**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO NỘI DUNG THỰC HÀNH MÔN XỬ LÝ ẢNH**

**Tên Sinh Viên:** Lê Thành Phương

**MSSV:** 59131949

**GVHD:** Nguyễn Đình Cường

**Nha Trang - 2019**

MỤC LỤC

[GIỚI THIỆU 3](#_Toc11093624)

[CHƯƠNG 1: CÀI ĐẶT OPENCV CHO C++ 4](#_Toc11093625)

[CHƯƠNG 2: CÀI ĐẶT OPENCV CHO PYTHON 18](#_Toc11093626)

[CHƯƠNG 3: CÁC BƯỚC LÀM ĐỐI VỚI CÁC FILE CHƯƠNG TRÌNH MINH HỌA 26](#_Toc11093627)

[3.1: Mở hình ảnh 26](#_Toc11093628)

[3.2: Mở video 27](#_Toc11093629)

[3.3: Mở camera 29](#_Toc11093630)

[3.4: Lọc ảnh bằng lọc Trung Bình 31](#_Toc11093631)

[3.5: Lọc ảnh bằng lọc Trung Vị 34](#_Toc11093632)

[3.6: Lọc ảnh bằng lọc Gauss 36](#_Toc11093633)

[3.7: Lọc ảnh bằng lọc Laplace 38](#_Toc11093634)

[3.8: Cân bằng ảnh bằng Histogram 41](#_Toc11093635)

[3.9: Dò biên ảnh bằng Sobel 44](#_Toc11093636)

[3.10: Phân vùng ảnh 46](#_Toc11093637)

[3.11: Nhị phân ảnh 49](#_Toc11093638)

[3.12: Biến đổi Hough 51](#_Toc11093639)

[3.13: Nhận diện mắt người 59](#_Toc11093640)

[3.14: Nhận diện khuôn mặt người 62](#_Toc11093641)

[3.15: Nhận diện bàn tay người 65](#_Toc11093642)

[3.16: Nhận diện con người 76](#_Toc11093643)

[3.17: Nhận dạng xe hơi 85](#_Toc11093644)

[3.18: Nhận dạng biển số xe 98](#_Toc11093645)

[3.19: Điều khiển chuột máy tính 103](#_Toc11093646)

[3.20 Nhập chữ bằng bàn phím qua hình ảnh camera 105](#_Toc11093647)

[CHƯƠNG 4: KẾT LUẬN 116](#_Toc11093648)

[TÀI LIỆU THAM KHẢO 117](#_Toc11093649)

# GIỚI THIỆU

Trong các ví dụ trong bài thực hành, em luôn hay dùng OpenCV cho môn Xử lý ảnh. Ngoài ra, chúng ta có thể thực hành Xử lý ảnh qua Matlab hoặc Octave. Trong OpenCV em chỉ đề cập ngôn ngữ C++ và Python, bên cạnh đó còn có các ngôn ngữ khác như Java, C#,… để tham khảo.

Nội dung đề cập sẽ luôn gắn liền với những buổi thực hành trên lớp với các bài tập cụ thể, chỉ đề cập đến mức tìm hiểu và giải quyết bài tập nên còn nhiều thiếu sót chỉ đến mức cơ bản chưa đạt mức chuyên sâu nâng cao.

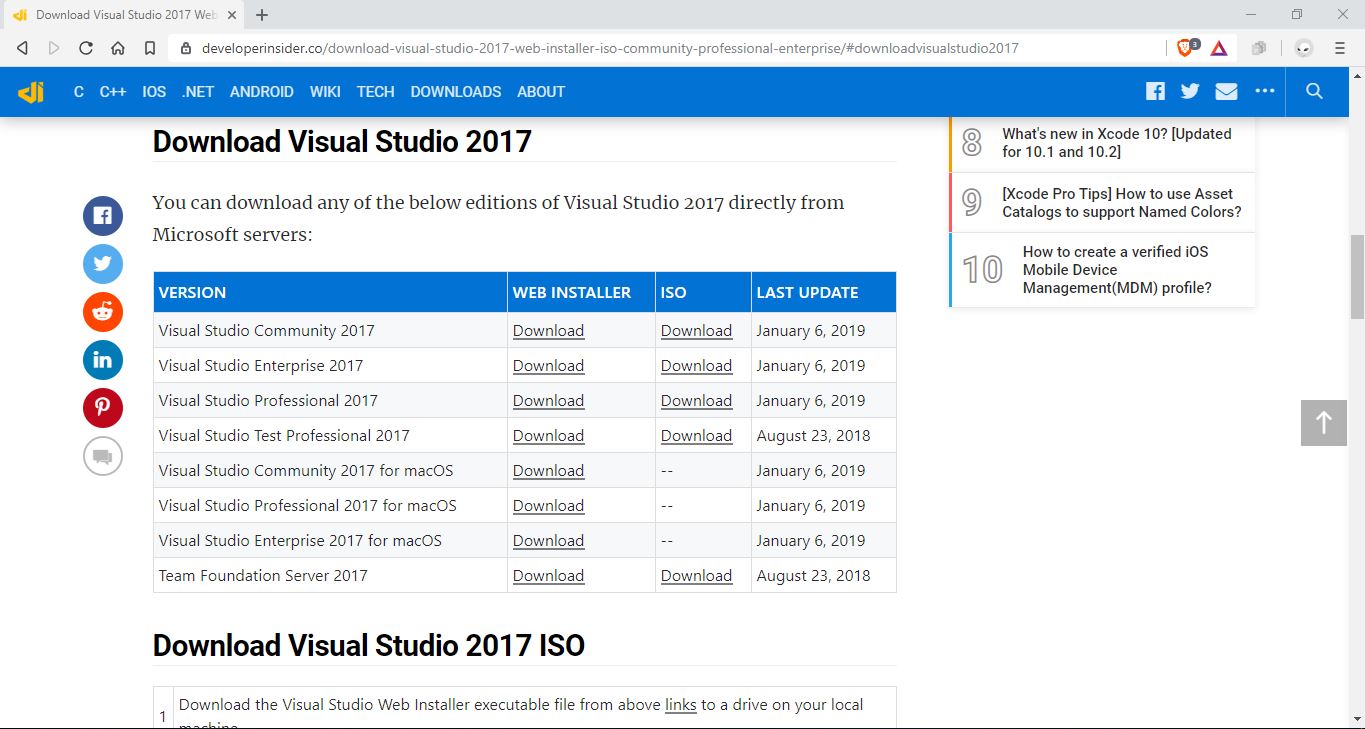
Mong Thầy và các bạn xem nhận xét về bài báo cáo của mình. Mình rất cần để có thể làm tốt hơn trong những bài báo cáo lần sau. Cảm ơn mọi người đang xem.



# CHƯƠNG 1: CÀI ĐẶT OPENCV CHO C++

**BƯỚC 1:** Mình sử dụng Visual studio 2017 nên ban đầu ta cài đặt Visual studio 2017 phiên bản Community trên trang web vì bản này là miễn phí, không tốn tiền.

“<https://developerinsider.co/download-visual-studio-2017-web-installer-iso-community-professional-enterprise/#downloadvisualstudio2017>”

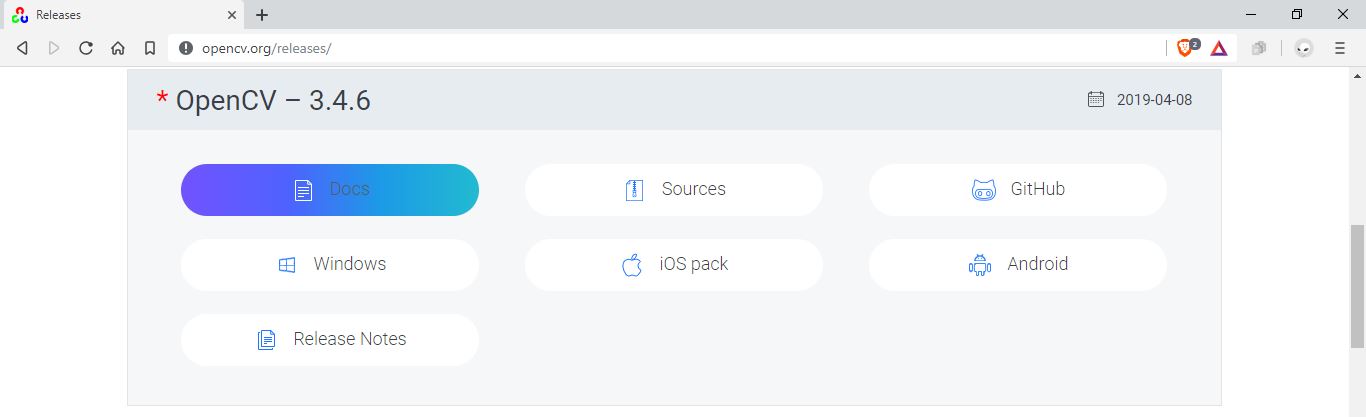


Cụ thể là chỗ này ta chọn Download trong bảng Web INSTALLER vì ta chọn cài đặt liền, file cài đặt sẽ là file .exe

Ta cứ cài đặt bình thường như các ứng dụng khác. Mở file Visual Studio Installer sau khi cài xong ta sẽ có phần mềm Visual Studio 2017 (Community )

**BƯỚC 2:** Ta tải OpenCV trên trang chủ OpenCV “<https://opencv.org/>”, rồi ta chọn “RELEASE” mình thì chọn phiên bản 3.4.6. Rồi mình muốn chạy trên Windows nên ta chọn “WIN”

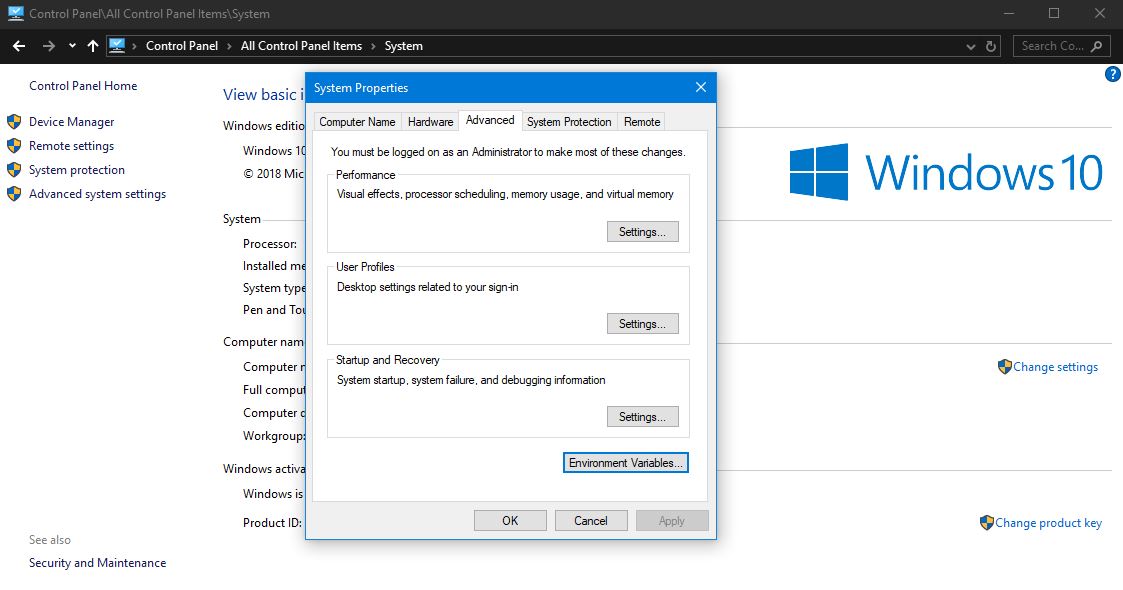


Và ta có link tải sau : “<https://sourceforge.net/projects/opencvlibrary/files/3.4.6/opencv-3.4.6-vc14_vc15.exe/download>”

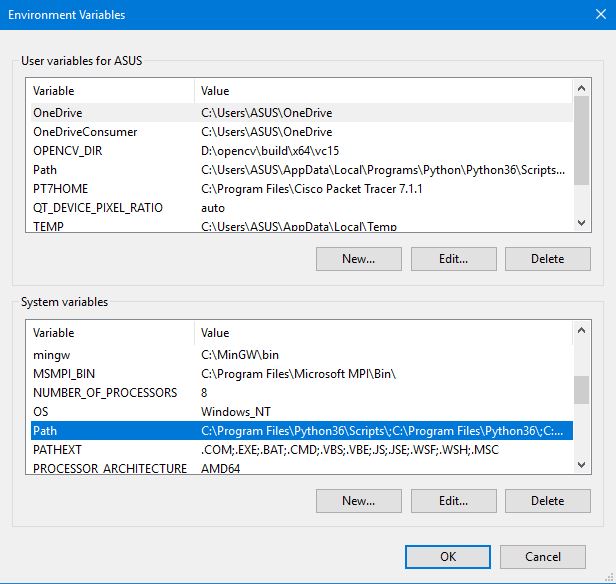
Tải xong ta được file .exe sau



**BƯỚC 3:** Ta click chuột vào “Control Panel” chọn “All Control Panel Items”-> “System” -> “Advanced system settings” được bảng “System Properties”

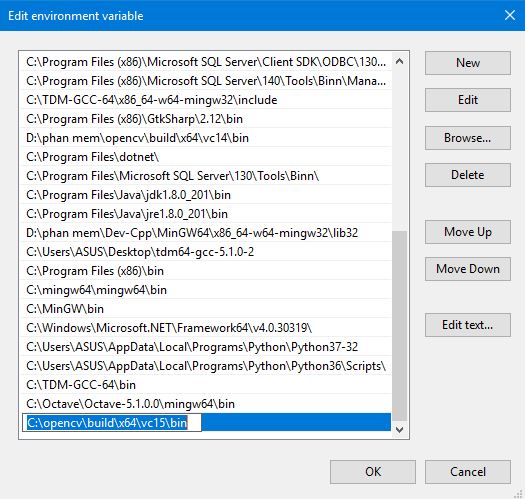


Trong thư mục “Advanced” ta chọn “Environment Variables…” ở thư mục “System variavles” ta chọn hàng “Path” rồi nhấn “New…” (hoặc double-click ở Path)

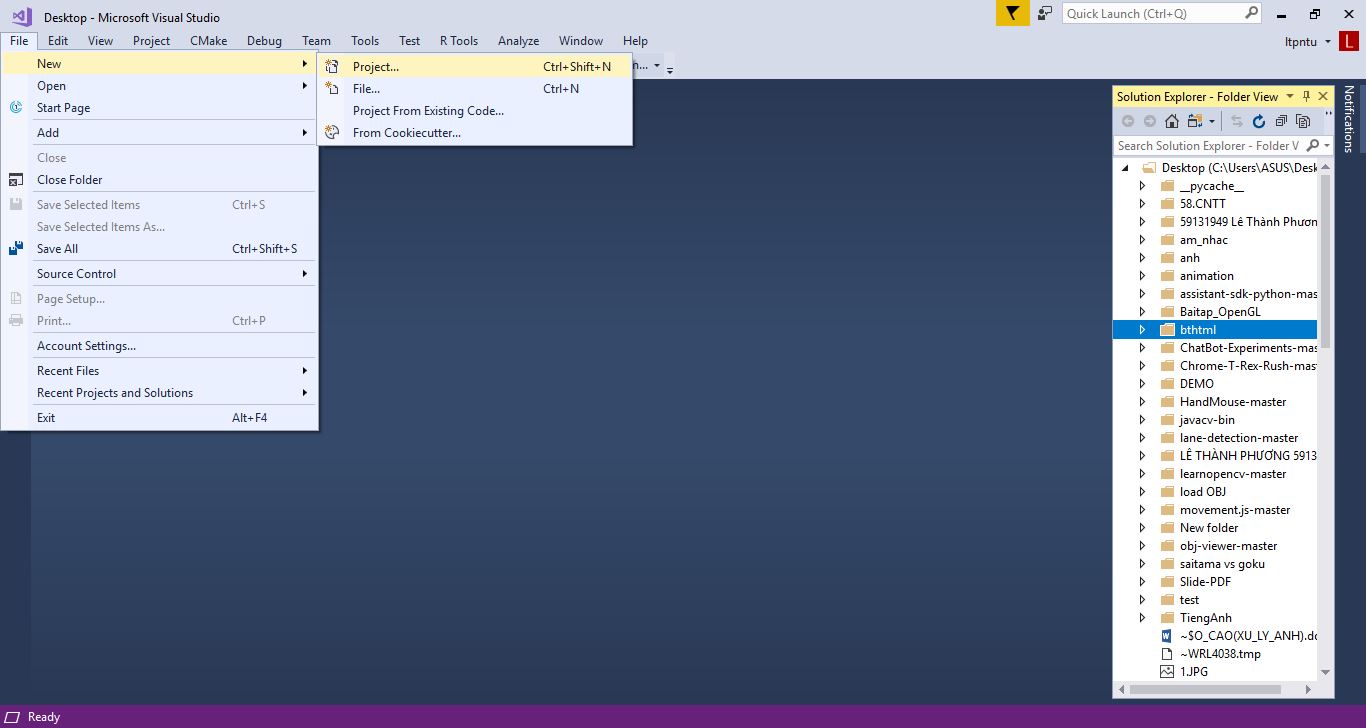


Sau đó, ta được bảng “Edit environment variable” a chọn “New” rồi nhập “C:\opencv\build\x64\vc15\bin” để có chương trình chạy OpenCV. Rồi nhấn phím

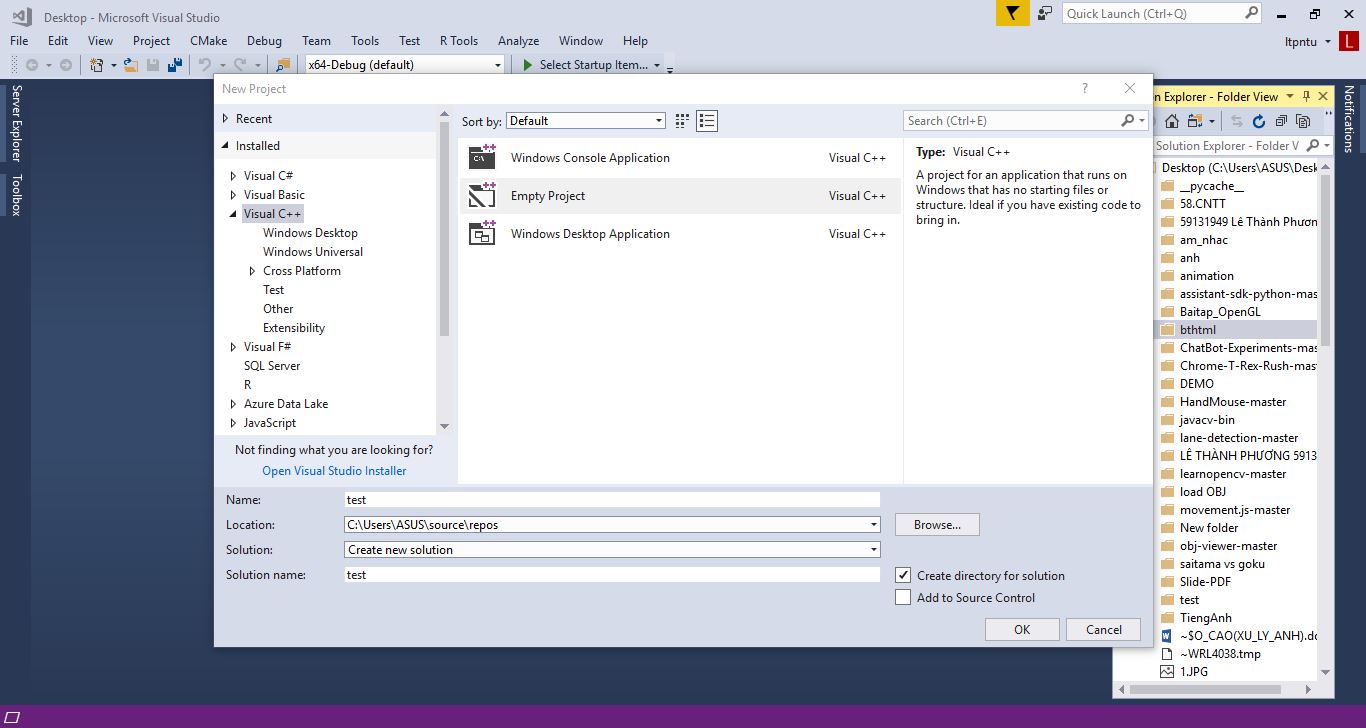
“Enter” rồi chọn “OK”ở tất cả các bảng



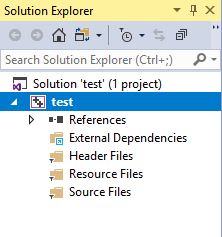
**BƯỚC 4:** Để tạo Project OpenCV dùng C++ ta mở Visual Studio 2017, tạo Project bằng cách chọn File->New->Project

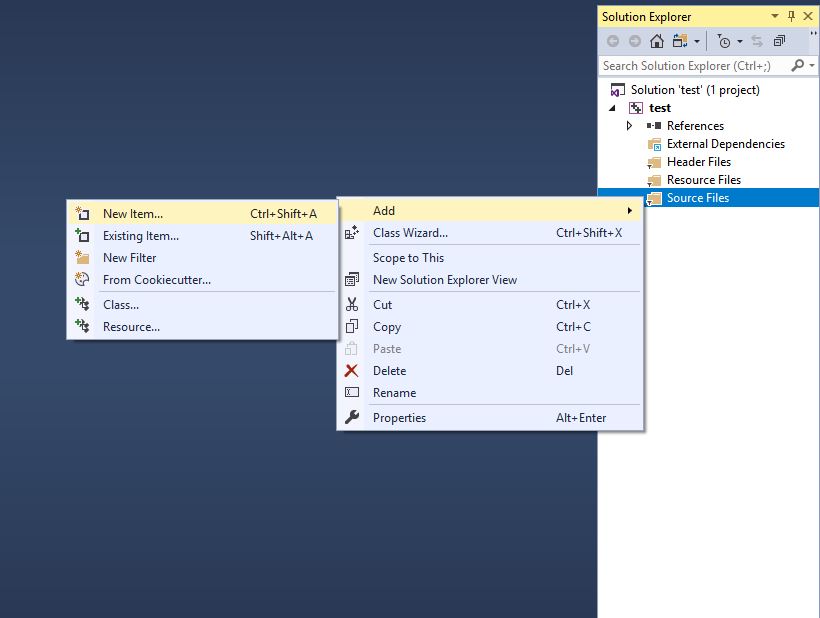


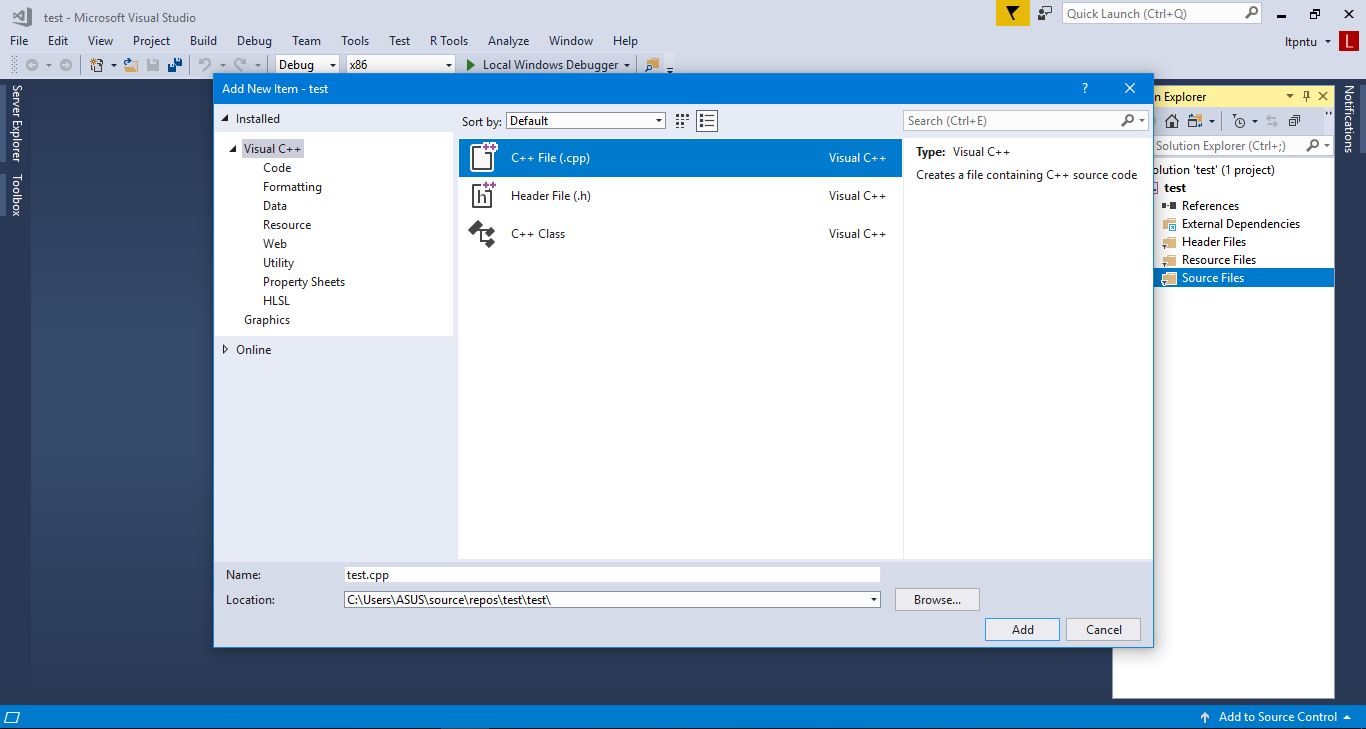
Ở đây mình đặt tên là “test”, lưu với địa chỉ thư mục “C:\Users\ASUS\source\repos”



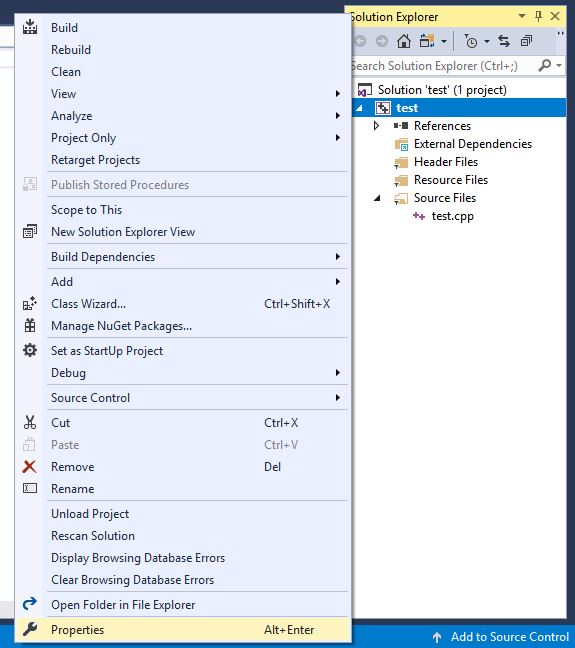
Tạo File xong ta được:



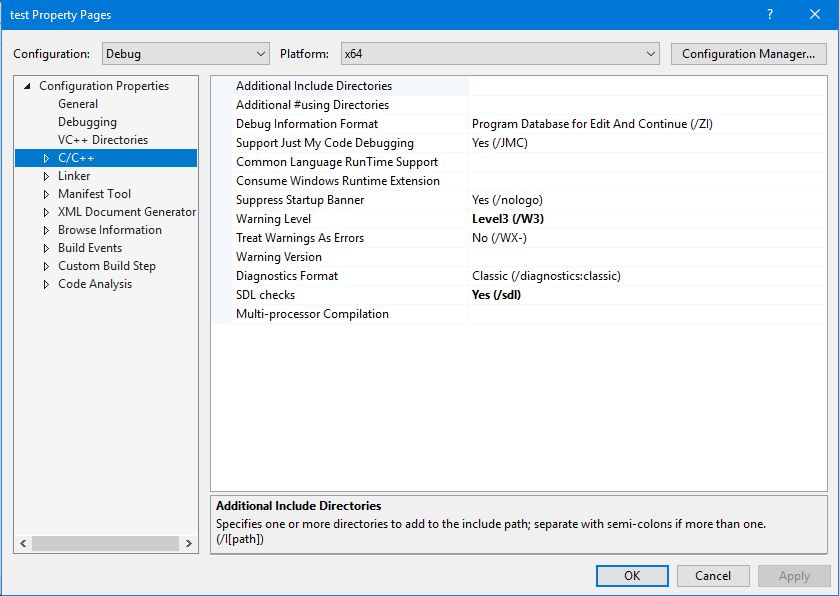
**BƯỚC 5:** Ta tạo File C++ bằng cách chọn chuột phải vào phần ” Source Files”, chọn Add ->New Item

Ta chon “C++ File(.cpp)” rồi đặt tên ví dụ như test.cpp rồi nhấn Add

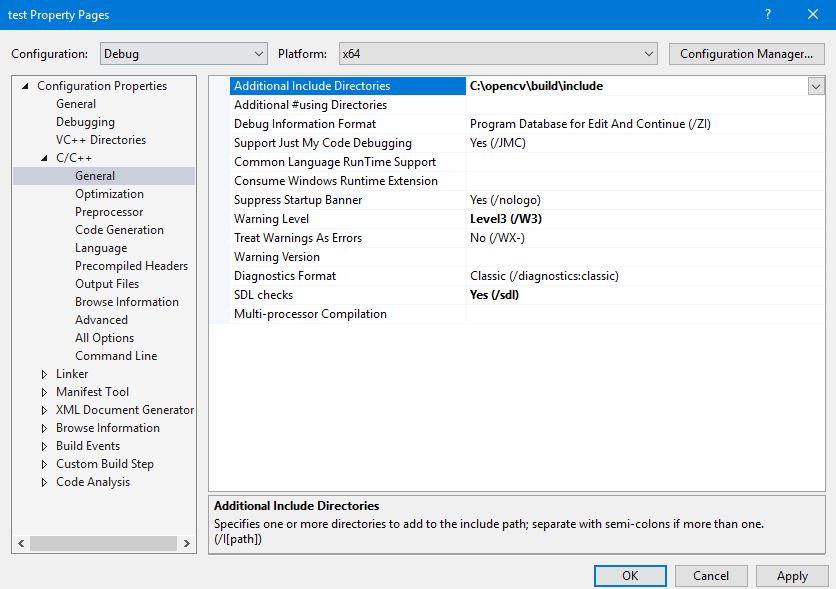
Tiếp theo, ta upload thư viện OpenCV vào Visual Studio 2017 như sau ta click chuột phải vào Project “test” chọn Properties



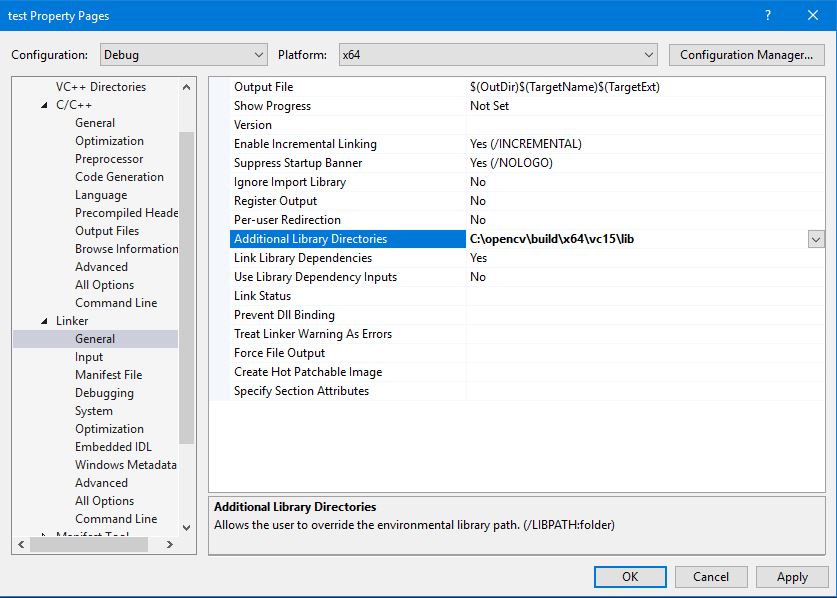
Rồi ta được bảng sau:

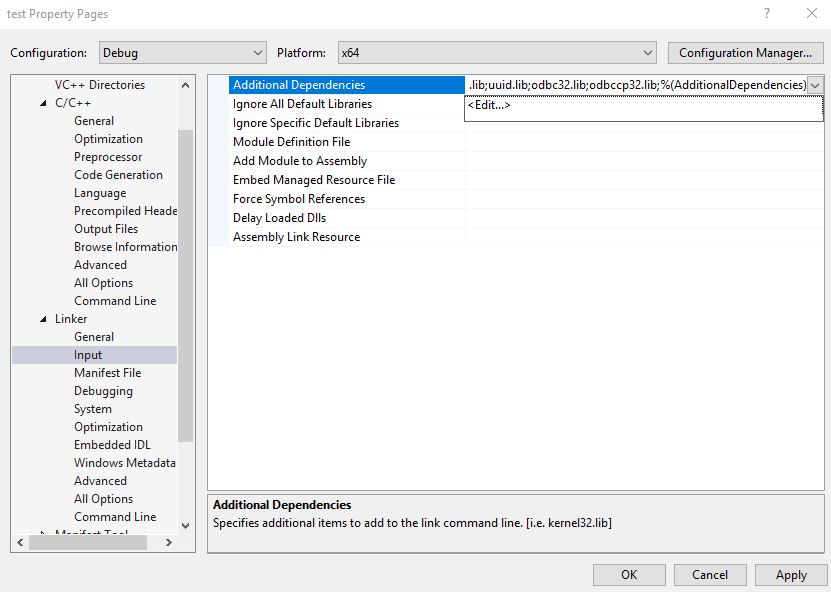


Trong phần ”C/C++” ta chọn ”General” trong bảng ”Additional Include Directories” ta nhập “C:\opencv\build\include “

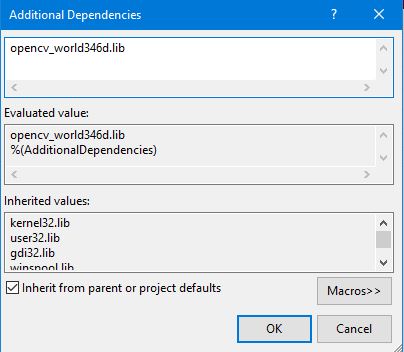


Trong phần ”Linker” ta chọn”Genaral” trong bảng ” Additional Library Directories” ta nhập “C:\opencv\build\x64\vc15\lib” vì máy mình là 64bit và là phiên bản Visual Studio 2017 trở lên nên là vc15 không phải vc14.



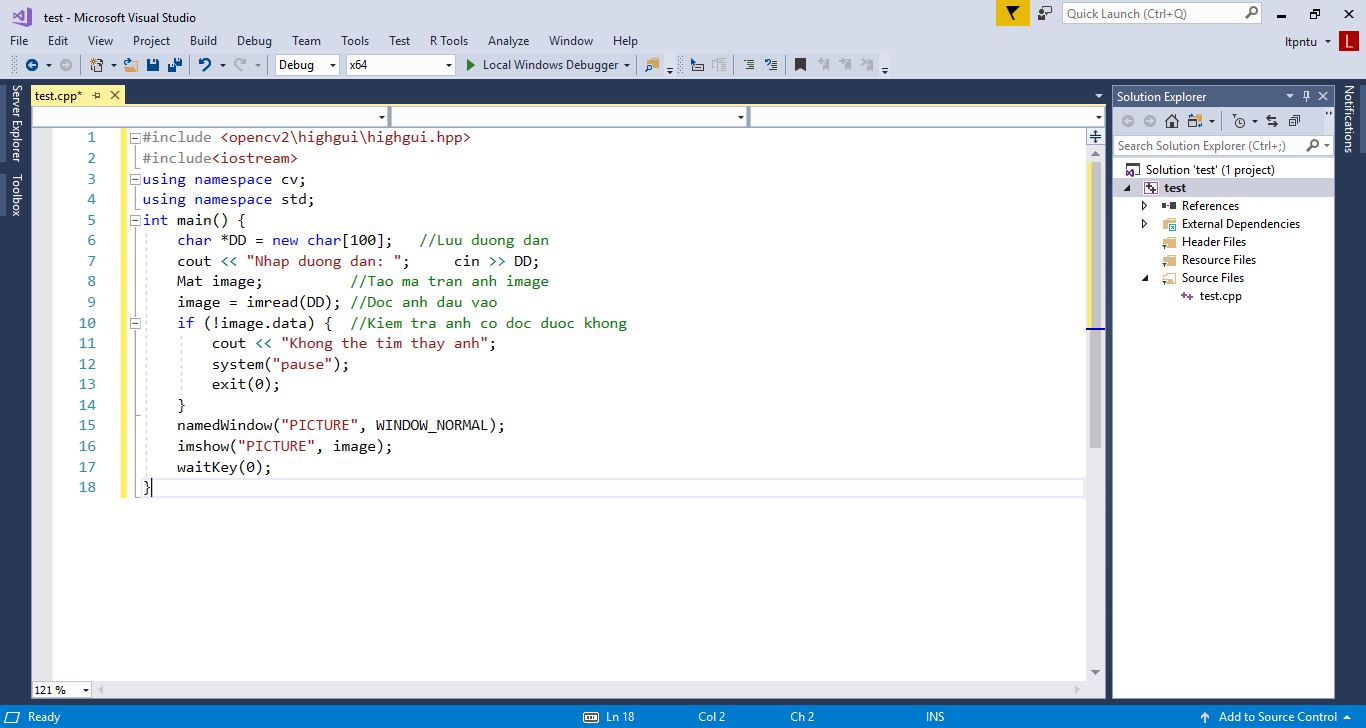
Còn trong “Input” trong bảng ”Additional Dependencies” ta làm như sau: 

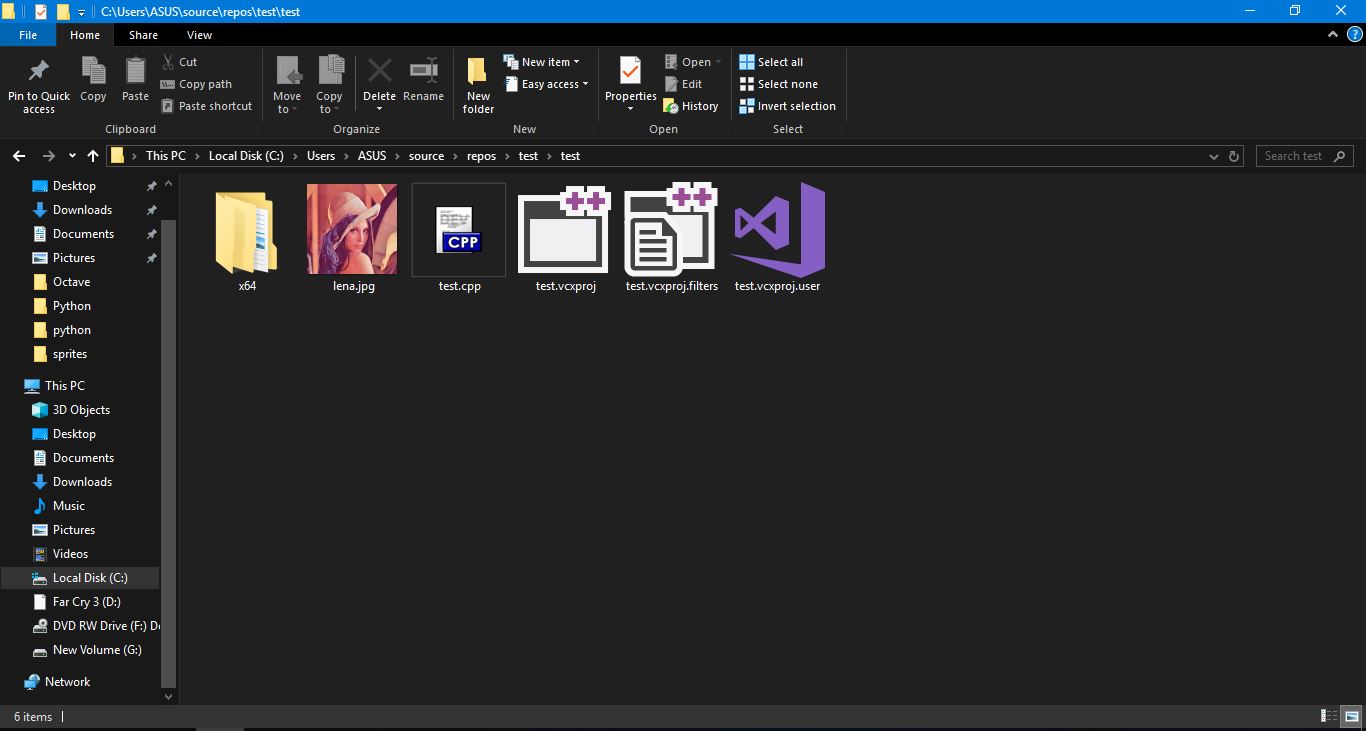
Chọn “Edit” ta có bảng sau, ta nhập ” opencv\_world346d.lib” vào bảng này . ta không nhập ” opencv\_world346.lib” vì ta muốn chạy chương trình bằng “Debug” với phiên bản “x64”



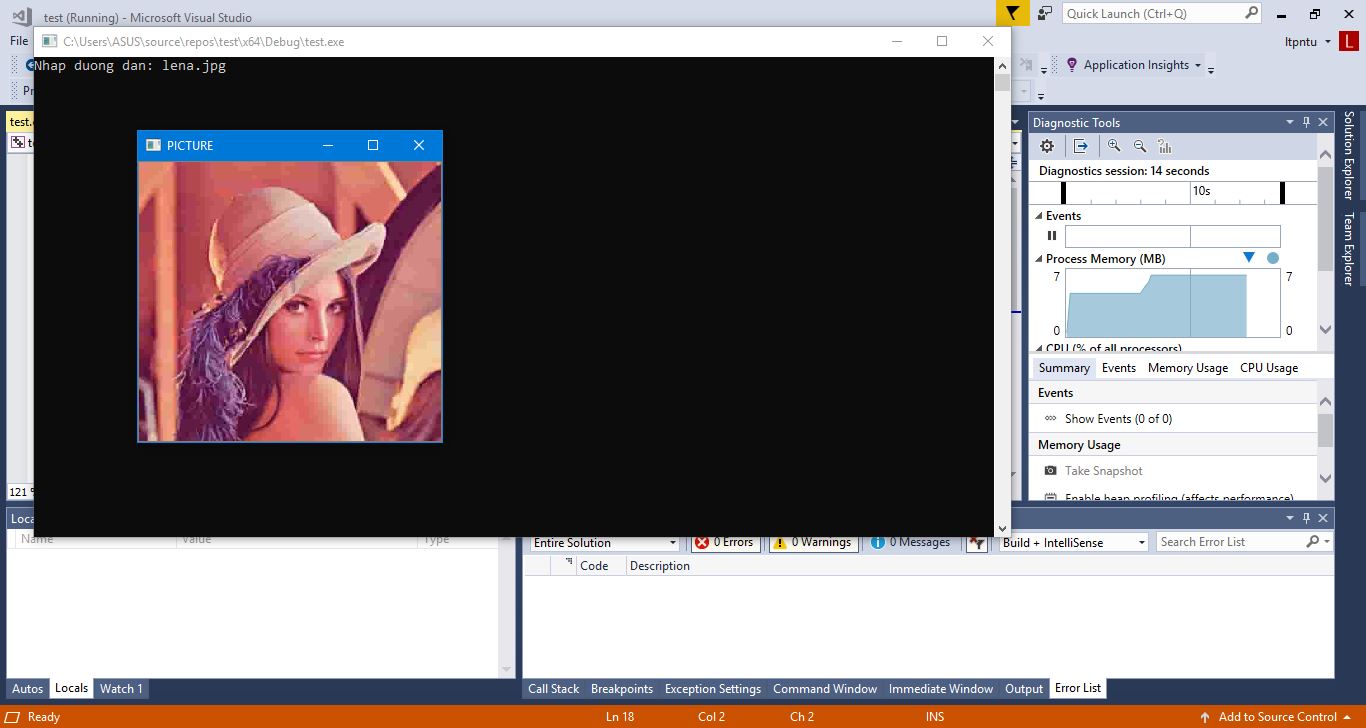
Nhấn “OK” rồi nhấn ”APPLY” trong bảng ”Additional Dependencies” và chọn ”OK” trong bảng ”test Property Pages”.

**BƯỚC 6:** Ta viết chương trình cpp vào Visual Studio 2017 như sau:

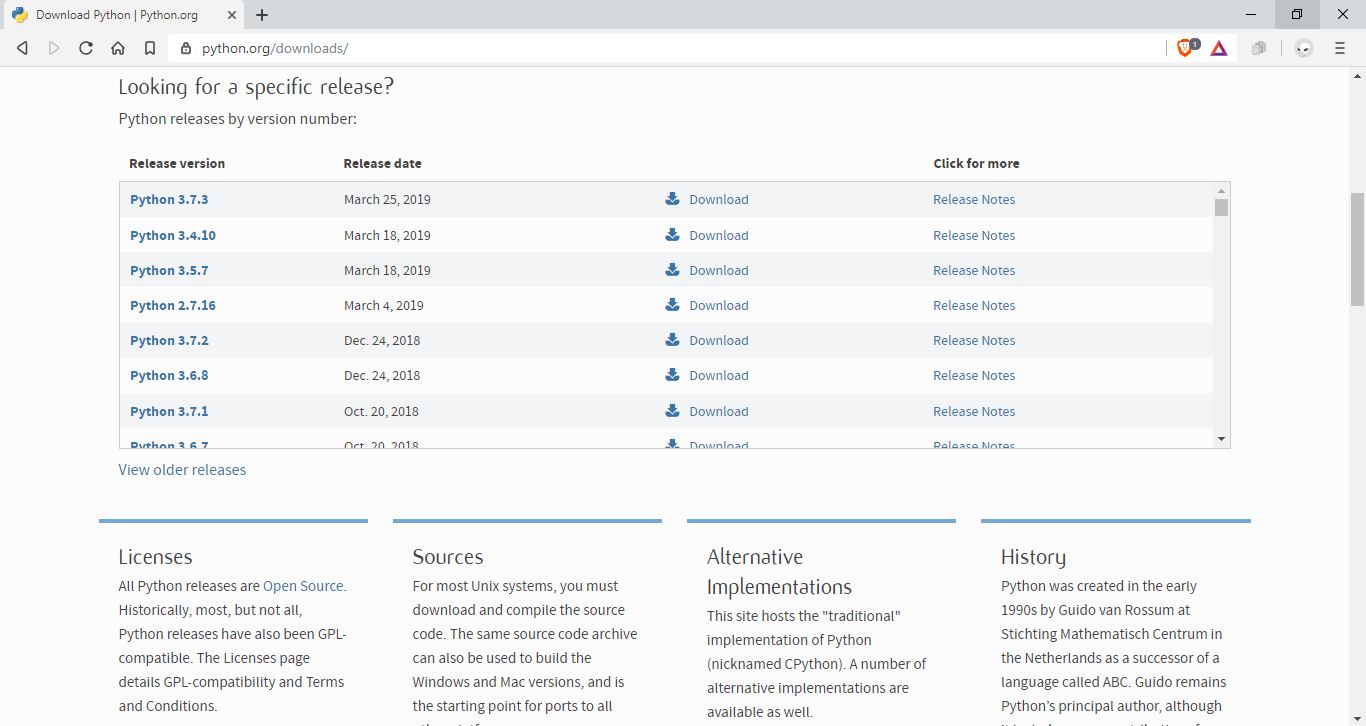


Ta tải 1 File hình ảnh có tên “lena.jpg” rồi chuyển vào File hình ảnh này vào trong Project

Rồi ta chạy chương trình, ta chọn ”Local Windows Debugger” hoặc nhấn phím “F5”. Chương trình yêu cầu nhập đường dẫn, ta nhập “lena.jpg” rồi ta nhấn phím Enter ta được như sau:



# CHƯƠNG 2: CÀI ĐẶT OPENCV CHO PYTHON

**BƯỚC 1:** Ta tải Python trên trang chủ “<https://www.python.org/downloads/>”

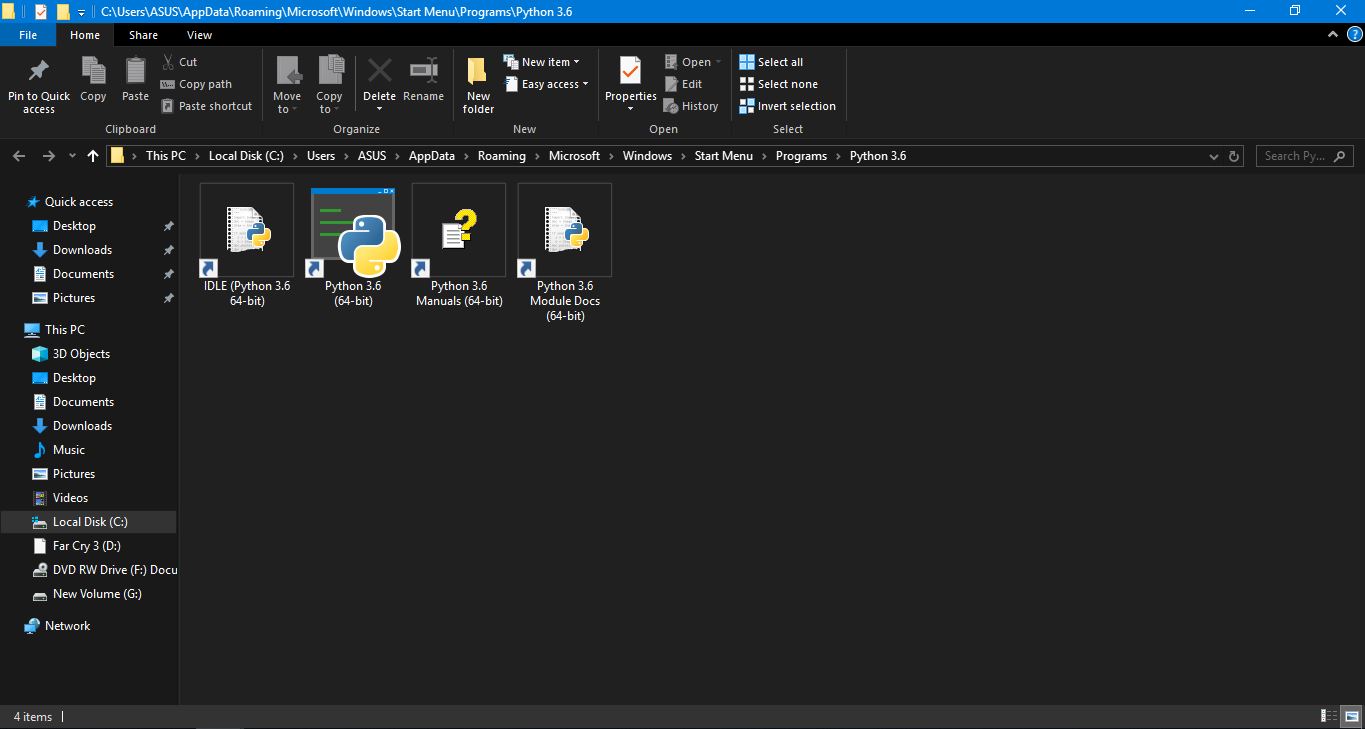
Cụ thể là ở đây:



Sau đó ta thấy File tải về như sau:

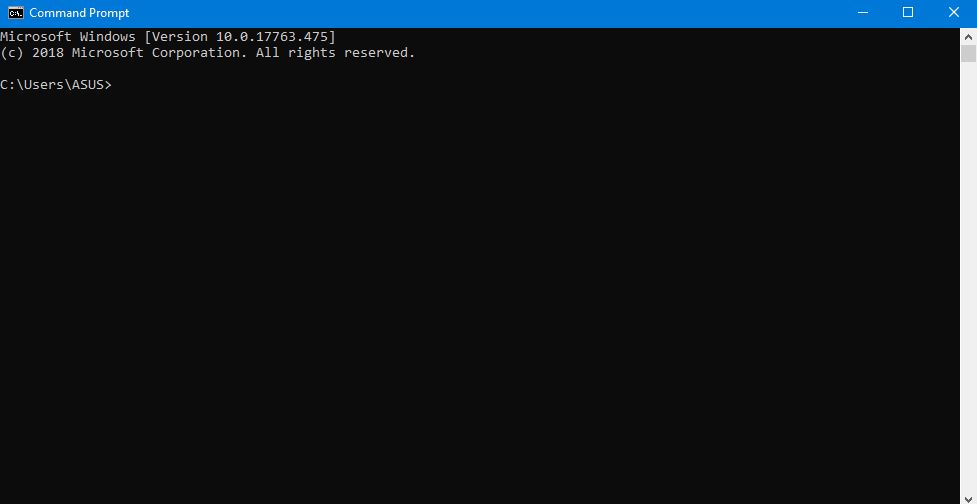


Rồi cài đặt như thường, ta được như sau:

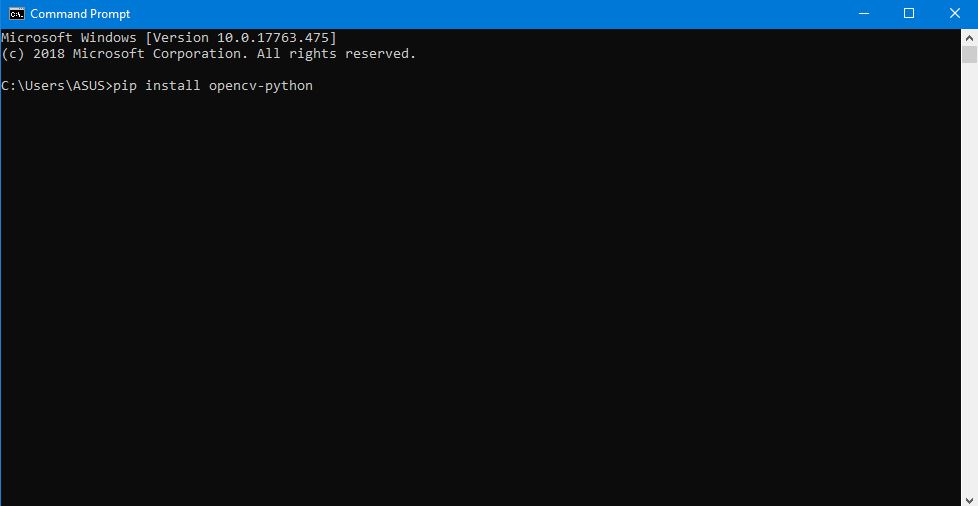


**BƯỚC 2:** Ta cài OpenCV cho Python, gồm các module “cv2, numpy” là bắt buộc cần có nhưng ngoài ra vì tất cả các File chương trình mình còn them các thư viện module khác như “future, argparse, matplotlib”

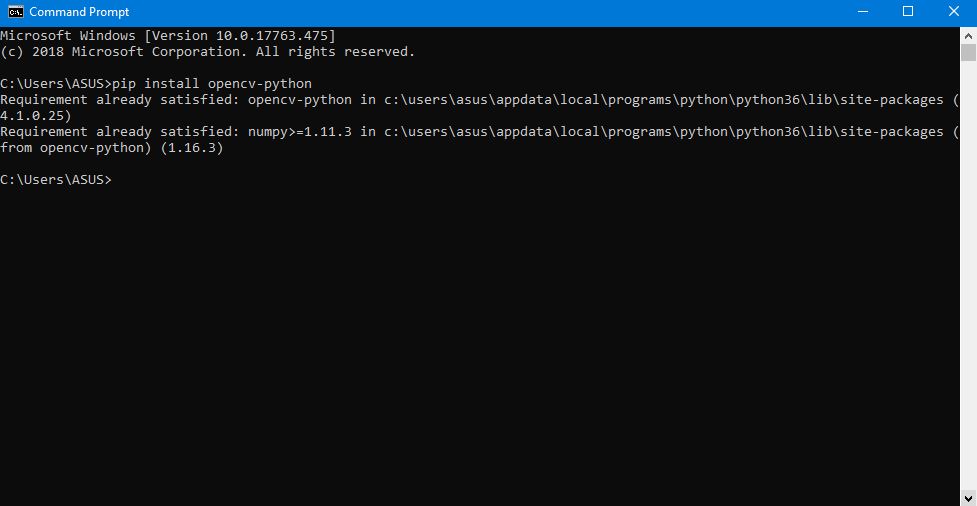
Ta mở Command Prompt (cmd) trên Windows



Ta muốn tải thư viện dùng PIP của Python, Vd: ta muốn tải opencv ta nhập “pip install opencv-python” như sau:



Máy sẽ tự động tải phiên bản mới nhất cho Python là OpenCV 4.0.1 vậy là tải xong 2 module “cv2 và numpy”



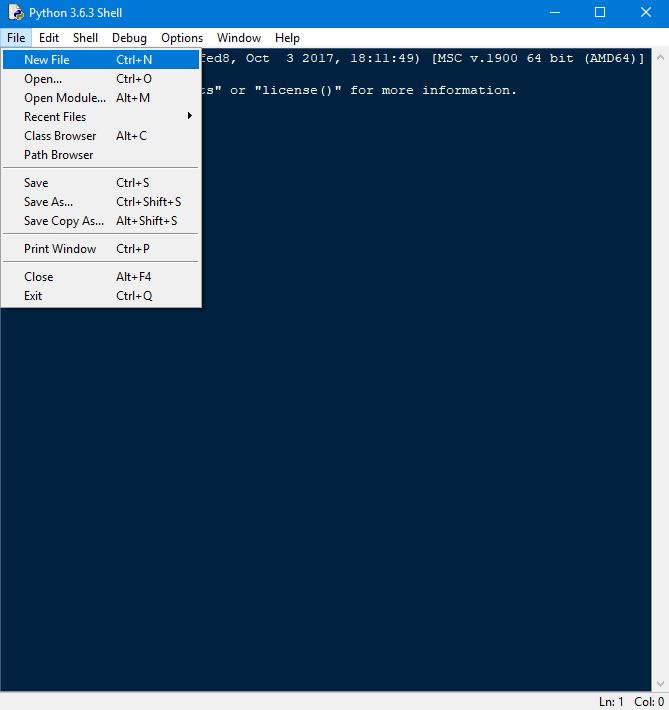
Còn các module khác thứ tự như sau:

argparse là “pip install argparse”

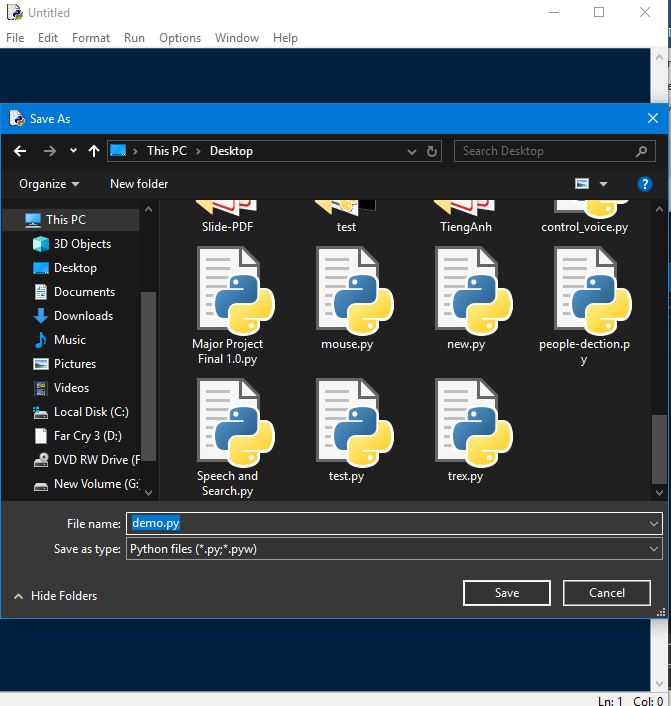
matplotlib là ”pip install matplotlib”

future là “pip install future”

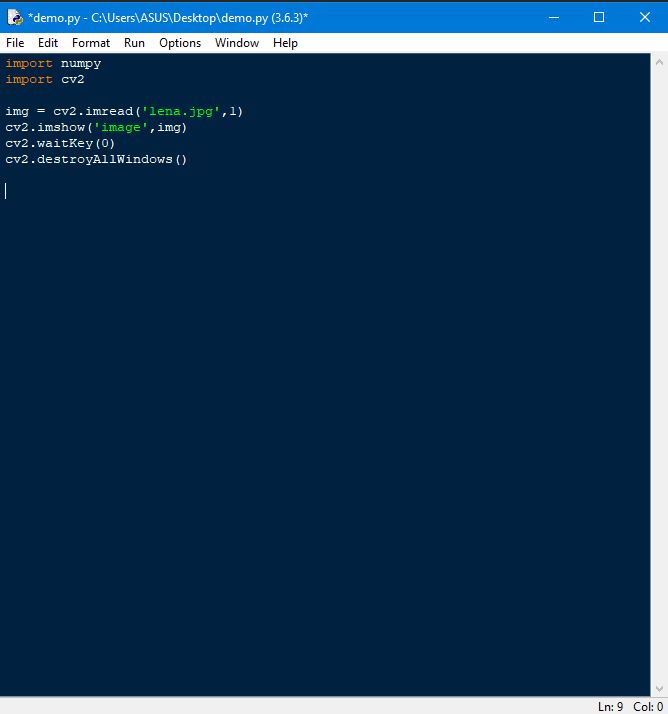
**BƯỚC 3:** Ta chạy thử chương trình đầu tiên, ta mở “IDLE (Python 3.6 64 bit)”



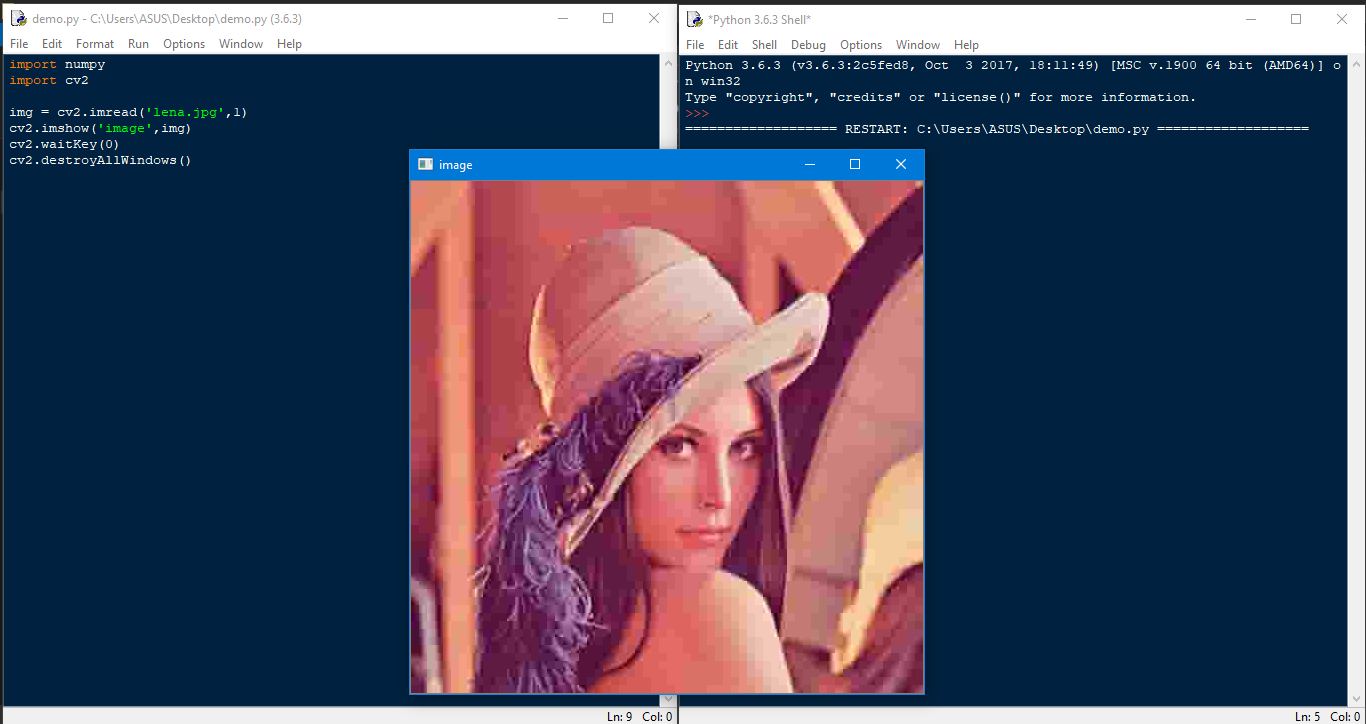
Ta lưu File với tên “demo.py” lưu ở thư mục “ảnh” chẳng hạn:



Ta viết chương trình cho Python:



Rồi nhấn phím F5 hoặc chọn thư mục Run->Run Module để chạy chương trình như sau:



# CHƯƠNG 3: CÁC BƯỚC LÀM ĐỐI VỚI CÁC FILE CHƯƠNG TRÌNH MINH HỌA

## 3.1: Mở hình ảnh

**C++**

#include <opencv2\highgui\highgui.hpp>

#include<iostream>

using namespace cv;

using namespace std;

int main() {

char \*DD = new char[100]; //Lưu đường dẫn

cout << "Nhap duong dan: "; cin >> DD;

Mat image; //Tạo ma trận ảnh tên image

image = imread(DD); //Đọc ảnh đầu vào

if (!image.data) { //Kiểm tra ảnh có đọc được không

cout << "Khong the tim thay anh";

system("pause");

exit(0);

}

namedWindow("PICTURE", WINDOW\_NORMAL); // Tạo một cửa sổ OpenCV có tên PICTURE

imshow("PICTURE", image); // Hiển thị hình ảnh trong cửa sổ OpenCV

waitKey(0);

}

**PYTHON**

import numpy as np

import cv2

img = cv2.imread('lena.jpg',1) // Tải một hình ảnh từ một tập tin với 1 là dạng màu sắc nguyên bản, nếu là 0 thì sẽ là ảnh xám

cv2.imshow('image',img) // Hiển thị một hình ảnh trong cửa sổ chỉ định với tên image

cv2.waitKey(0) ///Hàm chức năng ràng buộc bàn phím, vì ta nhập là 0 nên hình ảnh sẽ hiển thị vô thời hạn ,còn bạn muốn hình biến mất trong khoảng thời gian nào thì chỉnh số 0 thành số tùy ý

cv2.destroyAllWindows() // là hàm phá hủy tất cả các cửa sổ mà bạn đã tạo ra bằng bàn phím hoặc chuột

## 3.2: Mở video

**C++**

#include <opencv2/opencv.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main() {

VideoCapture cap("road\_car\_view.mp4");// Tạo một đối tượng VideoCapture và mở tệp đầu vào. Nếu đầu vào là camera web, hãy chuyển 0 thay vì tên tệp video

if (!cap.isOpened()) {// Kiểm tra xem camera đã mở thành công chưa

cout << "Error opening video stream or file" << endl;

return -1;

}

while (1) {

Mat frame; // Chụp từng khung hình

cap >> frame;

if (frame.empty()) // Nếu khung trống, ngắt ngay lập tức

break;

imshow("Video", frame); // Hiển thị khung kết quả

char c = (char)waitKey(25); // Nhấn ESC trên bàn phím để thoát

if (c == 27)

break;

}

cap.release(); // Khi mọi thứ đã hoàn tất, hãy giải phóng đối tượng quay video

destroyAllWindows(); // Đóng tất cả các khung

return 0;

}

**PYTHON**

import cv2

import numpy as np

cap = cv2.VideoCapture('road\_car\_view.mp4') # Tạo một đối tượng VideoCapture và đọc từ tệp đầu vào. Nếu đầu vào là máy ảnh, hãy chuyển 0 thay vì tên tệp video

if (cap.isOpened()== False): # Kiểm tra xem camera đã mở thành công chưa

print("Error opening video stream or file")

while(cap.isOpened()): # Đọc cho đến khi video được hoàn thành

ret, frame = cap.read() # Chụp từng khung hình

if ret == True:

cv2.imshow('Frame',frame) # Hiển thị khung kết quả

if cv2.waitKey(25) & 0xFF == ord('q'): #Nhấn Q trên bàn phím để thoát

break

else:

break

cap.release() # Khi mọi thứ đã hoàn tất, hãy giải phóng đối tượng quay video

cv2.destroyAllWindows() # Đóng tất cả các khung

## 3.3: Mở camera

**C++**

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>

using namespace cv;

using namespace std;

int main() {

VideoCapture stream1(0); // mở camera mặc định

if (!stream1.isOpened()) { // kiểm tra xem đã thành công chưa

cout << "cannot open camera";

}

//unconditional loop

while (true) { // lấy khung hình mới từ camera

Mat cameraFrame;

stream1.read(cameraFrame);

imshow("camera", cameraFrame);

if (waitKey(30) >= 0)

break;

}

return 0; // máy ảnh sẽ được tự động trong bộ hủy VideoCapture

}

**PYTHON**

import cv2

def save\_webcam(outPath,fps,mirror=False):

cap = cv2.VideoCapture(0) # Quay video từ webcam

currentFrame = 0

width = cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH) # Nhận chiều rộng hiện tại của khung

height = cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT) # Nhận chiều cao hiện tại của khung

fourcc = cv2.VideoWriter\_fourcc(\*"XVID") # Xác định codec và tạo đối tượng VideoWriter

out = cv2.VideoWriter(outPath, fourcc, fps, (int(width), int(height)))

while (cap.isOpened()):

ret, frame = cap.read() # Chụp từng khung hình

if ret == True:

if mirror == True:

frame = cv2.flip(frame, 1) # Phản chiếu khung hình video đầu ra

out.write(frame) # Lưu cho video

cv2.imshow('camera', frame) # Hiển thị khung kết quả

else:

break

if cv2.waitKey(1) & 0xFF == ord('q'): # Nhấn phím ’q’ để thoát

break

currentFrame += 1 # Để dừng hình ảnh trùng lặp

cap.release()

out.release() # Khi mọi thứ đã hoàn tất, hãy giải phóng bản chụp

cv2.destroyAllWindows()

def main():

save\_webcam('mouthwash.avi', 5.0,mirror=True)

if \_\_name\_\_ == '\_\_main\_\_':

main()

## 3.4: Lọc ảnh bằng lọc Trung Bình

**C++**

#include<iostream>

#include<opencv2/imgproc/imgproc.hpp>

#include<opencv2/highgui/highgui.hpp>

using namespace std;

using namespace cv;

int main()

{

Mat src, dst;

float sum;

src = imread("lena.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE); //CV\_LOAD\_IMAGE\_GRAYSCALE là ảnh xám

if (!src.data)

{

return -1;

}

// xác định kernel với kernel là dùng 1 cửa sổ để trượt trên mỗi pixel của ảnh

float Kernel[3][3] = {

{1 / 9.0, 1 / 9.0, 1 / 9.0},

{1 / 9.0, 1 / 9.0, 1 / 9.0},

{1 / 9.0, 1 / 9.0, 1 / 9.0}

};

dst = src.clone();

for (int y = 0; y < src.rows; y++)

for (int x = 0; x < src.cols; x++)

dst.at<uchar>(y, x) = 0.0;

// hoạt động chập với từng hàng và cột

for (int y = 1; y < src.rows - 1; y++) {

for (int x = 1; x < src.cols - 1; x++) {

sum = 0.0;

for (int k = -1; k <= 1; k++) {

for (int j = -1; j <= 1; j++) {

sum = sum + Kernel[j + 1][k + 1] \*src.at<uchar>(y - j, x - k);

}

}

dst.at<uchar>(y, x) = sum;

}

}

namedWindow("final");

imshow("final", dst);

namedWindow("average filter");

imshow("average filter", src);

waitKey();

return 0;

}

**PYTHON**

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('lena.jpg')

blur = cv2.blur(img,(5,5)) # Dùng để lấy trung bình tất cả các điểm pixel của kernel với (5,5) là kiểu ma trận pixel 5x5 với hàm blur

plt.subplot(121),plt.imshow(img),plt.title('ảnh gốc')

plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(blur),plt.title('ảnh lọc trung bình')

plt.xticks([]), plt.yticks([])

plt.show()# Là các câu lệnh hiển thị hình ảnh khác hàm gọi là do mình nhập thư viện pyplot(thư viện có thể thay đổi đồ thị như Matlab)

## 3.5: Lọc ảnh bằng lọc Trung Vị

**C++**

#include<iostream>

#include<opencv2/imgproc/imgproc.hpp>

#include<opencv2/highgui/highgui.hpp>

using namespace std;

using namespace cv;

// sắp xếp cửa sổ bằng cách sử dụng sắp xếp chèn

// sắp xếp chèn là tốt nhất cho việc sắp xếp này

void insertionSort(int window[])

{

int temp, i, j;

for (i = 0; i < 9; i++) {

temp = window[i];

for (j = i - 1; j >= 0 && temp < window[j]; j--) {

window[j + 1] = window[j];

}

window[j + 1] = temp;

}

}

int main()

{

Mat src, dst;

// Load an image

src = imread("lena.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE);

if (!src.data)

{

return -1;

}

// tạo một cửa sổ trượt có kích thước là 9

int window[9];

dst = src.clone();

for (int y = 0; y < src.rows; y++)

for (int x = 0; x < src.cols; x++)

dst.at<uchar>(y, x) = 0.0;

for (int y = 1; y < src.rows - 1; y++) {

for (int x = 1; x < src.cols - 1; x++) {

// Chọn phần tử cửa sổ

window[0] = src.at<uchar>(y - 1, x - 1);

window[1] = src.at<uchar>(y, x - 1);

window[2] = src.at<uchar>(y + 1, x - 1);

window[3] = src.at<uchar>(y - 1, x);

window[4] = src.at<uchar>(y, x);

window[5] = src.at<uchar>(y + 1, x);

window[6] = src.at<uchar>(y - 1, x + 1);

window[7] = src.at<uchar>(y, x + 1);

window[8] = src.at<uchar>(y + 1, x + 1);

insertionSort(window); // sắp xếp cửa sổ để tìm trung vị

dst.at<uchar>(y, x) = window[4]; // gán trung vị cho phần tử trung tâm của ma trận

}

}

namedWindow("final");

imshow("final", dst);

namedWindow("Median Filtering");

imshow("Median Filtering", src);

waitKey();

return 0;

}

**PYTHON**

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('lena.jpg')

median = cv2.medianBlur(img,1) #Ở đây, hàm cv2.medianBlur () lấy trung vị của tất cả các pixel trong vùng nhân và phần tử trung tâm được thay thế bằng giá trị trung bình này.

plt.subplot(121),plt.imshow(img),plt.title('ảnh gốc')

plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(median),plt.title('ảnh lọc trung vị')

plt.xticks([]), plt.yticks([])

plt.show() # Hiển thị ảnh như thường do mình dùng thư viện pyplot

## 3.6: Lọc ảnh bằng lọc Gauss

**C++**

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/opencv.hpp>

#include <opencv2/highgui/highgui\_c.h>

using namespace cv;

int main(int argc, char\*\* argv)

{

namedWindow("original", CV\_WINDOW\_AUTOSIZE);

// CV\_WINDOW\_AUTOSIZE là hàm tự động chọn kích thước khung hình

Mat src = imread("lena.jpg", 1);

Mat dst;

imshow("original", src);

for (int i = 1; i < 51; i = i + 2)

{

blur(src, dst, Size(i,i)); // làm mịn hình ảnh trong "src" và lưu nó vào "dst"

GaussianBlur(src, dst, Size(i, i), 0, 0); // Làm mịn Gaussian

imshow("Gaussian filter", dst); // Hiển thị hình ảnh mờ với văn bản

waitKey(5000); // chờ trong 5 giây

}

}

**PYTHON**

import cv2

import numpy

src = cv2.imread('lena.jpg', cv2.IMREAD\_UNCHANGED)

# áp dụng độ mờ gaussian trên ảnh src

dst = cv2.GaussianBlur(src,(5,5),cv2.BORDER\_DEFAULT)

# hiển thị hình ảnh đầu vào và đầu ra

cv2.imshow("Gaussian Smoothing",numpy.hstack((src, dst)))

cv2.waitKey(0) # waits until a key is pressed

cv2.destroyAllWindows()

## 3.7: Lọc ảnh bằng lọc Laplace

**C++**

#include "opencv2/imgproc.hpp"

#include "opencv2/imgcodecs.hpp"

#include "opencv2/highgui.hpp"

using namespace cv;

int main( int argc, char\*\* argv )

{

// Khai báo các biến chúng ta sẽ sử dụng

Mat src, src\_gray, dst;

int kernel\_size = 3;

int scale = 1;

int delta = 0;

int ddepth = CV\_16S;

const char\* window\_name = "Laplace Demo";

const char\* imageName = argc >=2 ? argv[1] : "../data/lena.jpg";

src = imread("lena.jpg", IMREAD\_COLOR );

// Kiểm tra xem hình ảnh có được tải tốt không

if(src.empty()){

printf(" Error opening image\n");

printf(" Program Arguments: [image\_name -- default ../data/lena.jpg] \n");

return -1;

}

// Giảm nhiễu bằng cách làm mờ bằng bộ lọc Gaussian (kích thước kernel = 3)

GaussianBlur( src, src, Size(3, 3), 0, 0, BORDER\_DEFAULT );

cvtColor( src, src\_gray, COLOR\_BGR2GRAY ); // Chuyển đổi hình ảnh sang ảnh có thang độ màu xám

Mat abs\_dst;

Laplacian( src\_gray, dst, ddepth, kernel\_size, scale, delta, BORDER\_DEFAULT );

// chuyển đổi trở lại CV\_8U (CV\_8U 8bit/pixel - tức là pixel có thể có giá trị 0-255, đây là phạm vi bình thường đối với hầu hết các định dạng hình ảnh và video)

convertScaleAbs( dst, abs\_dst );

imshow( window\_name, abs\_dst );

waitKey(0);

return 0;

}

**PYTHON**

import sys

import cv2 as cv

def main(argv):

    # Khai báo các biến chúng ta sẽ sử dụng

ddepth = cv.CV\_16S

kernel\_size = 3

window\_name = "Laplace Demo"

    # tải ảnh

imageName = argv[0] if len(argv) > 0 else 'lena.jpg'

src = cv.imread(cv.samples.findFile(imageName), cv.IMREAD\_COLOR)

# Kiểm tra xem hình ảnh có được tải tốt không

if src is None:

print ('Error opening image')

print ('Program Arguments: [image\_name -- default lena.jpg]')

return -1

    # Loại bỏ nhiễu bằng cách làm mờ bằng bộ lọc Gaussian

src = cv.GaussianBlur(src, (3, 3), 0)

    # Chuyển đổi hình ảnh sang thang độ xám

src\_gray = cv.cvtColor(src, cv.COLOR\_BGR2GRAY)

    # Tạo cửa sổ

cv.namedWindow(window\_name, cv.WINDOW\_AUTOSIZE)

    # Áp dụng hàm Laplace

dst = cv.Laplacian(src\_gray, ddepth, kernel\_size)

# chuyển đổi trở lại uint8

abs\_dst = cv.convertScaleAbs(dst)

cv.imshow(window\_name, abs\_dst)

cv.waitKey(0)

return 0

if \_\_name\_\_ == "\_\_main\_\_":

main(sys.argv[1:])

## 3.8: Cân bằng ảnh bằng Histogram

**C++**

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

#include <stdio.h>

using namespace std;

using namespace cv;

int main( int argc, char\*\* argv )

{

Mat src, dst;

src = imread("lena.jpg", 1 );

if( !src.data )

{ return -1; }

// Tách hình ảnh ở 3 vị trí (B, G và R)

vector<Mat> bgr\_planes;

split( src, bgr\_planes );

// Thiết lập số lượng thùng

int histSize = 256;

// Đặt phạm vi (cho B, G, R))

float range[] = { 0, 256 } ;

const float\* histRange = { range };

bool uniform = true; bool accumulate = false;

Mat b\_hist, g\_hist, r\_hist;

// Tính toán biểu đồ

calcHist( &bgr\_planes[0], 1, 0, Mat(), b\_hist, 1, &histSize, &histRange, uniform, accumulate );

calcHist( &bgr\_planes[1], 1, 0, Mat(), g\_hist, 1, &histSize, &histRange, uniform, accumulate );

calcHist( &bgr\_planes[2], 1, 0, Mat(), r\_hist, 1, &histSize, &histRange, uniform, accumulate );

// Vẽ biểu đồ cho B, G và R

int hist\_w = 512; int hist\_h = 400;

int bin\_w = cvRound( (double) hist\_w/histSize );

Mat histImage( hist\_h, hist\_w, CV\_8UC3, Scalar( 0,0,0) );

// Bình thường hóa kết quả thành [0, histImage.rows]

normalize(b\_hist, b\_hist, 0, histImage.rows, NORM\_MINMAX, -1, Mat() );

normalize(g\_hist, g\_hist, 0, histImage.rows, NORM\_MINMAX, -1, Mat() );

normalize(r\_hist, r\_hist, 0, histImage.rows, NORM\_MINMAX, -1, Mat() );

// Vẽ cho mỗi kênh, thuật toán này dùng trong biểu đồ Histogram 1-D

for( int i = 1; i < histSize; i++ )

{

line( histImage, Point( bin\_w\*(i-1), hist\_h - cvRound(b\_hist.at<float>(i-1)) ) ,

Point( bin\_w\*(i), hist\_h - cvRound(b\_hist.at<float>(i)) ),

Scalar( 255, 0, 0), 2, 8, 0 );

line( histImage, Point( bin\_w\*(i-1), hist\_h - cvRound(g\_hist.at<float>(i-1)) ) ,

Point( bin\_w\*(i), hist\_h - cvRound(g\_hist.at<float>(i)) ),

Scalar( 0, 255, 0), 2, 8, 0 );

line( histImage, Point( bin\_w\*(i-1), hist\_h - cvRound(r\_hist.at<float>(i-1)) ) ,

Point( bin\_w\*(i), hist\_h - cvRound(r\_hist.at<float>(i)) ),

Scalar( 0, 0, 255), 2, 8, 0 );

}

// Hiển thị

namedWindow("orignal", CV\_WINDOW\_AUTOSIZE);

imshow("orignal", src);

namedWindow("histogram", CV\_WINDOW\_AUTOSIZE );

imshow("histogram", histImage );

waitKey(0);

return 0;

}

**PYTHON**

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('lena.jpg',1)

cv2.imshow('image',img)

hist,bins = np.histogram(img.flatten(),256,[0,256]) # Hàm tính toán biểu đồ trong Numpy.Numpy cũng cung cấp cho bạn một hàm, np.histogram (). Vì vậy, thay vì hàm calcHist (), bạn có thể thử dòng dưới đây:

     Ví dụ : 1 hist, bins = np.histogram (img.ravel (), 256, [0,256])

hist giống như chúng ta đã tính toán trước đó. Nhưng các thùng sẽ có 257 phần tử, bởi vì Numpy tính các thùng là 0-0,99, 1-1,99, 2-2,99, v.v. Vì vậy, phạm vi cuối cùng sẽ là 255-255,99. Để thể hiện điều đó, họ cũng thêm 256 vào cuối thùng. Nhưng chúng tôi không cần 256. Tối đa 255 là đủ.

cdf = hist.cumsum() # Hàm cdf giúp định dạng giao diện pythonic cho phép thao tác các tệp theo cách quen thuộc và thoải mái hơn cho lập trình viên python cũng là bộ API

cdf\_normalized = cdf \* hist.max()/ cdf.max()

plt.plot(cdf\_normalized, color = 'b')

plt.hist(img.flatten(),256,[0,256], color = 'r')

plt.xlim([0,256])

plt.legend(('cdf','histogram'), loc = 'upper left')

plt.show()

## 3.9: Dò biên ảnh bằng Sobel

**C++**

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include "iostream"

using namespace cv;

using namespace std;

int main()

{

Mat src1;

src1 = imread("lena.jpg", CV\_LOAD\_IMAGE\_COLOR);

namedWindow("Original", CV\_WINDOW\_AUTOSIZE);

imshow("Original", src1);

Mat grey;

cvtColor(src1, grey, CV\_BGR2GRAY); // Lấy ảnh xám từ ảnh gốc

Mat sobelx;

Sobel(grey, sobelx, CV\_32F, 1, 0); // Hàm tìm biên theo x, chỉ dành cho opencv 3

double minVal, maxVal;

minMaxLoc(sobelx, &minVal, &maxVal); // Tìm cường độ tối thiểu và tối đa

cout << "minVal : " << minVal << endl << "maxVal : " << maxVal << endl;

Mat draw;

sobelx.convertTo(draw, CV\_8U, 255.0 / (maxVal - minVal), -minVal \* 255.0 / (maxVal - minVal));

namedWindow("sobel", CV\_WINDOW\_AUTOSIZE);

imshow("sobel", draw);

waitKey(0);

return 0;

}

**PYTHON**

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('lena.jpg',0)

# Mã dưới đây cho thấy tất cả các toán tử trong một sơ đồ duy nhất. Tất cả các hạt nhân có kích thước 5x5. Độ sâu của hình ảnh đầu ra được truyền -1 để có kết quả ở loại np.uint8. Với 2 hàm tìm biên như sau:

sobelx = cv2.Sobel(img,cv2.CV\_64F,1,0,ksize=5)

sobely = cv2.Sobel(img,cv2.CV\_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')

plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,2),plt.imshow(sobelx,cmap = 'gray')

plt.title('Sobel X'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,3),plt.imshow(sobely,cmap = 'gray')

plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()

## 3.10: Phân vùng ảnh

**C++**

#include "opencv2/imgproc.hpp"

#include "opencv2/imgcodecs.hpp"

#include "opencv2/highgui.hpp"

#include <iostream>

using namespace cv;

using std::cout;

int threshold\_value = 0;

int threshold\_type = 3;

int const max\_value = 255;

int const max\_type = 4;

int const max\_binary\_value = 255;

Mat src, src\_gray, dst; //src - hình ảnh đầu vào: không dấu 8 bit, không dấu 16 bit (CV\_16UC ...) hoặc dấu phẩy động chính xác đơn.

dst - hình ảnh đầu ra có cùng kích thước và độ sâu như src.

const char\* window\_name = "Threshold Demo";

const char\* trackbar\_type = "Type: \n 0: Binary \n 1: Binary Inverted \n 2: Truncate \n 3: To Zero \n 4: To Zero Inverted";

const char\* trackbar\_value = "Value";

static void Threshold\_Demo(int, void\*)

{

threshold(src\_gray, dst, threshold\_value, max\_binary\_value, threshold\_type);

// Hàm cv.threshold (src, dst, thresh, maxval, type)

Parameters

-src input array.

-dst output array of the same size and type and the same number of channels as src.

-thresh threshold value.

-maxval maximum value to use with the cv.THRESH\_BINARY and cv.THRESH\_BINARY\_INV thresholding types.

-type thresholding type(see cv.ThresholdTypes).

imshow(window\_name, dst);

}

int main(int argc, char\*\* argv)

{

String imageName("../data/stuff.jpg");

src = imread("lena.jpg", IMREAD\_COLOR);

if (src.empty())

{

cout << "Cannot read image: " << imageName << std::endl;

return -1;

}

cvtColor(src, src\_gray, COLOR\_BGR2GRAY); // Chuyển đổi hình ảnh sang màu xám

namedWindow(window\_name, WINDOW\_AUTOSIZE); // Tạo một cửa sổ để hiển thị kết quả

createTrackbar(trackbar\_type,

window\_name, &threshold\_type,

max\_type, Threshold\_Demo); // Tạo Trackbar để chọn loại Ngưỡng

createTrackbar(trackbar\_value,

window\_name, &threshold\_value,

max\_value, Threshold\_Demo); // Tạo Trackbar để chọn giá trị Ngưỡng

Threshold\_Demo(0, 0); // Gọi hàm để khởi tạo

waitKey();

return 0;

}

**PYTHON**

from \_\_future\_\_ import print\_function

import cv2 as cv

import argparse # Xem trang web này để có những thông tin giải thích cụ thể

[https://techblog.vn/pymotm-argparse, module argparse mình không hiểu kỹ mà chỉ tham khảo nên chưa có thể hiểu đúng được](https://techblog.vn/pymotm-argparse, module argparse mình không hiểu kỹ mà chỉ tham khảo nên chưa có thể hiểu đúng được )

max\_value = 255

max\_type = 4

max\_binary\_value = 255

trackbar\_type = 'Type: \n 0: Binary \n 1: Binary Inverted \n 2: Truncate \n 3: To Zero \n 4: To Zero Inverted'

trackbar\_value = 'Value'

window\_name = 'Threshold Demo'

def Threshold\_Demo(val):

threshold\_type = cv.getTrackbarPos(trackbar\_type, window\_name)

threshold\_value = cv.getTrackbarPos(trackbar\_value, window\_name)

\_, dst = cv.threshold(src\_gray, threshold\_value, max\_binary\_value, threshold\_type )

cv.imshow(window\_name, dst)

parser = argparse.ArgumentParser(description='Code for Basic Thresholding Operations tutorial.')

parser.add\_argument('--input', help='Path to input image.', default='stuff.jpg')

args = parser.parse\_args()

src = cv.imread('lena.jpg')

if src is None:

print('Could not open or find the image: ', args.input)

exit(0)

src\_gray = cv.cvtColor(src, cv.COLOR\_BGR2GRAY)

cv.namedWindow(window\_name)

cv.createTrackbar(trackbar\_type, window\_name , 3, max\_type, Threshold\_Demo)

cv.createTrackbar(trackbar\_value, window\_name , 0, max\_value, Threshold\_Demo)

Threshold\_Demo(0)

cv.waitKey()

## 3.11: Nhị phân ảnh

**C++**

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include "iostream"

using namespace cv;

using namespace std;

int main()

{

Mat image;

image = imread("lena.jpg", CV\_LOAD\_IMAGE\_COLOR);

if (!image.data)

{

cout << "Could not open or find the image" << std::endl;

return -1;

}

// Tạo một ma trận mới để giữ hình ảnh màu xám

Mat gray;

cvtColor(image, gray, CV\_BGR2GRAY); // chuyển đổi hình ảnh RGB sang màu xám

namedWindow("original image", CV\_WINDOW\_AUTOSIZE);

imshow("original image", image);

namedWindow("binary image", CV\_WINDOW\_AUTOSIZE);

imshow("binary image", gray);

waitKey(0);

return 0;

}

**PYTHON**

import cv2

import numpy

img = cv2.imread('download.jpg',0)

ret,gray=cv2.threshold(img,127,256,cv2.THRESH\_BINARY) # Nếu giá trị pixel lớn hơn một giá trị ngưỡng, nó được gán một giá trị (có thể là màu trắng), nếu không, nó được gán một giá trị khác (có thể là màu đen). Đây là hàm lấy ngưỡng xám với cv2.THRESH\_BINARY là giá trị mình chỉ lấy 2 màu đen và trắng trong hình ảnh mà không lấy màu xám hay lai màu khác. Ngoài ra còn có các giá trị khác cho bạn lựa chọn (cv2.THRESH\_BINARY\_INV,cv2.THRESH\_TRUNC, cv2.THRESH\_TOZERO, cv2.THRESH\_TOZERO\_INV)

cv2.imshow('orginal image',img)

cv2.imshow('binary image',gray)

cv2.waitKey(0)

cv2.DestroyAllWindows()

## 3.12: Biến đổi Hough

**C++**

Hough\_line(image) -biến đổi hough bằng đường thẳng trong ảnh

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/opencv.hpp>

#include <stdio.h>

#include <iostream>

using namespace cv;

using namespace std;

// biến để lưu trữ hình ảnh

Mat dst, cimg, gray, img, edges;

int initThresh;

const int maxThresh = 1000;

double th1, th2;

// tạo một vectơ để lưu trữ các điểm của đường

vector<Vec4i> lines;

void onTrackbarChange(int, void\*)

{

cimg = img.clone();

dst = img.clone();

th1 = initThresh;

th2 = th1 \* 0.4;

Canny(img, edges, th1, th2);

HoughLinesP(edges, lines, 2, CV\_PI / 180, 50, 10, 100); // áp dụng biến đổi dòng hough

// vẽ các đường trên các điểm được phát hiện

for (size\_t i = 0; i < lines.size(); i++)

{

Vec4i l = lines[i];

line(dst, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 0, 255), 1, LINE\_AA);

}

imshow("Result Image", dst); // hiển thị hình ảnh kết quả

imshow("Edges", edges);

}

int main(int argc, char\*\* argv) {

const char\* file = argv[1];

img = imread("lanes.jpg", 1); // Đọc hình ảnh (chế độ màu)

dst = img.clone();

if (img.empty())

{

cout << "Error in reading image" << file << endl;

return -1;

}

cvtColor(img, gray, COLOR\_BGR2GRAY); // Chuyển sang thang màu xám

// Phát hiện các cạnh bằng dò Canny Edge

// Canny (màu xám, dst, 50, 200, 3);

// Tạo một bản sao của hình ảnh gốc

// Sẽ giữ kết quả phát hiện

namedWindow("Edges", 1);

namedWindow("Result Image", 1);

// Khai báo thresh để thay đổi max\_radius của các vòng tròn được phát hiện trong biến đổi hough

initThresh = 500;

createTrackbar("threshold", "Result Image", &initThresh, maxThresh, onTrackbarChange); // Tạo trackbar để thay đổi giá trị ngưỡng

onTrackbarChange(initThresh, 0);

while (true)

{

int key;

key = waitKey(1);

if ((char)key == 27)

{

break;

}

}

destroyAllWindows();

}

Hough\_circle(image) -biến đổi hough bằng hình tròn trong ảnh

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/video/background\_segm.hpp>

#include <opencv2/opencv.hpp>

#include <stdio.h>

#include <iostream>

#include <string>

using namespace cv;

using namespace std;

// Khai báo các biến để lưu trữ hình ảnh

Mat gray, cimg, img, edges;

int initThresh;

const int maxThresh = 200;

double p1, p2;

// Vector để lưu điểm vòng tròn

vector<Vec3f> circles;

void onTrackbarChange(int, void\*) {

cimg = img.clone();

p1 = initThresh;

p2 = initThresh \* 0.4;

HoughCircles(gray, circles, HOUGH\_GRADIENT, 1, cimg.rows / 64, p1, p2, 25, 50); // Phát hiện các vòng tròn bằng cách sử dụng biến đổi HoughCircles

for (size\_t i = 0; i < circles.size(); i++)

{

Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));

int radius = cvRound(circles[i][2]);

circle(cimg, center, radius, Scalar(0, 255, 0), 2); // Vẽ vòng tròn bên ngoài

circle(cimg, center, 2, Scalar(0, 0, 255), 3); // Vẽ tâm đường tròn

}

imshow("Image", cimg); // Hiển thị hình ảnh đầu ra

Canny(gray, edges, p1, p2); // Hình ảnh cạnh để gỡ lỗi

imshow("Edges", edges);

}

int main(int argc, char\*\* argv) {

const char\* file = argv[1];

img = imread("brown-eyes.jpg", IMREAD\_COLOR);

if (img.empty())

{

cout << "Error reading image" << file << endl;

return -1;

}

cvtColor(img, gray, COLOR\_BGR2GRAY); // Chuyển sang thang màu xám

namedWindow("Edges", 1); // Sẽ giữ kết quả phát hiện

namedWindow("Image", 1);

initThresh = 105;

createTrackbar("Threshold", "Image", &initThresh, maxThresh, onTrackbarChange);

onTrackbarChange(initThresh, 0);

imshow("Image", img);

while (true)

{

int key;

key = waitKey(0);

if ((char)key == 27)

{

break;

}

}

destroyAllWindows();

}

**PYTHON**

hough\_lines(image) – biến đổi hough bằng đường thẳng trong ảnh

import cv2

import numpy as np

import sys

def onTrackbarChange(max\_slider):

global img

global dst

global gray

dst = np.copy(img)

th1 = max\_slider

th2 = th1 \* 0.4

edges = cv2.Canny(img, th1, th2)

# Áp dụng biến đổi dòng hough xác suất

lines = cv2.HoughLinesP(edges, 2, np.pi/180.0, 50, minLineLength=10, maxLineGap=100)

# Vẽ các đường trên các điểm được phát hiện

for line in lines:

x1, y1, x2, y2 = line[0]

cv2.line(dst, (x1, y1), (x2, y2), (0,0,255), 1)

cv2.imshow("Result Image", dst)

cv2.imshow("Edges",edges)

if \_\_name\_\_ == "\_\_main\_\_":

# Đọc hình ảnh

img = cv2.imread('lanes.jpg')

# Tạo một bản sao để sử dụng sau

dst = np.copy(img)

# Chuyển đổi hình ảnh sang màu xám

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Tạo cửa sổ hiển thị

cv2.namedWindow("Edges")

cv2.namedWindow("Result Image")

# Khởi tạo giá trị ngưỡng

initThresh = 500

# Giá trị ngưỡng tối đa

maxThresh = 1000

cv2.createTrackbar("threshold", "Result Image", initThresh, maxThresh, onTrackbarChange)

onTrackbarChange(initThresh)

while True:

key = cv2.waitKey(1)

if key == 27:

break

cv2.destroyAllWindows()

hough\_lines(video) – biến đổi hough bằng đường thẳng trong video

import cv2

import numpy as np

video = cv2.VideoCapture("road\_car\_view.mp4") # Khởi tạo việc tải video

while True:

ret, orig\_frame = video.read()

if not ret:

video = cv2.VideoCapture("road\_car\_view.mp4")

continue

frame = cv2.GaussianBlur(orig\_frame, (5, 5), 0)

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV) #chuyển ảnh từ BGR sang HSV. Phần cốt lõi của phát hiện là trích xuất chính xác các đường trên đường từ tất cả các hình ảnh còn lại. Chúng ta có thể làm điều này bằng cách áp dụng phát hiện màu hsv. Bằng cách này, chúng ta có thể phát hiện đối tượng bằng màu sắc của chúng, vì các đường của một con đường có thể chỉ có màu vàng hoặc trắng, chúng ta trích xuất một phần hình ảnh chỉ chứa hai màu này.

low\_yellow = np.array([18, 94, 140])

up\_yellow = np.array([48, 255, 255])

mask = cv2.inRange(hsv, low\_yellow, up\_yellow)

edges = cv2.Canny(mask, 75, 150) # Phần màu trắng tương ứng với các màu vàng trong khung ban đầu, vì vậy chúng ta có thể thấy các đường và một số phần của các tòa nhà màu vàng. Khi chúng ta có mặt nạ tìm thấy các cạnh, chúng ta sử dụng phương thức biến đổi hough và việc phát hiện được thực hiện

lines = cv2.HoughLinesP(edges, 1, np.pi/180, 50, maxLineGap=50)

if lines is not None:

for line in lines:

x1, y1, x2, y2 = line[0]

cv2.line(frame, (x1, y1), (x2, y2), (0, 255, 0), 5)

# Cuối cùng là hiển thị mọi thứ trên màn hình

cv2.imshow("frame", frame)

cv2.imshow("edges", edges)

key = cv2.waitKey(25)

if key == 27:

break

video.release()

cv2.destroyAllWindows()

## 3.13: Nhận diện mắt người

**C++**

#include <opencv2/opencv.hpp>

using namespace std;

using namespace cv;

void fillHoles(Mat &mask)

{

// Thuật toán này được giải mã cụ thể trong web này:

https://www.learnopencv.com/filling-holes-in-an-image-using-opencv-python-c/ Mat maskFloodfill = mask.clone();

floodFill(maskFloodfill, cv::Point(0, 0), Scalar(255));

Mat mask2;

bitwise\_not(maskFloodfill, mask2);

mask = (mask2 | mask);

}

int main(int argc, char\*\* argv)

{

Mat img = imread("red\_eyes2.jpg", CV\_LOAD\_IMAGE\_COLOR); // Đọc ảnh

Mat imgOut = img.clone(); // Xuất ảnh

CascadeClassifier eyesCascade("haarcascade\_eye.xml"); // Tải file HAAR cascade

std::vector<Rect> eyes; // Nhận dạng mắt

eyesCascade.detectMultiScale(img, eyes, 1.3, 4, 0 | CASCADE\_SCALE\_IMAGE, Size(100, 100));

// Với mọi mắt được phát hiện

for (size\_t i = 0; i < eyes.size(); i++)

{

Mat eye = img(eyes[i]); // Trích xuất mắt từ hình ảnh

vector<Mat>bgr(3); // Chia hình ảnh mắt thành 3 kênh

split(eye, bgr); // Máy dò mắt đỏ đơn giản

Mat mask = (bgr[2] > 150) & (bgr[2] > (bgr[1] + bgr[0]));

// Làm sạch mặt nạ - 1) Lỗ tệp 2) Mặt nạ giãn nở (mở rộng)

fillHoles(mask);

dilate(mask, mask, Mat(), Point(-1, -1), 3, 1, 1);

// Tính kênh trung bình bằng cách tính trung bình

// các kênh màu xanh lá cây và màu xanh

Mat mean = (bgr[0] + bgr[1]) / 2;

mean.copyTo(bgr[2], mask);

mean.copyTo(bgr[0], mask);

mean.copyTo(bgr[1], mask);

// Hợp nhất các kênh

Mat eyeOut;

cv::merge(bgr, eyeOut);

eyeOut.copyTo(imgOut(eyes[i])); // Sao chép mắt cố định vào hình ảnh đầu ra.

}

// Kết quả hiển thị

imshow("Red Eyes", img);

imshow("original ", imgOut);

waitKey(0);

}

**PYTHON**

import cv2

eye\_cascade = cv2.CascadeClassifier("eye.xml") # Nhập file eye.xml nhân dạng mắt có sẵn của opencv

video\_capture = cv2.VideoCapture(0) # Sử dụng camera

while True:

\_,frame = video\_capture.read()

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) # Lọc sang ảnh xám

eye\_detect = eye\_cascade.detectMultiScale(

gray,scaleFactor=1.3,minNeighbors=20,flags =cv2.CASCADE\_SCALE\_IMAGE,

minSize=(30,30) ) # Vì hầu hết các phát hiện mắt sử dụng vùng da quanh mí mắt, long mi và long mày để phát hiện và đó từ chức năng dò tìm MultiScale và đây là hàm sử dụng

for (x,y,w,h) in eye\_detect:

x\_center = int((x+(x+w))/2)

y\_center = int((y+(y+h))/2)

cv2.circle(frame,(x\_center,y\_center),20,(0,0,255),5) # Hàm nhận dạng mắt với hình tròn màu đỏ

cv2.imshow('Webcam',frame)

if cv2.waitKey(1) & 0xFF == ord('q'): # Nhấn phím ‘q’ trên bàn phím để thoát

break

video\_capture.release()

cv2.destroyAllWindows()

## 3.14: Nhận diện khuôn mặt người

**C++**

#include <iostream>

#include <opencv2\core\core.hpp>

#include <opencv2\highgui\highgui.hpp>

#include <opencv2\imgproc\imgproc.hpp>

#include <opencv2\objdetect\objdetect.hpp>

using namespace cv;

using namespace std;

int main() {

Mat image;

vector<Rect> faces;

CascadeClassifier nhandien;

VideoCapture camera;

camera.open(0);

nhandien.load("haarcascade\_frontalface\_alt.xml"); // Hàm khai báo file xml cua nhận dạng mặt của opencv

while (1) { // Hàm dành riêng cho file xml

camera.read(image); nhandien.detectMultiScale(image, faces, 1.1, 2, CV\_HAAR\_SCALE\_IMAGE, Size(30, 30));

if (faces.empty()) {

cout << "Khong phat hien mat nguoi!" << endl;

}

for (int i = 0; i < faces.size(); ++i)

rectangle(image, faces.at(i), CV\_RGB(200, 0, 0), 2); // Nhận diện với ảnh RGB

cout << "So mat phat hien duoc: " << faces.size() << endl;

imshow("PHAT HIEN GUONG MAT", image);

cvWaitKey(10);

}

waitKey(0);

}

**PYTHON**

import cv2

face\_cascade = cv2.CascadeClassifier("face.xml") # Nhập file xml nhận dạng mặt của opencv

video\_capture = cv2.VideoCapture(0) # Kích hoạt camera

while True:

\_,frame = video\_capture.read()

face\_detect = face\_cascade.detectMultiScale(

frame,scaleFactor=1.1,minNeighbors=5,flags =cv2.CASCADE\_SCALE\_IMAGE,

minSize=(30,30)

) # Hàm nhận dạng mặt người. Ở đây chúng tôi đọc hình ảnh và chuyển đổi nó thành thang độ xám. Hàm này phát hiện khuôn mặt thực tế và là phần quan trọng trong mã của chúng tôi, vì vậy hãy để Lướt qua các tùy chọn:

Hàm DetMultiScale là một hàm chung phát hiện các đối tượng. Vì chúng tôi đang gọi nó trên thác mặt, đó là những gì nó phát hiện ra.

Tùy chọn đầu tiên là hình ảnh thang độ xám.

Thứ hai là scaleFactor. Vì một số khuôn mặt có thể gần camera hơn, chúng sẽ xuất hiện lớn hơn khuôn mặt ở phía sau. Các yếu tố quy mô bù đắp cho điều này.

Thuật toán phát hiện sử dụng một cửa sổ di chuyển để phát hiện các đối tượng. minNeighbor xác định có bao nhiêu đối tượng được phát hiện gần đối tượng hiện tại trước khi nó tuyên bố khuôn mặt được tìm thấy. Trong khi đó, minSize cho kích thước của mỗi cửa sổ.

for (x,y,w,h) in face\_detect:

cv2.rectangle(frame,(x,y),(x+w,y+h),(0,0,255),5) # Hàm trả về một danh sách các hình chữ nhật trong đó nó tin rằng nó tìm thấy một khuôn mặt. Tiếp theo, chúng tôi sẽ lặp đi lặp lại nơi mà nó nghĩ rằng nó đã tìm thấy thứ gì đó.

cv2.imshow('Webcam',frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

video\_capture.release()

cv2.destroyAllWindows()

## 3.15: Nhận diện bàn tay người

**C++** (Bạn thấy đấy, hãy để Giả sử bạn ở phía trước một nền với màu sắc khác với da của bạn. Ngoài ra, hãy để Lôi giả định rằng bàn tay của bạn là phần lớn nhất trên cơ thể bạn xuất hiện. Trong trường hợp này, nhận dạng bàn tay sẽ dễ dàng, bằng cách chỉ cần chọn các pixel thuộc về màu da của bạn và sau đó lấy ra vùng lớn nhất. Đó chính xác là những gì mình sẽ làm! xem trang web này để hiểu cụ thể “<https://picoledelimao.github.io/blog/2015/11/15/fingertip-detection-on-opencv/>” )

#include <iostream>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

int minH = 0, maxH = 20, minS = 30, maxS = 150, minV = 60, maxV = 255;

cv::Mat frame;

int count = 0;

float innerAngle(float px1, float py1, float px2, float py2, float cx1, float cy1)

{

float dist1 = std::sqrt((px1 - cx1)\*(px1 - cx1) + (py1 - cy1)\*(py1 - cy1));

float dist2 = std::sqrt((px2 - cx1)\*(px2 - cx1) + (py2 - cy1)\*(py2 - cy1));

float Ax, Ay;

float Bx, By;

float Cx, Cy;

// Tìm điểm gần nhất với C

Cx = cx1;

Cy = cy1;

if (dist1 < dist2)

{

Bx = px1;

By = py1;

Ax = px2;

Ay = py2;

}

else {

Bx = px2;

By = py2;

Ax = px1;

Ay = py1;

}

float Q1 = Cx - Ax;

float Q2 = Cy - Ay;

float P1 = Bx - Ax;

float P2 = By - Ay;

float A = std::acos((P1\*Q1 + P2 \* Q2) / (std::sqrt(P1\*P1 + P2 \* P2) \* std::sqrt(Q1\*Q1 + Q2 \* Q2)));

A = A \* 180 / CV\_PI;

return A;

}

void CallbackFunc(int event, int x, int y, int flags, void\* userdata)

{

cv::Mat RGB = frame(cv::Rect(x, y, 1, 1));

cv::Mat HSV;

cv::cvtColor(RGB, HSV, CV\_BGR2HSV);

cv::Vec3b pixel = HSV.at<cv::Vec3b>(0, 0);

if (event == cv::EVENT\_LBUTTONDBLCLK) // Khi nhấp đúp chuột trái

{

std::cout << "Click" << std::endl;

int h = pixel.val[0];

int s = pixel.val[1];

int v = pixel.val[2];

if (count == 0)

{

minH = h;

maxH = h;

minS = s;

maxS = s;

minV = v;

maxV = v;

}

else

{

if (h < minH)

{

minH = h;

}

else if (h > maxH)

{

maxH = h;

}

if (s < minS)

{

minS = s;

}

else if (s > maxS)

{

maxS = s;

}

if (v < minV)

{

minV = v;

}

else if (v > maxV)

{

maxV = v;

}

}

count++;

}

std::cout << pixel << std::endl;

}

int main()

{

cv::VideoCapture cap(0);

const char\* windowName = "Fingertip detection";

cv::namedWindow(windowName);

cv::setMouseCallback(windowName, CallbackFunc, NULL);

int inAngleMin = 200, inAngleMax = 300, angleMin = 180, angleMax = 359, lengthMin = 10, lengthMax = 80;

cv::createTrackbar("Inner angle min", windowName, &inAngleMin, 360);

cv::createTrackbar("Inner angle max", windowName, &inAngleMax, 360);

cv::createTrackbar("Angle min", windowName, &angleMin, 360);

cv::createTrackbar("Angle max", windowName, &angleMax, 360);

cv::createTrackbar("Length min", windowName, &lengthMin, 100);

cv::createTrackbar("Length max", windowName, &lengthMax, 100);

while (1)

{

cap >> frame;

cv::Mat hsv;

cv::cvtColor(frame, hsv, CV\_BGR2HSV);

cv::inRange(hsv, cv::Scalar(minH, minS, minV), cv::Scalar(maxH, maxS, maxV), hsv);

int blurSize = 5;

int elementSize = 5;

cv::medianBlur(hsv, hsv, blurSize);

cv::Mat element = cv::getStructuringElement(cv::MORPH\_ELLIPSE, cv::Size(2 \* elementSize + 1, 2 \* elementSize + 1), cv::Point(elementSize, elementSize));

cv::dilate(hsv, hsv, element);

// Phát hiện đường viền

std::vector<std::vector<cv::Point> > contours;

std::vector<cv::Vec4i> hierarchy;

cv::findContours(hsv, contours, hierarchy, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_SIMPLE, cv::Point(0, 0));

size\_t largestContour = 0;

for (size\_t i = 1; i < contours.size(); i++)

{

if (cv::contourArea(contours[i]) > cv::contourArea(contours[largestContour]))

largestContour = i;

}

cv::drawContours(frame, contours, largestContour, cv::Scalar(0, 0, 255), 1);

// Vỏ lồi

if (!contours.empty())

{

std::vector<std::vector<cv::Point> > hull(1);

cv::convexHull(cv::Mat(contours[largestContour]), hull[0], false);

cv::drawContours(frame, hull, 0, cv::Scalar(0, 255, 0), 3);

if (hull[0].size() > 2)

{

std::vector<int> hullIndexes;

cv::convexHull(cv::Mat(contours[largestContour]), hullIndexes, true);

std::vector<cv::Vec4i> convexityDefects;

cv::convexityDefects(cv::Mat(contours[largestContour]), hullIndexes, convexityDefects);

cv::Rect boundingBox = cv::boundingRect(hull[0]);

cv::rectangle(frame, boundingBox, cv::Scalar(255, 0, 0));

cv::Point center = cv::Point(boundingBox.x + boundingBox.width / 2, boundingBox.y + boundingBox.height / 2);

std::vector<cv::Point> validPoints;

for (size\_t i = 0; i < convexityDefects.size(); i++)

{

cv::Point p1 = contours[largestContour][convexityDefects[i][0]];

cv::Point p2 = contours[largestContour][convexityDefects[i][1]];

cv::Point p3 = contours[largestContour][convexityDefects[i][2]];

double angle = std::atan2(center.y - p1.y, center.x - p1.x) \* 180 / CV\_PI;

double inAngle = innerAngle(p1.x, p1.y, p2.x, p2.y, p3.x, p3.y);

double length = std::sqrt(std::pow(p1.x - p3.x, 2) + std::pow(p1.y - p3.y, 2));

if (angle > angleMin - 180 && angle < angleMax - 180 && inAngle > inAngleMin - 180 && inAngle < inAngleMax - 180 && length > lengthMin / 100.0 \* boundingBox.height && length < lengthMax / 100.0 \* boundingBox.height)

{

validPoints.push\_back(p1);

}

}

for (size\_t i = 0; i < validPoints.size(); i++)

{

cv::circle(frame, validPoints[i], 9, cv::Scalar(0, 255, 0), 2);

}

}

}

cv::imshow(windowName, frame);

if (cv::waitKey(30) >= 0) break;

}

return 0;

}

**PYTHON**

hand\_detection(cơ bản)

import cv2

import numpy as np

cap=cv2.VideoCapture(0) # Khởi động camera

hand\_cascade=cv2.CascadeClassifier('hand.xml') # Nhập file xml nhận dạng bàn tay từ opencv

while(True):

ret,frame=cap.read()

gray=cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY) # Nhập hình ảnh sang màu xám

hands=hand\_cascade.detectMultiScale(gray,1.1,5)

for(x,y,w,h) in hands: # x,y,w,h là 4 trục của khung hình nhận dạng chữ nhật

cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)

cv2.imshow('frame',frame)

if cv2.waitKey(1) &0xFF==ord('q'): # Nhấn phím’q’ để thoát

break

cap.release()

cv2.destroyAllWindows()

hand\_detection(phức tạp)

import cv2

import numpy as np

import imutils

import math

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

# Nhập file xml nhận dạng của opencv

#YCrCb = Y: thành phần độ chói, Cr: màu đỏ chrominat, Cb: màu xanh lam với việc lọc với kích thước kernel (phần tham khảo)

min\_YCrCb = np.array([0, 131, 100],np.uint8)

max\_YCrCb = np.array([255, 185, 135],np.uint8)

kernelopen= np.ones((1,1))

kernelclose= np.ones((1,1))

kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(3,3))

fgbg = cv2.createBackgroundSubtractorMOG2(history = 20, varThreshold =25)

cap = cv2.VideoCapture(0)

while True:

ret, img = cap.read()

img = cv2.resize(img,(340,220))

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

gray = cv2.GaussianBlur(gray, (5, 5), 0)

faces = face\_cascade.detectMultiScale(gray, 1.3, 5)

for(x,y,w,h) in faces :

cv2.rectangle(img, (x,y-20), (x+w,y+h+50), (0,0,0), -1)

roi\_gray = gray[y:y+h, x:x+w]

roi\_color = img[y:y+h, x:x+w]

imageYCrCb = cv2.cvtColor(img,cv2.COLOR\_BGR2YCR\_CB)

skinRegion = cv2.inRange(imageYCrCb,min\_YCrCb,max\_YCrCb)

maskopen = cv2.morphologyEx(skinRegion,cv2.MORPH\_OPEN, kernelopen)

maskclose = cv2.morphologyEx(maskopen,cv2.MORPH\_OPEN, kernelclose)

contours, hierarchy = cv2.findContours(maskclose.copy(), cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

try:

cnt = max(contours, key=lambda x:cv2.contourArea(x))

x,y,w,h = cv2.boundingRect(cnt) # Tạo trục cho khung hình nhận dạng

cv2.rectangle(img,(x,y), (x+w, y+h), (0,0,255), 0)

hull = cv2.convexHull(cnt)

drawing = np.zeros(img.shape, np.uint8)

cv2.drawContours(drawing, [cnt], -1, (0,255,0), 0)

cv2.drawContours(drawing, [hull], -1, (0,0,255), 0)

hull = cv2.convexHull(cnt, returnPoints=False)

defects = cv2.convexityDefects(cnt,hull)

d=0

for i in range(defects.shape[0]):

s,e,f,d = defects[i,0]

start = tuple(cnt[s][0])

end = tuple(cnt[e][0])

far = tuple(cnt[f][0])

a=math.sqrt((end[0] - start[0]) \*\*2 + (end[1] - start[1])\*\*2)

b=math.sqrt((far[0] - start[0]) \*\*2 + (far[1] - start[1])\*\*2)

c=math.sqrt((end[0] - far[0]) \*\*2 + (end[1] - far[1])\*\*2)

s = (a+b+c)/2

ar = math.sqrt(s\*(s-a)\*(s-b)\*(s-c))

d = (2\*ar)/a

angle = math.acos((b\*\*2 + c\*\*2 - a\*\*2)/(2\*b\*c))\*57

#if angle <= 90 : (it gives all the defects far,end and start)

if angle <= 90 and d>30:

d+=1

cv2.circle(img, far, 3, [0,0,255],-1)

cv2.line(img, start, end, [0,255,0], 2)

FONT = cv2.FONT\_HERSHEY\_SIMPLEX

if d == 4:

cv2.putText(img, "5", (0, 50), FONT, 2,(0,0,255), 2)

else:

pass

except:

pass

cv2.imshow('img',img)

cv2.imshow("maskclose",maskclose)

all\_img = np.hstack((drawing, img))

cv2.imshow('last', all\_img)

key = cv2.waitKey(1)

if key == ord('q'):

break

cv2.destroyAllWindows()

cap.release()

## 3.16: Nhận diện con người

**C++**

Blob.h, Blob.cpp (Là các file có sẵn chỉ cần khai báo nên không được đề cập)

main.cpp

#include<opencv2/core/core.hpp>

#include<opencv2/highgui/highgui.hpp>

#include<opencv2/imgproc/imgproc.hpp>

#include<iostream>

#ifdef WINDOWS

#include<conio.h> // Có thể cần phải thay đổi hoặc xóa dòng này nếu không sử dụng Windows

#endif

#include "Blob.h"

// Biến toàn cục

const cv::Scalar SCALAR\_BLACK = cv::Scalar(0.0, 0.0, 0.0);

const cv::Scalar SCALAR\_WHITE = cv::Scalar(255.0, 255.0, 255.0);

const cv::Scalar SCALAR\_BLUE = cv::Scalar(255.0, 0.0, 0.0);

const cv::Scalar SCALAR\_GREEN = cv::Scalar(0.0, 200.0, 0.0);

const cv::Scalar SCALAR\_RED = cv::Scalar(0.0, 0.0, 255.0);

int main(void) {

cv::VideoCapture capVideo;

cv::Mat imgFrame1;

cv::Mat imgFrame2;

capVideo.open("768x576.avi"); // Mở video

if (!capVideo.isOpened()) { // Nếu không thể mở tệp video

std::cout << "\nerror reading video file" << std::endl << std::endl; // Hiển thị thông báo lỗi

#ifdef WINDOWS

\_getch(); // Có thể cần phải thay đổi hoặc xóa dòng này nếu không sử dụng Windows

#endif

return(0);

}

if (capVideo.get(CV\_CAP\_PROP\_FRAME\_COUNT) < 2) {

std::cout << "\nerror: video file must have at least two frames";

#ifdef WINDOWS

\_getch();

#endif

return(0);

}

capVideo.read(imgFrame1);

capVideo.read(imgFrame2);

char chCheckForEscKey = 0;

while (capVideo.isOpened() && chCheckForEscKey != 27) // Nếu video mở được thì sẽ khởi động các hàm sau

{

std::vector<Blob> blobs;

cv::Mat imgFrame1Copy = imgFrame1.clone();

cv::Mat imgFrame2Copy = imgFrame2.clone();

cv::Mat imgDifference;

cv::Mat imgThresh;

cv::cvtColor(imgFrame1Copy, imgFrame1Copy, CV\_BGR2GRAY);

cv::cvtColor(imgFrame2Copy, imgFrame2Copy, CV\_BGR2GRAY);

cv::GaussianBlur(imgFrame1Copy, imgFrame1Copy, cv::Size(5, 5), 0);

cv::GaussianBlur(imgFrame2Copy, imgFrame2Copy, cv::Size(5, 5), 0);

cv::absdiff(imgFrame1Copy, imgFrame2Copy, imgDifference);

cv::threshold(imgDifference, imgThresh, 30, 255.0, CV\_THRESH\_BINARY);

cv::imshow("imgThresh", imgThresh);

cv::Mat structuringElement3x3 = cv::getStructuringElement(cv::MORPH\_RECT, cv::Size(3, 3));

cv::Mat structuringElement5x5 = cv::getStructuringElement(cv::MORPH\_RECT, cv::Size(5, 5));

cv::Mat structuringElement7x7 = cv::getStructuringElement(cv::MORPH\_RECT, cv::Size(7, 7));

cv::Mat structuringElement9x9 = cv::getStructuringElement(cv::MORPH\_RECT, cv::Size(9, 9));

cv::dilate(imgThresh, imgThresh, structuringElement5x5);

cv::dilate(imgThresh, imgThresh, structuringElement5x5);

cv::erode(imgThresh, imgThresh, structuringElement5x5);

cv::Mat imgThreshCopy = imgThresh.clone();

std::vector<std::vector<cv::Point> > contours;

cv::findContours(imgThreshCopy, contours, cv::RETR\_EXTERNAL, cv::CHAIN\_APPROX\_SIMPLE);

cv::Mat imgContours(imgThresh.size(), CV\_8UC3, SCALAR\_BLACK);

cv::drawContours(imgContours, contours, -1, SCALAR\_WHITE, -1);

cv::imshow("imgContours", imgContours);

std::vector<std::vector<cv::Point> > convexHulls(contours.size());

for (unsigned int i = 0; i < contours.size(); i++) {

cv::convexHull(contours[i], convexHulls[i]);

}

for (auto &convexHull : convexHulls) {

Blob possibleBlob(convexHull);

if (possibleBlob.boundingRect.area() > 100 &&

possibleBlob.dblAspectRatio >= 0.2 &&

possibleBlob.dblAspectRatio <= 1.2 &&

possibleBlob.boundingRect.width > 15 &&

possibleBlob.boundingRect.height > 20 &&

possibleBlob.dblDiagonalSize > 30.0) {

blobs.push\_back(possibleBlob);

}

}

cv::Mat imgConvexHulls(imgThresh.size(), CV\_8UC3, SCALAR\_BLACK);

convexHulls.clear();

for (auto &blob : blobs) {

convexHulls.push\_back(blob.contour);

}

cv::drawContours(imgConvexHulls, convexHulls, -1, SCALAR\_WHITE, -1);

cv::imshow("imgConvexHulls", imgConvexHulls);

imgFrame2Copy = imgFrame2.clone(); // Lấy một bản sao khác của khung 2 vì chúng tôi đã thay đổi bản sao của khung 2 trước đó trong quá trình xử lý ở trên

for (auto &blob : blobs) { // Cho mỗi blob

cv::rectangle(imgFrame2Copy, blob.boundingRect, SCALAR\_RED, 2); // Vẽ một hộp màu đỏ xung quanh blob

cv::circle(imgFrame2Copy, blob.centerPosition, 3, SCALAR\_GREEN, -1); // Vẽ một vòng tròn màu xanh lá cây đầy ở trung tâm

}

cv::imshow("imgFrame2Copy", imgFrame2Copy);

// Bây giờ chúng ta chuẩn bị cho lần lặp tiếp theo

imgFrame1 = imgFrame2.clone(); // Di chuyển khung 1 lên đến vị trí của khung 2

if ((capVideo.get(CV\_CAP\_PROP\_POS\_FRAMES) + 1) < capVideo.get(CV\_CAP\_PROP\_FRAME\_COUNT)) { // Nếu có ít nhất một khung hình nữa

capVideo.read(imgFrame2); // read it

}

else {

std::cout << "end of video\n"; // Hiển thị kết thúc tin nhắn video

break; // Nhảy ra khỏi vòng lặp while

}

chCheckForEscKey = cv::waitKey(1); // Lấy phím bấm trong trường hợp người dùng nhấn esc }

if (chCheckForEscKey != 27) { // Nếu người dùng không nhấn esc (tức là chúng tôi đã đến cuối video)

cv::waitKey(0); // Giữ các cửa sổ mở để cho phép thông báo "kết thúc video" hiển thị }

// lưu ý rằng nếu người dùng đã nhấn esc, chúng ta không cần phải mở các cửa sổ, chúng ta chỉ cần để chương trình kết thúc sẽ đóng các cửa sổ

return(0);

}

**PYTHON**

Bằng hình ảnh

from \_\_future\_\_ import print\_function

import numpy as np

import cv2 as cv

# Phần này ta nên tìm hiểu phương pháp HOG (Histogram of Oriented Gradients) tham khảo tại “https://thedatafrog.com/human-detection-video/”

Ý tưởng cơ bản của phương pháp là như sau:

Hình ảnh được quét với một cửa sổ phát hiện có kích thước khác nhau.

Đối với mỗi vị trí và kích thước của cửa sổ phát hiện, cửa sổ được chia nhỏ trong các ô. Các tế bào trong thực tế tương đối nhỏ: chúng thường chỉ chứa một phần nhỏ của người được phát hiện, có thể là một bên cánh tay hoặc đỉnh đầu.

Trong mỗi ô, một độ dốc được tính cho từng pixel và độ dốc được sử dụng để điền vào biểu đồ: giá trị là góc của độ dốc và trọng số là độ lớn của độ dốc.

Biểu đồ của tất cả các ô được ghép lại và đưa vào bộ phân biệt đối xử học máy để quyết định xem các ô của cửa sổ phát hiện hiện tại có tương ứng với một người hay không.

def inside(r, q):

rx, ry, rw, rh = r

qx, qy, qw, qh = q

return rx > qx and ry > qy and rx + rw < qx + qw and ry + rh < qy + qh

def draw\_detections(img, rects, thickness = 1):

for x, y, w, h in rects:

# the HOG detector returns slightly larger rectangles than the real objects.

# so we slightly shrink the rectangles to get a nicer output.

pad\_w, pad\_h = int(0.15\*w), int(0.05\*h)

cv.rectangle(img, (x+pad\_w, y+pad\_h), (x+w-pad\_w, y+h-pad\_h), (0, 255, 0), thickness)

def main():

import sys

from glob import glob

import itertools as it

hog = cv.HOGDescriptor()

hog.setSVMDetector( cv.HOGDescriptor\_getDefaultPeopleDetector() )

default = [cv.samples.findFile('dance2.jpg')] if len(sys.argv[1:]) == 0 else []

for fn in it.chain(\*map(glob, default + sys.argv[1:])):

print(fn, ' - ',)

try:

img = cv.imread(fn)

if img is None:

print('Failed to load image file:', fn)

continue

except:

print('loading error')

continue

found, w = hog.detectMultiScale(img, winStride=(8,8), padding=(32,32), scale=1.05)

found\_filtered = []

for ri, r in enumerate(found):

for qi, q in enumerate(found):

if ri != qi and inside(r, q):

break

else:

found\_filtered.append(r)

draw\_detections(img, found)

draw\_detections(img, found\_filtered, 3)

print('%d (%d) found' % (len(found\_filtered), len(found)))

cv.imshow('img', img)

ch = cv.waitKey()

if ch == 27:

break

print('Done')

if \_\_name\_\_ == '\_\_main\_\_':

print(\_\_doc\_\_)

main()

cv.destroyAllWindows()

Bằng video

import cv2

import numpy as np

cap=cv2.VideoCapture('human.mp4') # Video ta muốn nhận dạng

human\_cascade=cv2.CascadeClassifier('haarcascade\_fullbody.xml') # Nhập file xml hỗ trợ nhận dạng con người của opencv

while True:

ret,frame=cap.read()

gray=cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY) # Chuyển sang màu xám

human=human\_cascade.detectMultiScale(gray, 1.1,4) # Hàm hổ trợ nhận dạng từ file xml

for(x,y,w,h) in human:

cv2.rectangle(frame,(x,y),(x+w,y+h),(0,0,220),3)

cv2.imshow('video',frame)

if cv2.waitKey(25)& 0xFF==ord('q'): # Nhấn phím’q’ trên bàn phím để thoát

break

cap.release()

cv2.destroyAllWindows()

## 3.17: Nhận dạng xe hơi

**C++**

Blob.h (sẽ không nói cụ thể vì nó là file có sẵn, chỉ cần khai báo- Blob(A Binary Large Object) là tập hợp dữ liệu nhị phân được lưu trữ dưới dạng một thực thể trong hệ thống quản lý cơ sở dữ liệu, tìm hiểu trong “https://en.wikipedia.org/wiki/Binary\_large\_object”)

Blob.cpp (cũng như file blob.h nên cũng không đề cập)

Main.cpp

#include<opencv2/core/core.hpp>

#include<opencv2/highgui/highgui.hpp>

#include<opencv2/imgproc/imgproc.hpp>

#include<iostream>

#include<conio.h> // Có thể cần phải thay đổi hoặc xóa dòng này nếu không sử dụng Windows

#include "Blob.h"

#define SHOW\_STEPS // Biến toàn cục

const cv::Scalar SCALAR\_BLACK = cv::Scalar(0.0, 0.0, 0.0);

const cv::Scalar SCALAR\_WHITE = cv::Scalar(255.0, 255.0, 255.0);

const cv::Scalar SCALAR\_YELLOW = cv::Scalar(0.0, 255.0, 255.0);

const cv::Scalar SCALAR\_GREEN = cv::Scalar(0.0, 200.0, 0.0);

const cv::Scalar SCALAR\_RED = cv::Scalar(0.0, 0.0, 255.0);

// Hàm nguyên mẫu , cv::Scalar là các hàm màu sắc cho khung hình nhận dạng

// Khai báo các hàm sẽ sử dụng

void matchCurrentFrameBlobsToExistingBlobs(std::vector<Blob> &existingBlobs, std::vector<Blob> &currentFrameBlobs);

void addBlobToExistingBlobs(Blob &currentFrameBlob, std::vector<Blob> &existingBlobs, int &intIndex);

void addNewBlob(Blob &currentFrameBlob, std::vector<Blob> &existingBlobs);

double distanceBetweenPoints(cv::Point point1, cv::Point point2);

void drawAndShowContours(cv::Size imageSize, std::vector<std::vector<cv::Point> > contours, std::string strImageName);

void drawAndShowContours(cv::Size imageSize, std::vector<Blob> blobs, std::string strImageName);

bool checkIfBlobsCrossedTheLine(std::vector<Blob> &blobs, int &intHorizontalLinePosition, int &carCount);

void drawBlobInfoOnImage(std::vector<Blob> &blobs, cv::Mat &imgFrame2Copy);

void drawCarCountOnImage(int &carCount, cv::Mat &imgFrame2Copy);

int main(void) {

cv::VideoCapture capVideo;

cv::Mat imgFrame1;

cv::Mat imgFrame2;

std::vector<Blob> blobs;

cv::Point crossingLine[2];

int carCount = 0;

capVideo.open("CarsDrivingUnderBridge.mp4");

if (!capVideo.isOpened()) { // Nếu không thể mở tệp video

std::cout << "error reading video file" << std::endl << std::endl; // Hiển thị thông báo lỗi

\_getch(); // Có thể cần phải thay đổi hoặc xóa dòng này nếu không sử dụng Windows

return(0);

}

if (capVideo.get(CV\_CAP\_PROP\_FRAME\_COUNT) < 2) {

std::cout << "error: video file must have at least two frames";

\_getch();

return(0);

}

capVideo.read(imgFrame1);

capVideo.read(imgFrame2);

int intHorizontalLinePosition = (int)std::round((double)imgFrame1.rows \* 0.35);

crossingLine[0].x = 0;

crossingLine[0].y = intHorizontalLinePosition;

crossingLine[1].x = imgFrame1.cols - 1;

crossingLine[1].y = intHorizontalLinePosition;

char chCheckForEscKey = 0;

bool blnFirstFrame = true;

int frameCount = 2;

while (capVideo.isOpened() && chCheckForEscKey != 27) {

std::vector<Blob> currentFrameBlobs;

cv::Mat imgFrame1Copy = imgFrame1.clone();

cv::Mat imgFrame2Copy = imgFrame2.clone();

cv::Mat imgDifference;

cv::Mat imgThresh;

cv::cvtColor(imgFrame1Copy, imgFrame1Copy, CV\_BGR2GRAY);

cv::cvtColor(imgFrame2Copy, imgFrame2Copy, CV\_BGR2GRAY);

cv::GaussianBlur(imgFrame1Copy, imgFrame1Copy, cv::Size(5, 5), 0);

cv::GaussianBlur(imgFrame2Copy, imgFrame2Copy, cv::Size(5, 5), 0);

cv::absdiff(imgFrame1Copy, imgFrame2Copy, imgDifference);

cv::threshold(imgDifference, imgThresh, 30, 255.0, CV\_THRESH\_BINARY);

cv::imshow("imgThresh", imgThresh);

cv::Mat structuringElement3x3 = cv::getStructuringElement(cv::MORPH\_RECT, cv::Size(3, 3));

cv::Mat structuringElement5x5 = cv::getStructuringElement(cv::MORPH\_RECT, cv::Size(5, 5));

cv::Mat structuringElement7x7 = cv::getStructuringElement(cv::MORPH\_RECT, cv::Size(7, 7));

cv::Mat structuringElement15x15 = cv::getStructuringElement(cv::MORPH\_RECT, cv::Size(15, 15));

for (unsigned int i = 0; i < 2; i++) {

cv::dilate(imgThresh, imgThresh, structuringElement5x5);

cv::dilate(imgThresh, imgThresh, structuringElement5x5);

cv::erode(imgThresh, imgThresh, structuringElement5x5);

}

cv::Mat imgThreshCopy = imgThresh.clone();

std::vector<std::vector<cv::Point> > contours;

cv::findContours(imgThreshCopy, contours, cv::RETR\_EXTERNAL, cv::CHAIN\_APPROX\_SIMPLE);

drawAndShowContours(imgThresh.size(), contours, "imgContours");

std::vector<std::vector<cv::Point> > convexHulls(contours.size());

for (unsigned int i = 0; i < contours.size(); i++) {

cv::convexHull(contours[i], convexHulls[i]);

}

drawAndShowContours(imgThresh.size(), convexHulls, "imgConvexHulls");

for (auto &convexHull : convexHulls) {

Blob possibleBlob(convexHull);

if (possibleBlob.currentBoundingRect.area() > 400 &&

possibleBlob.dblCurrentAspectRatio > 0.2 &&

possibleBlob.dblCurrentAspectRatio < 4.0 &&

possibleBlob.currentBoundingRect.width > 30 &&

possibleBlob.currentBoundingRect.height > 30 &&

possibleBlob.dblCurrentDiagonalSize > 60.0 &&

(cv::contourArea(possibleBlob.currentContour) / (double)possibleBlob.currentBoundingRect.area()) > 0.50) {

currentFrameBlobs.push\_back(possibleBlob);

}

}

drawAndShowContours(imgThresh.size(), currentFrameBlobs, "imgCurrentFrameBlobs");

if (blnFirstFrame == true) {

for (auto &currentFrameBlob : currentFrameBlobs) {

blobs.push\_back(currentFrameBlob);

}

}

else {

matchCurrentFrameBlobsToExistingBlobs(blobs, currentFrameBlobs);

}

drawAndShowContours(imgThresh.size(), blobs, "imgBlobs");

imgFrame2Copy = imgFrame2.clone(); // Lấy một bản sao khác của khung 2 vì chúng tôi đã thay đổi bản sao của khung 2 trước đó trong quá trình xử lý ở trên

drawBlobInfoOnImage(blobs, imgFrame2Copy);

bool blnAtLeastOneBlobCrossedTheLine = checkIfBlobsCrossedTheLine(blobs, intHorizontalLinePosition, carCount);

if (blnAtLeastOneBlobCrossedTheLine == true) {

cv::line(imgFrame2Copy, crossingLine[0], crossingLine[1], SCALAR\_GREEN, 2);

}

else {

cv::line(imgFrame2Copy, crossingLine[0], crossingLine[1], SCALAR\_RED, 2);

}

drawCarCountOnImage(carCount, imgFrame2Copy);

cv::imshow("imgFrame2Copy", imgFrame2Copy);

//cv::waitKey(0); // Bỏ ghi chú dòng này để đi từng khung hình để gỡ lỗi

// Bây giờ chúng ta chuẩn bị cho lần lặp tiếp theo

currentFrameBlobs.clear();

imgFrame1 = imgFrame2.clone(); // Di chuyển khung 1 lên đến vị trí của khung 2

if ((capVideo.get(CV\_CAP\_PROP\_POS\_FRAMES) + 1) < capVideo.get(CV\_CAP\_PROP\_FRAME\_COUNT)) {

capVideo.read(imgFrame2);

}

else {

std::cout << "end of video\n";

break;

}

blnFirstFrame = false;

frameCount++;

chCheckForEscKey = cv::waitKey(1);

}

if (chCheckForEscKey != 27) { // Nếu người dùng không nhấn esc (tức là đã đến cuối video)

cv::waitKey(0); // giữ các cửa sổ mở để cho phép thông báo "kết thúc video" hiển thị

}

// lưu ý rằng nếu người dùng đã nhấn esc, chúng ta không cần phải mở các cửa sổ, chúng ta chỉ cần để chương trình kết thúc sẽ đóng các cửa sổ

return(0);

}

void matchCurrentFrameBlobsToExistingBlobs(std::vector<Blob> &existingBlobs, std::vector<Blob> &currentFrameBlobs) // Hàm tạo blob hiện tại phù hợp với blob đã có

{

for (auto &existingBlob : existingBlobs) {

existingBlob.blnCurrentMatchFoundOrNewBlob = false;

existingBlob.predictNextPosition();

}

for (auto &currentFrameBlob : currentFrameBlobs) {

int intIndexOfLeastDistance = 0;

double dblLeastDistance = 100000.0;

for (unsigned int i = 0; i < existingBlobs.size(); i++) {

if (existingBlobs[i].blnStillBeingTracked == true) {

double dblDistance = distanceBetweenPoints(currentFrameBlob.centerPositions.back(), existingBlobs[i].predictedNextPosition);

if (dblDistance < dblLeastDistance) {

dblLeastDistance = dblDistance;

intIndexOfLeastDistance = i;

}

}

}

if (dblLeastDistance < currentFrameBlob.dblCurrentDiagonalSize \* 0.5) {

addBlobToExistingBlobs(currentFrameBlob, existingBlobs, intIndexOfLeastDistance);

}

else {

addNewBlob(currentFrameBlob, existingBlobs);

}

}

for (auto &existingBlob : existingBlobs) {

if (existingBlob.blnCurrentMatchFoundOrNewBlob == false) {

existingBlob.intNumOfConsecutiveFramesWithoutAMatch++;

}

if (existingBlob.intNumOfConsecutiveFramesWithoutAMatch >= 5) {

existingBlob.blnStillBeingTracked = false;

}

}

}

void addBlobToExistingBlobs(Blob &currentFrameBlob, std::vector<Blob> &existingBlobs, int &intIndex) // Hàm thêm Blob vào Blobs hiện có

{

existingBlobs[intIndex].currentContour = currentFrameBlob.currentContour;

existingBlobs[intIndex].currentBoundingRect = currentFrameBlob.currentBoundingRect;

existingBlobs[intIndex].centerPositions.push\_back(currentFrameBlob.centerPositions.back());

existingBlobs[intIndex].dblCurrentDiagonalSize = currentFrameBlob.dblCurrentDiagonalSize;

existingBlobs[intIndex].dblCurrentAspectRatio = currentFrameBlob.dblCurrentAspectRatio;

existingBlobs[intIndex].blnStillBeingTracked = true;

existingBlobs[intIndex].blnCurrentMatchFoundOrNewBlob = true;

}

void addNewBlob(Blob &currentFrameBlob, std::vector<Blob> &existingBlobs) // Hàm tự động thêm giá trị blob

{

currentFrameBlob.blnCurrentMatchFoundOrNewBlob = true;

existingBlobs.push\_back(currentFrameBlob);

}

double distanceBetweenPoints(cv::Point point1, cv::Point point2) // Hàm khoảng cách giữa 2 điểm

{

int intX = abs(point1.x - point2.x);

int intY = abs(point1.y - point2.y);

return(sqrt(pow(intX, 2) + pow(intY, 2)));

}

void drawAndShowContours(cv::Size imageSize, std::vector<std::vector<cv::Point> > contours, std::string strImageName) // Hàm định dạng đường viền màu trắng

{

cv::Mat image(imageSize, CV\_8UC3, SCALAR\_BLACK);

cv::drawContours(image, contours, -1, SCALAR\_WHITE, -1);

cv::imshow(strImageName, image);

}

void drawAndShowContours(cv::Size imageSize, std::vector<Blob> blobs, std::string strImageName) // Hàm định dạng đường viền màu đen

{

cv::Mat image(imageSize, CV\_8UC3, SCALAR\_BLACK);

std::vector<std::vector<cv::Point> > contours;

for (auto &blob : blobs) {

if (blob.blnStillBeingTracked == true) {

contours.push\_back(blob.currentContour);

}

}

cv::drawContours(image, contours, -1, SCALAR\_WHITE, -1);

cv::imshow(strImageName, image);

}

bool checkIfBlobsCrossedTheLine(std::vector<Blob> &blobs, int &intHorizontalLinePosition, int &carCount) // Hàm tạo đường thẳng để nhận dạng số xe đi qua đường chỉ áp dụng cho 1 mặt đường

{

bool blnAtLeastOneBlobCrossedTheLine = false;

for (auto blob : blobs) {

if (blob.blnStillBeingTracked == true && blob.centerPositions.size() >= 2) {

int prevFrameIndex = (int)blob.centerPositions.size() - 2;

int currFrameIndex = (int)blob.centerPositions.size() - 1;

if (blob.centerPositions[prevFrameIndex].y > intHorizontalLinePosition && blob.centerPositions[currFrameIndex].y <= intHorizontalLinePosition) {

carCount++;

blnAtLeastOneBlobCrossedTheLine = true;

}

}

}

return blnAtLeastOneBlobCrossedTheLine;

}

void drawBlobInfoOnImage(std::vector<Blob> &blobs, cv::Mat &imgFrame2Copy) // Hàm tạo các đường blob trong video gốc

{

for (unsigned int i = 0; i < blobs.size(); i++) {

if (blobs[i].blnStillBeingTracked == true) {

cv::rectangle(imgFrame2Copy, blobs[i].currentBoundingRect, SCALAR\_RED, 2);

int intFontFace = CV\_FONT\_HERSHEY\_SIMPLEX;

double dblFontScale = blobs[i].dblCurrentDiagonalSize / 60.0;

int intFontThickness = (int)std::round(dblFontScale \* 1.0);

cv::putText(imgFrame2Copy, std::to\_string(i), blobs[i].centerPositions.back(), intFontFace, dblFontScale, SCALAR\_GREEN, intFontThickness);

}

}

}

void drawCarCountOnImage(int &carCount, cv::Mat &imgFrame2Copy) // Hàm nhận dạng số lượng xe, mã số xe theo thứ tự

{

int intFontFace = CV\_FONT\_HERSHEY\_SIMPLEX;

double dblFontScale = (imgFrame2Copy.rows \* imgFrame2Copy.cols) / 300000.0;

int intFontThickness = (int)std::round(dblFontScale \* 1.5);

cv::Size textSize = cv::getTextSize(std::to\_string(carCount), intFontFace, dblFontScale, intFontThickness, 0);

cv::Point ptTextBottomLeftPosition;

ptTextBottomLeftPosition.x = imgFrame2Copy.cols - 1 - (int)((double)textSize.width \* 1.25);

ptTextBottomLeftPosition.y = (int)((double)textSize.height \* 1.25);

cv::putText(imgFrame2Copy, std::to\_string(carCount), ptTextBottomLeftPosition, intFontFace, dblFontScale, SCALAR\_GREEN, intFontThickness);

}

**PYTHON**

import cv2

cascade\_src = 'cars.xml'

video\_src = 'video.avi'

cap = cv2.VideoCapture(video\_src)

car\_cascade = cv2.CascadeClassifier(cascade\_src) #cv2.CascaseClassifier() là một trong các mô hình được sàng lọc được đặt trong thư mục dữ liệu trong bản cài đặt OpenCV hoặc có thể tìm thấy ở đây

while True:

ret, img = cap.read()

if (type(img) == type(None)):

break

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) # Lấy ảnh xám

cars = car\_cascade.detectMultiScale(gray, 1.1, 2) # Hàm nhận dạng xe của file xml

for (x,y,w,h) in cars: # Định dạng khung hình

cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),2)

cv2.imshow('video', img)

if cv2.waitKey(33) == 27:

break

cv2.destroyAllWindows()

## 3.18: Nhận dạng biển số xe

**C++**

DetectChars.h, DetectPlates.h, PossibleChar.h, PossiblePlate.h, Preprocess.h

DetectChars.cpp, DetectPlates.cpp, PossibleChar.cpp, PossiblePlate.cpp, Preprocess.cpp (Các chường trình khởi tạo biển báo trên thì mình không đề cập cụ thể ở đây vì lý do nội dung rất dài, chỉ đề cập phần chính yếu)

Main.h

#ifndef MY\_MAIN // Đã sử dụng MY\_MAIN cho việc này bao gồm bảo vệ thay vì MAIN chỉ trong trường hợp một số trình biên dịch hoặc môi trường #define MAIN

#define MY\_MAIN

#include<opencv2/core/core.hpp>

#include<opencv2/highgui/highgui.hpp>

#include<opencv2/imgproc/imgproc.hpp>

#include "DetectPlates.h"

#include "PossiblePlate.h"

#include "DetectChars.h"

#include<iostream>

#include<conio.h> // Có thể phải sửa đổi dòng này nếu không sử dụng Windows

// hằng số toàn cầu

const cv::Scalar SCALAR\_BLACK = cv::Scalar(0.0, 0.0, 0.0);

const cv::Scalar SCALAR\_WHITE = cv::Scalar(255.0, 255.0, 255.0);

const cv::Scalar SCALAR\_YELLOW = cv::Scalar(0.0, 255.0, 255.0);

const cv::Scalar SCALAR\_GREEN = cv::Scalar(0.0, 255.0, 0.0);

const cv::Scalar SCALAR\_RED = cv::Scalar(0.0, 0.0, 255.0);

// Hàm nguyên mẫu

int main(void);

void drawRedRectangleAroundPlate(cv::Mat &imgOriginalScene, PossiblePlate &licPlate);

void writeLicensePlateCharsOnImage(cv::Mat &imgOriginalScene, PossiblePlate &licPlate);

# endif

Main.cpp

#include "Main.h"

int main(void) {

bool blnKNNTrainingSuccessful = loadKNNDataAndTrainKNN();

if (blnKNNTrainingSuccessful == false) { // Nếu thử KNNTraining không thành công

// hiển thị thông báo lỗi

std::cout << std::endl << std::endl << "error: error: KNN traning was not successful" << std::endl << std::endl;

return(0);

}

cv::Mat imgOriginalScene; // Nhập ảnh

imgOriginalScene = cv::imread("image1.png"); // Mở ảnh

if (imgOriginalScene.empty()) { // Nếu không mở được ảnh

std::cout << "error: image not read from file\n\n"; // Hiển thị thông báo lỗi trên dòng lệnh

\_getch(); // Có thể phải sửa đổi dòng này nếu không sử dụng Windows

return(0);

}

std::vector<PossiblePlate> vectorOfPossiblePlates = detectPlatesInScene(imgOriginalScene); // Tấm được nhận dạng

vectorOfPossiblePlates = detectCharsInPlates(vectorOfPossiblePlates); // Phát hiện ký tự trong các tấm

cv::imshow("imgOriginalScene", imgOriginalScene); // Hiển thị hình ảnh cảnh

if (vectorOfPossiblePlates.empty()) { // Nếu không tìm thấy tấm nào

std::cout << std::endl << "no license plates were detected" << std::endl; // Thông báo cho người dùng không tìm thấy tấm nào

}

else {

// Sắp xếp vectơ của các tấm có thể theo thứ tự MÔ TẢ (số lượng ký tự nhiều nhất đến số lượng ký tự ít nhất)

std::sort(vectorOfPossiblePlates.begin(), vectorOfPossiblePlates.end(), PossiblePlate::sortDescendingByNumberOfChars);

// Giả sử tấm có ký tự được nhận biết nhiều nhất (tấm đầu tiên được sắp xếp theo thứ tự giảm dần độ dài chuỗi) là tấm thực tế

PossiblePlate licPlate = vectorOfPossiblePlates.front();

cv::imshow("imgPlate", licPlate.imgPlate); // Hiển thị crop của tấm và ngưỡng của tấm

cv::imshow("imgThresh", licPlate.imgThresh);

if (licPlate.strChars.length() == 0) { // Nếu không tìm thấy ký tự trong bảng

std::cout << std::endl << "no characters were detected" << std::endl << std::endl; // Hiển thị thông báo

return(0);

}

drawRedRectangleAroundPlate(imgOriginalScene, licPlate); // Vẽ hình chữ nhật màu đỏ xung quanh tấm

std::cout << std::endl << "license plate read from image = " << licPlate.strChars << std::endl; // write license plate text to std out

std::cout << std::endl << "-----------------------------------------" << std::endl;

writeLicensePlateCharsOnImage(imgOriginalScene, licPlate); // Viết văn bản biển số trên hình ảnh

cv::imshow("imgOriginalScene", imgOriginalScene); // Hiển thị lại hình ảnh cảnh

cv::imwrite("imgOriginalScene.png", imgOriginalScene); // Ghi hình ảnh ra tập tin

}

cv::waitKey(0); // Giữ cửa sổ mở cho đến khi người dùng nhấn phím

return(0);

}

void drawRedRectangleAroundPlate(cv::Mat &imgOriginalScene, PossiblePlate &licPlate) {

cv::Point2f p2fRectPoints[4];

licPlate.rrLocationOfPlateInScene.points(p2fRectPoints); // Lấy 4 đỉnh của trực trục xoay

for (int i = 0; i < 4; i++) { // Vẽ 4 đường màu đỏ

cv::line(imgOriginalScene, p2fRectPoints[i], p2fRectPoints[(i + 1) % 4], SCALAR\_RED, 2);

}

}

void writeLicensePlateCharsOnImage(cv::Mat &imgOriginalScene, PossiblePlate &licPlate) {

cv::Point ptCenterOfTextArea; // Đây sẽ là trung tâm của khu vực văn bản sẽ được viết

cv::Point ptLowerLeftTextOrigin; // Đây sẽ là phía dưới bên trái của khu vực mà văn bản sẽ được viết

int intFontFace = CV\_FONT\_HERSHEY\_SIMPLEX; // Chọn một phông chữ jane đơn giản

double dblFontScale = (double)licPlate.imgPlate.rows / 30.0; // Tỷ lệ phông chữ cơ sở trên chiều cao của khu vực tấm

int intFontThickness = (int)std::round(dblFontScale \* 1.5); // Độ dày phông chữ cơ bản trên tỷ lệ phông chữ

int intBaseline = 0;

cv::Size textSize = cv::getTextSize(licPlate.strChars, intFontFace, dblFontScale, intFontThickness, &intBaseline); // Gọi hàm getTextSize

ptCenterOfTextArea.x = (int)licPlate.rrLocationOfPlateInScene.center.x; // Vị trí ngang của vùng văn bản giống với tấm

if (licPlate.rrLocationOfPlateInScene.center.y < (imgOriginalScene.rows \* 0.75)) { // Nếu biển số xe nằm ở 3/4 trên của hình ảnh

// Viết các ký tự bên dưới tấm

ptCenterOfTextArea.y = (int)std::round(licPlate.rrLocationOfPlateInScene.center.y) + (int)std::round((double)licPlate.imgPlate.rows \* 1.6);

}

else { // Khác nếu biển số xe nằm ở 1/4 dưới của hình ảnh // Viết các ký tự ở trên tấm

ptCenterOfTextArea.y = (int)std::round(licPlate.rrLocationOfPlateInScene.center.y) - (int)std::round((double)licPlate.imgPlate.rows \* 1.6);

}

ptLowerLeftTextOrigin.x = (int)(ptCenterOfTextArea.x - (textSize.width / 2)); // Tính toán gốc bên trái của vùng văn bản

ptLowerLeftTextOrigin.y = (int)(ptCenterOfTextArea.y + (textSize.height / 2)); // Dựa trên trung tâm vùng văn bản, chiều rộng và chiều cao

// Viết văn bản lên hình ảnh

cv::putText(imgOriginalScene, licPlate.strChars, ptLowerLeftTextOrigin, intFontFace, dblFontScale, SCALAR\_YELLOW, intFontThickness);

}

## 3.19: Điều khiển chuột máy tính

**PYTHON**

import cv2

import numpy as np

import imutils

from pynput.mouse import Button, Controller # Thư viện hỗ trợ các nút bấm, điều khiển chuột

import wx

mouse= Controller()

app = wx.App(False)

(sx,sy) = wx.GetDisplaySize()

(camx,camy) = (420,340)

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

#YCrCb = Y: thành phần độ chói, Cr: màu đỏ chrominat, Cb: màu xanh lam

min\_YCrCb = np.array([0, 131, 100],np.uint8)

max\_YCrCb = np.array([255, 185, 135],np.uint8)

kernelopen= np.ones((1,1))

kernelclose= np.ones((1,1))

kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(3,3)) #Hình nhận dạng với hình đa giác tự động với các điểm kernel

fgbg = cv2.createBackgroundSubtractorMOG2(history = 20, varThreshold =25)

# Là hàm trả về số lượng khung hình cuối cùng ảnh hưởng đến mô hình nền

cam = cv2.VideoCapture(0) # Khởi động camera

cam.set(3,camx)

cam.set(4,camy)

while True:

ret, img = cam.read()

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) # Xử lý các màu sang xám

gray = cv2.GaussianBlur(gray, (5, 5), 0) # xử dụng bộ lọc gauss cho ảnh xám

faces = face\_cascade.detectMultiScale(gray, 1.3, 5) # Hàm hỗ trợ nhận dạng mặt vật thể từ ảnh xám

for(x,y,w,h) in faces : # Tạo điểm cho khung hình nhận dạng

cv2.rectangle(img, (x,y), (x+w,y+h), (0,0,0), -1)

roi\_gray = gray[y:y+h, x:x+w]

roi\_color = img[y:y+h, x:x+w]

imageYCrCb = cv2.cvtColor(img,cv2.COLOR\_BGR2YCR\_CB)

skinRegion = cv2.inRange(imageYCrCb,min\_YCrCb,max\_YCrCb)

maskopen = cv2.morphologyEx(skinRegion,cv2.MORPH\_OPEN, kernelopen)

maskclose = cv2.morphologyEx(maskopen,cv2.MORPH\_OPEN, kernelclose)

contours, hierarchy = cv2.findContours(maskclose.copy(), cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

cnt = max(contours, key=lambda x:cv2.contourArea(x))

x,y,w,h = cv2.boundingRect(cnt)

cv2.rectangle(img,(x,y), (x+w, y+h-40), (0,0,255), 0)

hull = cv2.convexHull(cnt)

drawing = np.zeros(img.shape, np.uint8)

cv2.drawContours(drawing, [cnt], -1, (0,255,0), 0)

cv2.drawContours(drawing, [hull], -1, (0,0,255), 0)

cx = x+w//2

cy = y+h//2

cv2.circle(img,(cx,cy),2,(255,0,0),2)

mouse.position = (sx-(cx\*sx//camx), cy\*sy//camy)

while mouse.position != (sx-(cx\*sx//camx), cy\*sy//camy):

pass

cv2.imshow('img',img)

cv2.imshow("maskclose",maskclose)

all\_img = np.hstack((drawing, img))

cv2.imshow('last', all\_img)

key = cv2.waitKey(1)

if key == ord('q'):

break

cv2.destroyAllWindows()

cam.release()

## 3.20 Nhập chữ bằng bàn phím qua hình ảnh camera

**PYTHON (Xem video sau để hiểu rõ chi tiết hơn “https://www.youtube.com/playlist?list=PL6Yc5OUgcoTlvHb5OfFLUJ90ofBuoU5g8”)**

import cv2

import numpy as np

import dlib

from math import hypot

import pyglet

import time

# Tải file âm thanh

sound = pyglet.media.load("sound.wav", streaming=False)

left\_sound = pyglet.media.load("left.wav", streaming=False)

right\_sound = pyglet.media.load("right.wav", streaming=False)

cap = cv2.VideoCapture(0) # Mở camera

board = np.zeros((500, 500), np.uint8)

board[:] = 255

detector = dlib.get\_frontal\_face\_detector()

predictor = dlib.shape\_predictor("shape\_predictor\_68\_face\_landmarks.dat")

# file .dat giúp hỗ trợ chữ cái bàn phím

# Cài đặt bàn phím

keyboard = np.zeros((600, 700, 3), np.uint8)

keys\_set\_1 = {0: "A", 1: "B", 2: "C", 3: "D", 4: "E",

5: "F", 6: "G", 7: "H", 8: "I", 9: "J",

10: "K", 11: "L", 12: "M", 13: "N", 14: "O",

15: "P", 16: "Q", 17: "R", 18: "S", 19: "T",

20: "U", 21: "V", 22: "W", 23: "X", 24: "Y",

25: "Z"}

def letter(letter\_index, text, letter\_light):

# Khóa thứ tự cho bảng chữ cái

if letter\_index == 0:

x = 0

y = 0

elif letter\_index == 1:

x = 100

y = 0

elif letter\_index == 2:

x = 200

y = 0

elif letter\_index == 3:

x = 300

y = 0

elif letter\_index == 4:

x = 400

y = 0

elif letter\_index == 5:

x = 0

y = 100

elif letter\_index == 6:

x = 100

y = 100

elif letter\_index == 7:

x = 200

y = 100

elif letter\_index == 8:

x = 300

y = 100

elif letter\_index == 9:

x = 400

y = 100

elif letter\_index == 10:

x = 0

y = 200

elif letter\_index == 11:

x = 100

y = 200

elif letter\_index == 12:

x = 200

y = 200

elif letter\_index == 13:

x = 300

y = 200

elif letter\_index == 14:

x = 400

y = 200

elif letter\_index == 15:

x = 0

y = 300

elif letter\_index == 16:

x = 100

y = 300

elif letter\_index == 17:

x = 200

y = 300

elif letter\_index == 18:

x = 300

y = 300

elif letter\_index == 19:

x = 400

y = 300

elif letter\_index == 20:

x = 0

y = 400

elif letter\_index == 21:

x = 100

y = 400

elif letter\_index == 22:

x = 200

y = 400

elif letter\_index == 23:

x = 300

y = 400

elif letter\_index == 24:

x = 400

y = 400

elif letter\_index == 25:

x = 0

y = 500

width = 100 # Độ rộng

height = 100 # Độ cao

th = 3 # Độ dày

if letter\_light is True:

cv2.rectangle(keyboard, (x + th, y + th), (x + width - th, y + height - th), (255, 255, 255), -1)

else:

cv2.rectangle(keyboard, (x + th, y + th), (x + width - th, y + height - th), (255, 0, 0), th)

# Cài đặt chữ cái

font\_letter = cv2.FONT\_HERSHEY\_PLAIN

font\_scale = 10

font\_th = 4

text\_size = cv2.getTextSize(text, font\_letter, font\_scale, font\_th)[0]

width\_text, height\_text = text\_size[0], text\_size[1]

text\_x = int((width - width\_text) / 2) + x

text\_y = int((height + height\_text) / 2) + y

cv2.putText(keyboard, text, (text\_x, text\_y), font\_letter, font\_scale, (255, 0, 0), font\_th)

def midpoint(p1 ,p2): # Tìm điểm trung tâm

return int((p1.x + p2.x)/2), int((p1.y + p2.y)/2)

font = cv2.FONT\_HERSHEY\_PLAIN

def get\_blinking\_ratio(eye\_points, facial\_landmarks): # Thiết lập điểm mắt

left\_point = (facial\_landmarks.part(eye\_points[0]).x, facial\_landmarks.part(eye\_points[0]).y)

right\_point = (facial\_landmarks.part(eye\_points[3]).x, facial\_landmarks.part(eye\_points[3]).y)

center\_top = midpoint(facial\_landmarks.part(eye\_points[1]), facial\_landmarks.part(eye\_points[2]))

center\_bottom = midpoint(facial\_landmarks.part(eye\_points[5]), facial\_landmarks.part(eye\_points[4]))

#hor\_line = cv2.line(frame, left\_point, right\_point, (0, 255, 0), 2)

#ver\_line = cv2.line(frame, center\_top, center\_bottom, (0, 255, 0), 2)

hor\_line\_lenght = hypot((left\_point[0] - right\_point[0]), (left\_point[1] - right\_point[1]))

ver\_line\_lenght = hypot((center\_top[0] - center\_bottom[0]), (center\_top[1] - center\_bottom[1]))

ratio = hor\_line\_lenght / ver\_line\_lenght

return ratio

def get\_gaze\_ratio(eye\_points, facial\_landmarks): # Kích thước, tỉ lệ các điểm

left\_eye\_region = np.array([(facial\_landmarks.part(eye\_points[0]).x, facial\_landmarks.part(eye\_points[0]).y),

(facial\_landmarks.part(eye\_points[1]).x, facial\_landmarks.part(eye\_points[1]).y),

(facial\_landmarks.part(eye\_points[2]).x, facial\_landmarks.part(eye\_points[2]).y),

(facial\_landmarks.part(eye\_points[3]).x, facial\_landmarks.part(eye\_points[3]).y),

(facial\_landmarks.part(eye\_points[4]).x, facial\_landmarks.part(eye\_points[4]).y),

(facial\_landmarks.part(eye\_points[5]).x, facial\_landmarks.part(eye\_points[5]).y)], np.int32)

# cv2.polylines(frame, [left\_eye\_region], True, (0, 0, 255), 2)

height, width, \_ = frame.shape

mask = np.zeros((height, width), np.uint8)

cv2.polylines(mask, [left\_eye\_region], True, 255, 2)

cv2.fillPoly(mask, [left\_eye\_region], 255)

eye = cv2.bitwise\_and(gray, gray, mask=mask)

min\_x = np.min(left\_eye\_region[:, 0])

max\_x = np.max(left\_eye\_region[:, 0])

min\_y = np.min(left\_eye\_region[:, 1])

max\_y = np.max(left\_eye\_region[:, 1])

gray\_eye = eye[min\_y: max\_y, min\_x: max\_x]

\_, threshold\_eye = cv2.threshold(gray\_eye, 70, 255, cv2.THRESH\_BINARY)

height, width = threshold\_eye.shape

left\_side\_threshold = threshold\_eye[0: height, 0: int(width / 2)]

left\_side\_white = cv2.countNonZero(left\_side\_threshold)

right\_side\_threshold = threshold\_eye[0: height, int(width / 2): width]

right\_side\_white = cv2.countNonZero(right\_side\_threshold)

if left\_side\_white == 0:

gaze\_ratio = 1

elif right\_side\_white == 0:

gaze\_ratio = 5

else:

gaze\_ratio = left\_side\_white / right\_side\_white

return gaze\_ratio

# Bộ đếm

frames = 0

letter\_index = 0

blinking\_frames = 0

text = ""

keyboard\_selected = "left"

last\_keyboard\_selected = "left"

while True:

\_, frame = cap.read()

frame = cv2.resize(frame, None, fx=0.5, fy=0.5)

keyboard[:] = (0, 0, 0)

frames += 1

new\_frame = np.zeros((500, 500, 3), np.uint8)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

active\_letter = keys\_set\_1[letter\_index]

faces = detector(gray)

for face in faces:

#x, y = face.left(), face.top()

#x1, y1 = face.right(), face.bottom()

#cv2.rectangle(frame, (x, y), (x1, y1), (0, 255, 0), 2)

landmarks = predictor(gray, face)

# Nhận dạng nháy mắt

left\_eye\_ratio = get\_blinking\_ratio([36, 37, 38, 39, 40, 41], landmarks)

right\_eye\_ratio = get\_blinking\_ratio([42, 43, 44, 45, 46, 47], landmarks)

blinking\_ratio = (left\_eye\_ratio + right\_eye\_ratio) / 2

if blinking\_ratio > 5.7:

cv2.putText(frame, "BLINKING", (50, 150), font, 4, (255, 0, 0), thickness=3)

blinking\_frames += 1

frames -= 1

# Đánh chữ

if blinking\_frames == 5:

text += active\_letter

sound.play()

time.sleep(1)

else:

blinking\_frames = 0

# Phát hiện ánh mắt

gaze\_ratio\_left\_eye = get\_gaze\_ratio([36, 37, 38, 39, 40, 41], landmarks)

gaze\_ratio\_right\_eye = get\_gaze\_ratio([42, 43, 44, 45, 46, 47], landmarks)

gaze\_ratio = (gaze\_ratio\_right\_eye + gaze\_ratio\_left\_eye) / 2

if gaze\_ratio <= 0.9:

keyboard\_selected = "right"

if keyboard\_selected != last\_keyboard\_selected:

right\_sound.play()

time.sleep(1)

last\_keyboard\_selected = keyboard\_selected

else:

keyboard\_selected = "left"

if keyboard\_selected != last\_keyboard\_selected:

left\_sound.play()

time.sleep(1)

last\_keyboard\_selected = keyboard\_selected

# Chữ

if frames == 26:

letter\_index += 1

frames = 0

if letter\_index == 26:

letter\_index = 0

for i in range(26):

if i == letter\_index:

light = True

else:

light = False

letter(i, keys\_set\_1[i], light)

cv2.putText(board, text, (10, 100), font, 4, 0, 3)

cv2.imshow("Frame", frame)

cv2.imshow("Virtual keyboard", keyboard)

cv2.imshow("Board", board)

key = cv2.waitKey(1)

if key == 27:

break

cap.release()

cv2.destroyAllWindows()

# CHƯƠNG 4: KẾT LUẬN

TRONG CÁC PHẦN LÀM FILE BÀI TẬP THÌ MÌNH CÓ NHỮNG CHIA SẼ SAU:

1. Các bài tập có sự tìm hiểu đơn giản chưa cụ thể, nhưng ai muốn hiểu cụ thể có thể tham khảo các link trang web mình đã đưa kèm bài tập tương ứng.
2. Phần file bài tập, đa số đều có đầy đủ cả C++ và Python. Chỉ có 3 phần cuối là có 1 trong 2 ngôn ngữ là C++ và Python.
3. Phần chữ màu lục là phần chú thích cho phần code bên cạnh cho từng bài.
4. Có những câu lệnh lặp lại giống nhau từ các file chương trình trước thì mình có thể sẽ bỏ qua trong lần lặp lại tới, những phần không quan trọng thì các bạn có thể tìm hiểu trong qua tài liệu tham khảo.
5. Cách mở file C++ để chạy chương trình:

Thay vì ta phải mở Visual studio qua file .sln (visual studio solution) thì chúng ta mở file .exe trực tiếp chạy ngay mà không mất thời gian bằng cách như sau:

Ví dụ : với file “mở ảnh” ta chọn “C++” ->”open\_image” có 3 file (open\_image, open\_image.sln)

Trong đó: “open\_image” ta mở ra sẽ có các file cpp, hình ảnh, video, xml,.. tùy file bài tập

“x64” là phần ta đã chạy được trên visual studio nên có file này

“open\_image.sln” là file project visual studio gốc

Trong các phần này ta chọn “x64” rồi chọn “Debug”, có nhieuf file khác nhau ta chọn file “open\_image.exe” để chạy trực tiếp.

1. Cách mở file chạy chương trình Python trực tiếp:

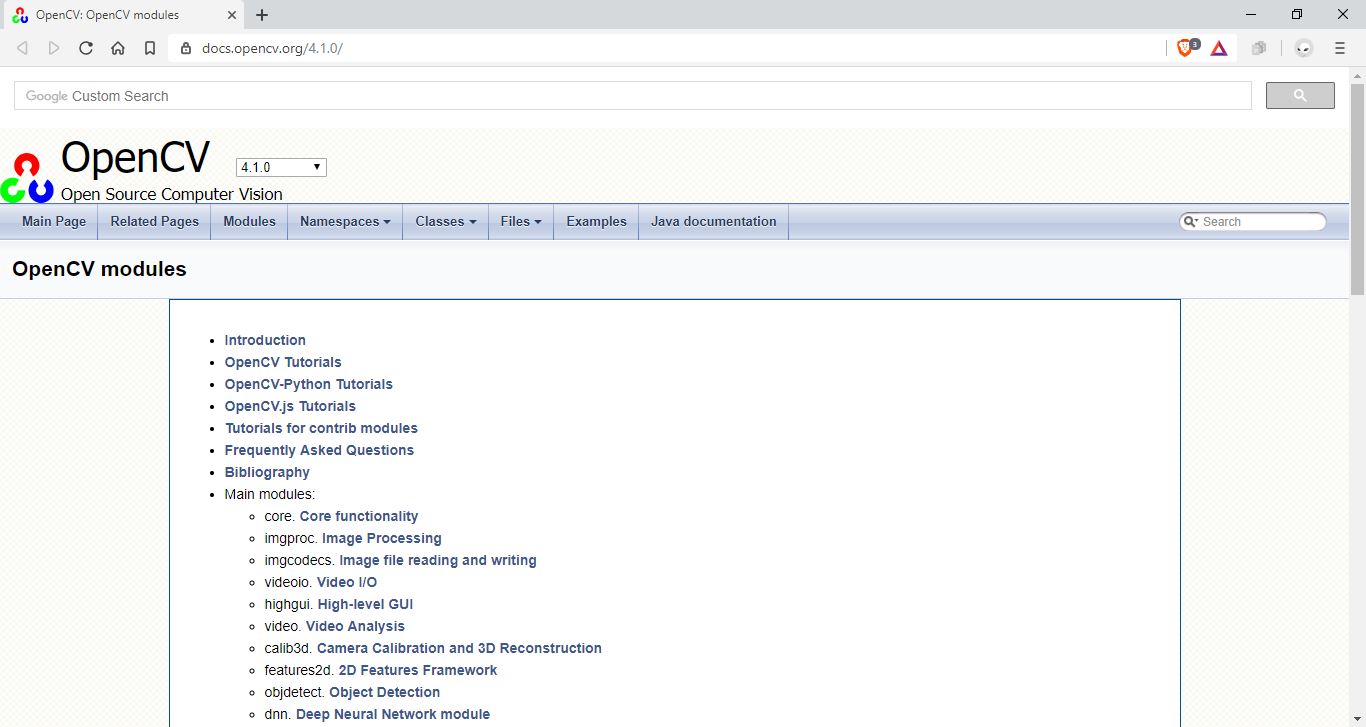
Ta chọn file python trực tiếp trên folder

Ví dụ : với file “mở ảnh” ta chọn “PYTHON” có 2 file hình ảnh .jpg và file python .py ta click chuột trực tiếp vào file này để chạy định dạng .exe, còn muốn xem nội dung code thì ta click chuột phải rồi chọn “Edit with IDLE”->” Edit with IDLE 3.6 (64-bit)” thì sẽ hiện ra phần nội dung code.

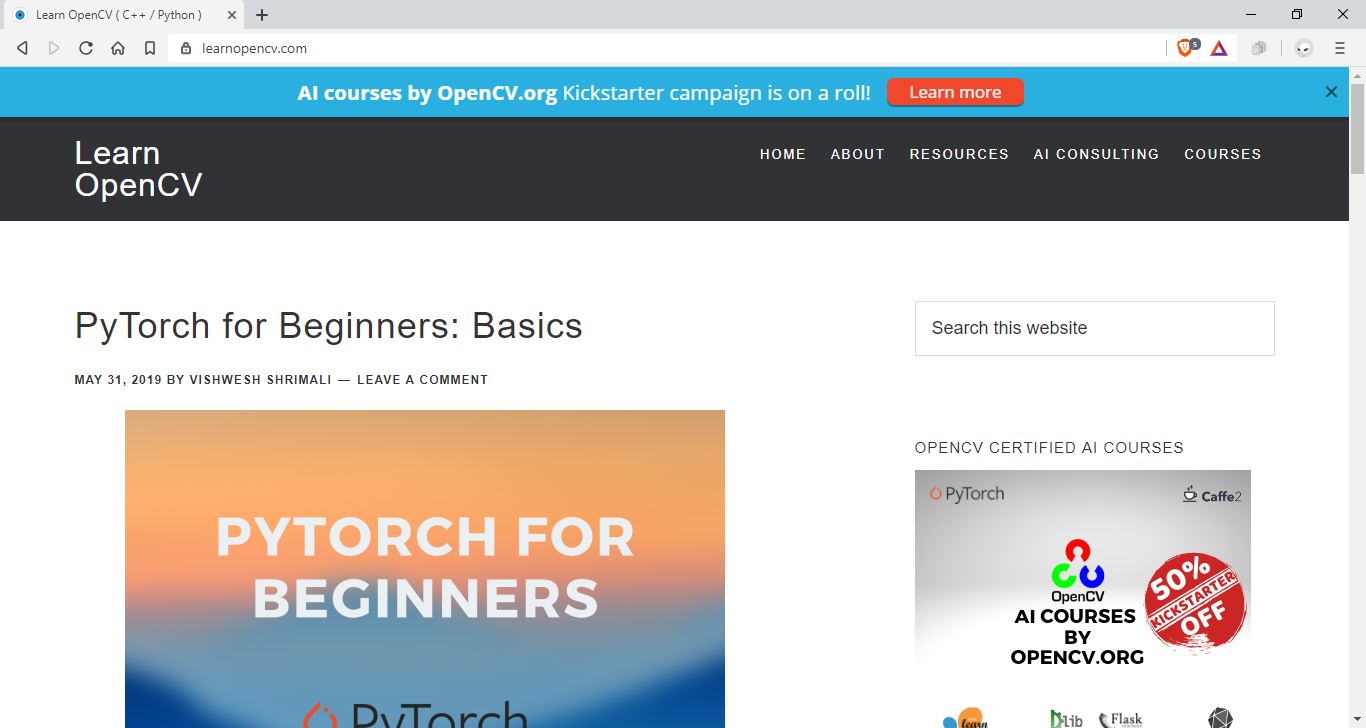
1. Đây chỉ là file word báo cáo nội dung thực hành môn xử lý ảnh bao gồm:”hướng dẫn cài đặt và chạy chương trình, giải thích nội dung cho các dòng lệnh code của chương trình căn bản” nên chỉ là những gì mình đã học được, biết được về môn này, không phải cuốn tài liệu gì cả, nên mình sẵn sàng chia sẻ về nội dung báo cáo này, hãy coi đây như là file chia sẻ bài tập tham khảo cho ai cần. Cảm ơn đã xem.

# TÀI LIỆU THAM KHẢO

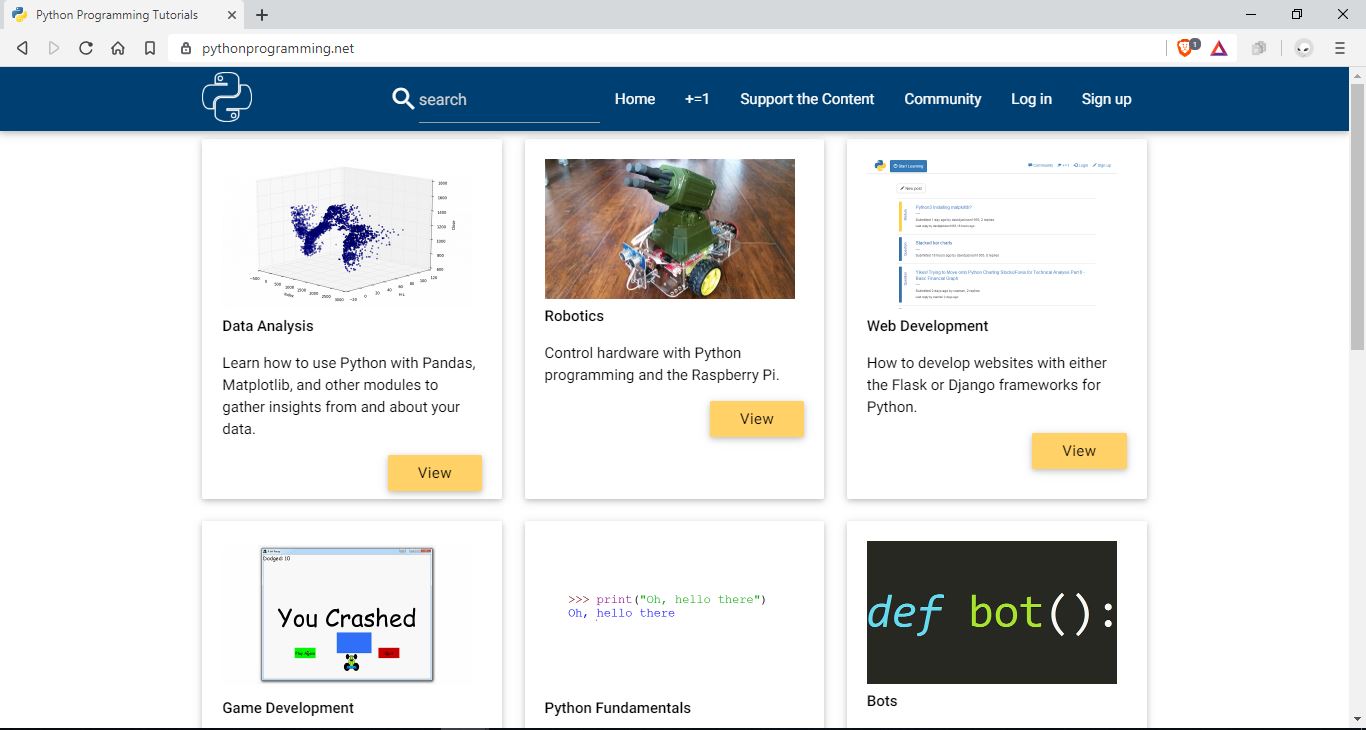
1. <https://docs.opencv.org/4.1.0/>



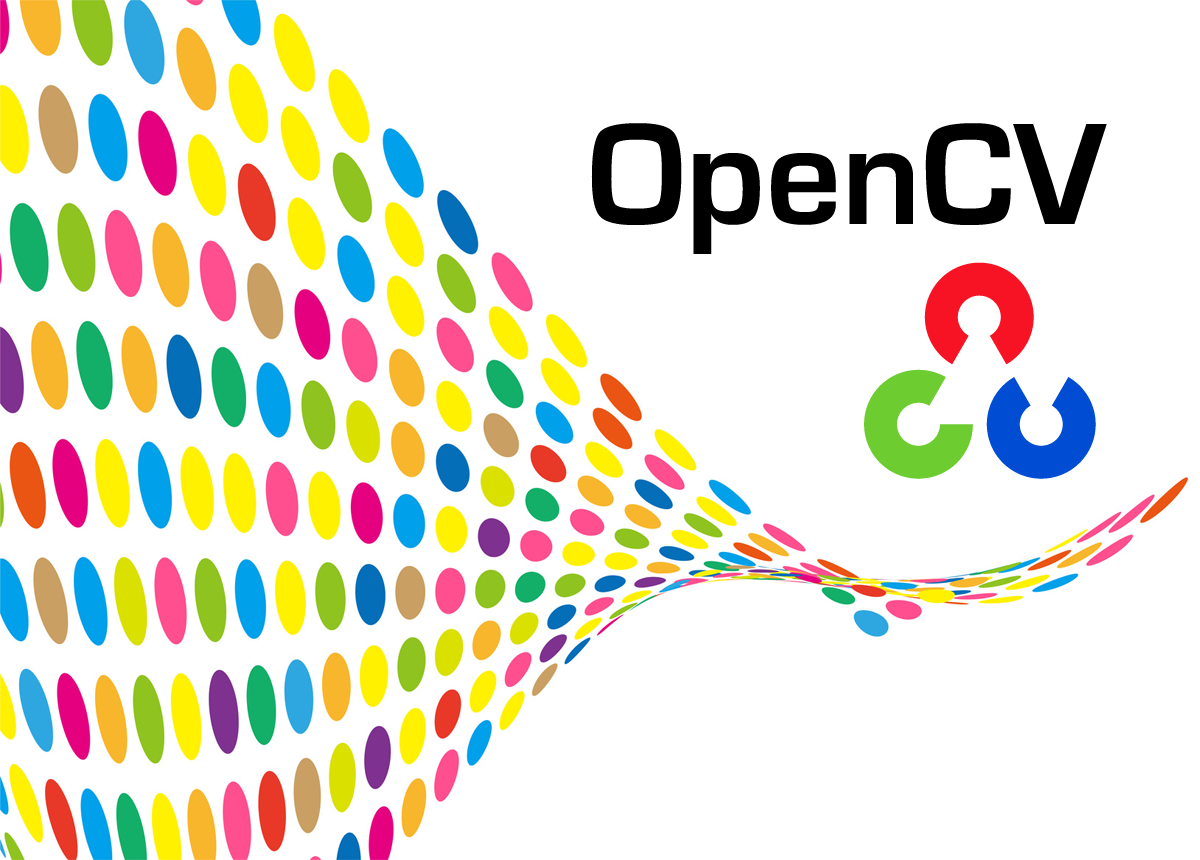
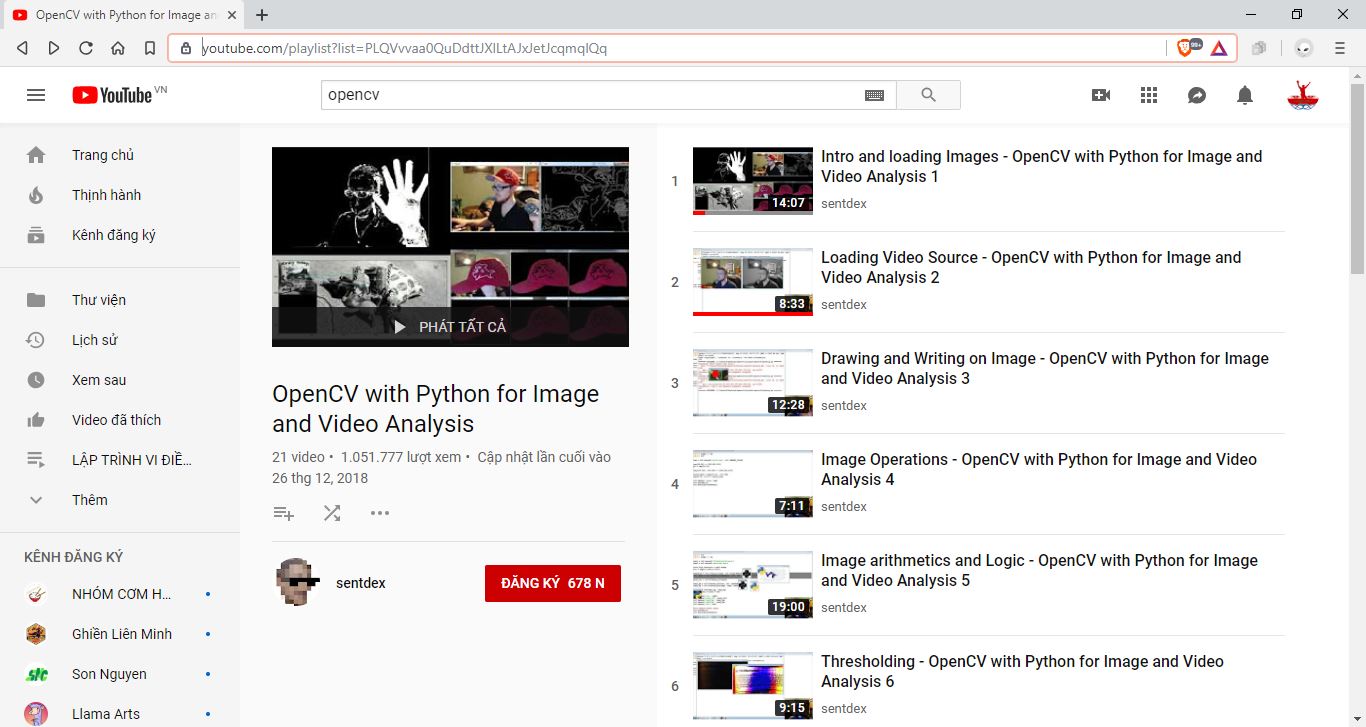
1. <https://www.learnopencv.com/>



1. <https://pythonprogramming.net/>

****

1. <https://www.youtube.com/watch?v=Z78zbnLlPUA&list=PLQVvvaa0QuDdttJXlLtAJxJetJcqmqlQq>

****