201's subject
    im3h = convolve(im2, [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    im3v = convolve(im2, [[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

    # Get gradient and direction
    # np.power(a,0.5) is square root & np.power(a,0.5) is square
    grad = np.power(np.power(im3h, 2.0) + np.power(im3v, 2.0), 0.5)
    theta = np.arctan2(im3v, im3h)
    thetaQ = (np.round(theta * (5.0 / np.pi)) + 5) % 5  # Quantize
direction
```

```
    # 3. Non-maximum suppression
    gradSup = grad.copy()
    for r in range(im.shape[0]):
        for c in range(im.shape[1]):
            # Suppress pixels at the image edge
            if r == 0 or r == im.shape[0] - 1 or c == 0 or c == im.shape[1]
- 1:
                gradSup[r, c] = 0
                continue
            tq = thetaQ[r, c] % 4
            #           N - North
            # W - West            E - East
            #           S - South
            if tq == 0:  # 0 is E-W (horizontal)
                if grad[r, c] <= grad[r, c - 1] or grad[r, c] <= grad[r, c
+ 1]:
                    gradSup[r, c] = 0
            if tq == 1:  # 1 is NE-SW
                if grad[r, c] <= grad[r - 1, c + 1] or grad[r, c] <= grad[r
+ 1, c - 1]:
                    gradSup[r, c] = 0
            if tq == 2:  # 2 is N-S (vertical)
                if grad[r, c] <= grad[r - 1, c] or grad[r, c] <= grad[r +
1, c]:
                    gradSup[r, c] = 0
            if tq == 3:  # 3 is NW-SE
                if grad[r, c] <= grad[r - 1, c - 1] or grad[r, c] <= grad[r
+ 1, c + 1]:
                    gradSup[r, c] = 0

    # 4. Double threshold
    strongEdges = (gradSup > highThreshold)

    # Strong has value 2, weak has value 1
    thresholdedEdges = np.array(strongEdges, dtype=np.uint8) + (gradSup >
lowThreshold)

    # 5.Tracing edges with hysteresis
    # Find weak edge pixels near strong edge pixels
    finalEdges = strongEdges.copy()
    currentPixels = []
    for r in range(1, im.shape[0] - 1):
        for c in range(1, im.shape[1] - 1):
            if thresholdedEdges[r, c] != 1:
                continue  # Not a weak pixel

            # Get 3x3 patch
            localPatch = thresholdedEdges[r - 1:r + 2, c - 1:c + 2]
            patchMax = localPatch.max()
            if patchMax == 2:
                currentPixels.append((r, c))
                finalEdges[r, c] = 1

    # Extend strong edges based on current pixels
    while len(currentPixels) > 0:
        newPix = []
        for r, c in currentPixels:
            for dr in range(-1, 2):
```

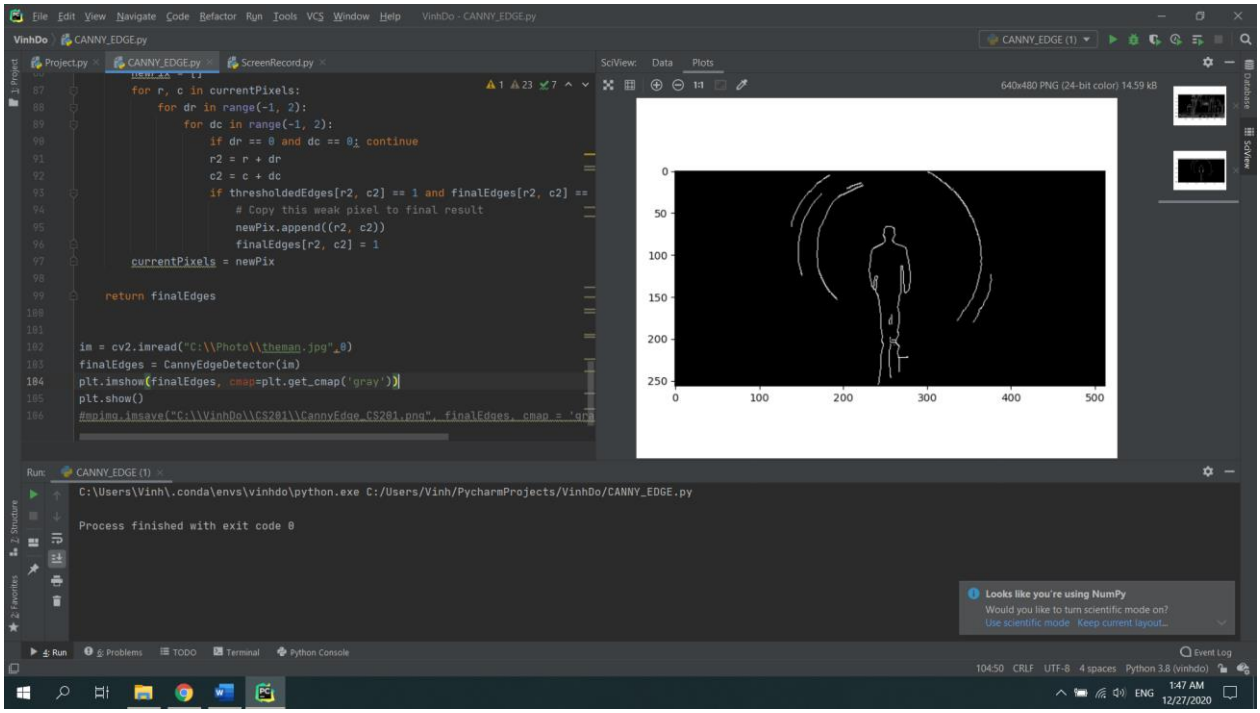
```
                for dc in range(-1, 2):
                    if dr == 0 and dc == 0: continue
                    r2 = r + dr
                    c2 = c + dc
                    if thresholdedEdges[r2, c2] == 1 and finalEdges[r2, c2]
== 0:
                        # Copy this weak pixel to final result
                        newPix.append((r2, c2))
                        finalEdges[r2, c2] = 1
        currentPixels = newPix

    return finalEdges

im = cv2.imread("C:\\Photo\\theman.jpg",0)
finalEdges = CannyEdgeDetector(im)
plt.imshow(finalEdges, cmap=plt.get_cmap('gray'))
plt.show()
# if you wanna to save picture which is named finalEdges in any Folder.
#mpimg.imsave("C:\\VinhDo\\CS201\\CannyEdge_CS201.png", finalEdges, cmap =
'gray')
```

III. KẾT QUẢ VÀ NHẬN XÉT

1. Sử dụng thư viện OpenCv có sẵn hàm cv2.Canny() cho ra kết quả



2. Viết lại hàm theo thuật toán cho ra kết quả.



Nhận xét: Dù bất kể thay đổi cách tiếp cận như thế nào thì việc đáp ứng của kết quả cả hai đều khá giống nhau. Nếu sử dụng thư viện thì hàm sẽ trả về cho ta ở dạng đơn giản và dễ load và view hơn, bên cạnh đó còn dùng để linking với các thư viện GUI mà nhóm tìm hiểu và xây dựng bonus thêm cho Bài tập lớn lần này. Với việc tìm hiểu và linking chúng lại với nhau thì việc hiển thị, giao diện người dùng cũng có thể sử dụng được trên ngôn ngữ Python nhưng cũng có một số nhược điểm như hiển thị ảnh phải ở dạng đồng nhất với dạng mà thư viện PySimpleGUI hỗ trợ được.

Nếu có những khó khan trong việc xem code thì hãy truy cập vào Github mà nhóm đã upload lên: ***https://github.com/vinhhoangdo/HCMUT-CS201***

Nhóm xin chân thành cảm ơn Thầy và các bạn!

TÀI LIỆU THAM KHẢO

1. https://pysimplegui.readthedocs.io/en/latest/

2. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

3. https://tinyurl.com/y8beee9z (Wikipedia Canny Edge Detection Algorithm).

4. http://justin-liang.com/tutorials/canny/

5. https://tinyurl.com/vhujehe

6. https://www.tutorialspoint.com/scipy/index.htm

7. *Slide bài giảng Nhập môn thị giác máy*