ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

———————— * ————————

ĐỀ CƯƠNG LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC

XE TỰ HÀNH KHÔNG NGƯỜI LÁI

HỘI ĐỒNG: Kỹ thuật máy tính
GVHD: ThS. Trần Thanh Bình
GVPB: ThS. Nguyễn Cao Trí
—o0o—
SVTH 1: Nguyễn Quốc Mạnh (1813043)
SVTH 2: Huỳnh Trương Quốc Khánh (1810990)

TP. HỒ CHÍ MINH, 12/2021

Lời cam đoan

Nhóm sinh viên thực hiện xin cam đoan rằng đề cương luận văn: "**XE TỰ HÀNH KHÔNG NGƯỜI LÁI**" là kết quả nghiên cứu do nhóm thực hiện dưới sự hướng dẫn của thầy ThS. Trần Thanh Bình.

Những tài liệu tham khảo, nội dung trích dẫn được trình bày chi tiết, cụ thể. Còn lại những nội dung khác chưa từng được công bố hoặc sử dụng để nhận bằng cấp ở những nơi khác.

Nếu phát hiện có bất kỳ sự gian lận nào, nhóm xin hoàn toàn chịu trách nhiệm về nội dung đề cương luận văn của mình. Trường đại học Bách Khoa TP. Hồ Chí Minh không liên quan đến những vi phạm (nếu có) về tác quyền, bản quyền do nhóm gây ra trong quá trình thực hiện.

TP.Hồ Chí Minh, ngày 06 tháng 12 năm 2021

Nhóm sinh viên thực hiện

Nguyễn Quốc Mạnh

Huỳnh Trương Quốc Khánh

Lời cảm ơn

Để hoàn thành đề cương luận văn này, nhóm sinh viên thực hiện xin gửi lời cảm ơn tới:

Quý thầy, cô trong Khoa Khoa học và Kỹ thuật máy tính, bộ môn Kỹ thuật máy tính, trường Đại học Bách Khoa TP. Hồ Chí Minh, những người đã hết lòng truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập tại trường. Đặc biệt, nhóm xin gửi lời cảm ơn chân thành và trân trọng nhất đến thầy **ThS. Trần Thanh Bình** đã tận tình hướng dẫn, giúp đỡ nhóm trong suốt quá trình nghiên cứu và thực hiện đề tài. Ngoài ra, nhóm xin được gửi lời cảm ơn chân thành đến thầy *TS. Nguyễn Trần Hữu Nguyên* - GVCN lớp Kỹ thuật máy tính khoá 2018 đã giúp đỡ, định hướng cho chúng em trong quá trình học tập, những kiến thức được thầy truyền đạt là hành trang quý báu cho chúng em trên con đường học tập, công tác và nghiên cứu sau này.

Xin gửi lời cảm ơn chân thành đến tất cả các bạn lớp Kỹ thuật máy tính khoá 2018 đã luôn giúp đỡ và hỗ trợ trong suốt quá trình học tập lý thuyết tại trường.

Xin chân thành cảm ơn!

Tóm tắt nội dung

Đề cương luận văn tập trung vào việc nghiên cứu và mô phỏng xe tự hành có khả năng di chuyển theo làn đường và biển báo giao thông sử dụng Hệ điều hành robot (ROS) và công cụ xử lý ảnh OpenCV. Hệ thống được đánh giá thử nghiệm trên công cụ mô phỏng Gazebo với robot Turtlebot3.

Mục lục

Danh sách hình vẽ

Chương 1

Giới thiệu

1.1 Mở đầu

Xe tự hành không người lái là mục tiêu phát triển trong tương lai của ngành giao thông vận tải. Một chiếc xe để có thể tự lái cần phải hiểu rõ môi trường xung quanh và tìm được đường đi với sự trợ giúp tối thiểu của con người. Giữ an toàn cho người lái và hành khách là yêu cầu quan trọng trong việc phát triển xe tự hành. Bám theo làn đường và hỗ trợ rẽ là một số hệ thống cho phép các phương tiện phát hiện làn đường và giúp giữ xe ở vị trí lý tưởng so với làn đường và hỗ trợ trong việc rẽ của xe.

1.2 Yêu cầu và mục tiêu của đề tài

1.2.1 Yêu cầu

- Nghiên cứu ROS (Robot Operating System).
- Nghiên cứu kỹ thuật xử lý hình ảnh bằng OpenCV và Machine learning.
- Làm quen với mô phỏng trên Gazebo.
- Xây dựng 1 hệ thống nhận diện hình ảnh và điểu khiển robot tự động di chuyển.
- Thử nghiệm hệ thống trên môi trường mô phỏng Gazebo tích hợp ROS sử dụng robot Turtlebot3.

1.2.2 Mục tiêu

Về kiến thức

- Nắm vững kiên thức về ROS, sử dụng ROS để điều khiển robot.
- Nắm vững các kiến thức lập trình python, ROS
- Nắm vững các kỹ thuật xử lý hình ảnh: nhân diện màu sắc, machine learning,...
- Phân tích, giải quyết yêu cầu bài toán nhận diện hình ảnh bằng OpenCV.

Về sản phẩm

- Robot có thể nhận diện được làn đường và các biển báo điều hướng từ đó di chuyển đúng làn đường được thiết kế sẵn.
- Nắm được quy trình phát triển sản phẩm: phân tích - thiết kế - hiện thực - kiểm tra.

1.3 Bố cục của luận văn

- **Chương 1: Giới thiệu**
  - Giới thiệu tổng quan về đề tài.
  - Trình bày yêu cầu, mục tiêu, đối tượng và phạm vi nghiên cứu.
- **Chương 2: Xây dựng mô hình và bộ điều khiển cho robot**
  - Giải thuật tìm làn đường
  - Xây dựng mô hình hỗ trợ nhận diện biển báo
  - Xây dựng bộ điều khiển cho robot
  - Xây dựng thế giới dùng trong mô phỏng
- **Chương 3: Nhận diện biển báo**
  - Xây dựng model CNN hỗ trợ nhận diện biển báo
- **Chương 4: Kết luận và hướng phát triển**
  - Đánh giá về ưu nhược điểm đề tài đã thực hiện.
  - Hướng hoàn thiện và phát triển đề tài.

1.4 Giới thiệu các công cụ

1.4.1 Robot Operating System (ROS)

Robot operating system là một hệ thống phần mềm chuyên dụng để lập trình và điều khiển robot, bao gồm các công cụ để lập trình, hiển thị, tương tác trực tiếp với phần cứng, và kết nối cộng đồng robot trên toàn thế giới.

Nó không chỉ cung cấp các dịch vụ hệ điều hành tiêu chuẩn (trừu tượng hóa phần cứng, quản lý tranh chấp, quản lý quy trình) mà còn cung cấp các chức năng cấp cao (cơ sở dữ liệu tập trung, hệ thống cấu hình robot, v.v.). Nó cũng cung cấp các công cụ và thư viện để xây dựng, viết và chạy mã nguồn trên nhiều máy tính.

Một số khái niệm trong ROS:

- **Package**: Package là đơn vị tổ chức phần mềm của mã ROS. Mỗi package có thể chứa thư viện, tệp thực thi, tập lệnh.
- **Node**: Một node là một process thực hiện tính toán. Các node được kết hợp với nhau thành một mạng lưới và giao tiếp với nhau bằng cách sử dụng các luồng topic, dịch vụ RPC và Máy chủ tham số (Parameter Server). Các node này hoạt động ở quy mô chi tiết. Một hệ thống điều khiển robot thường sẽ bao gồm nhiều node. Ví dụ: một node điều khiển công cụ laser xác định khoảng cách, một node điều khiển động cơ bánh xe của robot, một node thực hiện bản địa hóa (localization), một node thực hiện lập kế hoạch đường đi, một node cung cấp chế độ xem đồ họa của hệ thống.
- **Message**: Các node giao tiếp với nhau bằng cách publish các message đến các topic. Một message là một cấu trúc dữ liệu đơn giản, bao gồm các trường đã nhập. Các kiểu nguyên thủy tiêu chuẩn (số nguyên, dấu phẩy động, boolean,v.v) được hỗ trợ, cũng như các mảng của kiểu nguyên thủy. Message có thể bao gồm các cấu trúc và mảng được lồng nhau tùy ý (giống như cấu trúc C).
- **Topic**: Topic là các "tuyến đường" được đặt tên mà các nút trao đổi message. Các topic có cơ chế publish/subscribe ẩn danh, giúp tách rời việc truyền thông tin khỏi việc nhận nó.

Các node thường không nhận thức được chúng đang giao tiếp với ai. Thay vào đó, các node quan tâm đến dữ liệu cần thiết sẽ subscribe vào topic có liên quan; các node tạo dữ liệu sẽ publish đến topic có liên quan. Có thể có nhiều publisher và subscriber trên cùng một topic.

1.4.2 OpenCV cho Python

Project OpenCV được bắt đầu từ Intel năm 1999 bởi Gary Bradsky. OpenCV viết tắt cho Open Source Computer Vision Library. OpenCV là thư viện nguồn mở hàng đầu cho Computer Vision và Machine Learning, và hiện có thêm tính năng tăng tốc GPU cho các hoạt động theo real-time.

OpenCV có một cộng đồng người dùng khá hùng hậu hoạt động trên khắp thế giới bởi nhu cầu cần đến nó ngày càng tăng theo xu hướng chạy đua về sử dụng computer vision của các công ty công nghệ. OpenCV hiện được ứng dụng rộng rãi toàn cầu, với cộng đồng hơn 47.000 người, với nhiều mục đích và tính năng khác nhau từ interactive art, đến khai thác mỏ, khai thác web map hoặc qua robotic cao cấp.

Python là ngôn ngữ được dùng nhiều để demo/test OpenCV do tính ngắn gọn, ít phải thiết lập. Bên cạnh đó, nếu dùng Python thì cũng có thể code được trên nhiều hệ điều hành.

1.4.3 Gazebo

Gazebo là một công cụ mô phỏng robot 3D mã nguồn mở, Gazebo có thể sử dụng nhiều công cụ vật lý (physics engines) hiệu suất cao, chẳng hạn như ODE, Bullet, v.v. (mặc định là ODE). Nó cho phép kết xuất (render) môi trường rất chân thực với ánh sáng, bóng và kết cấu. Nó có thể mô hình hóa các cảm biến cảm nhận môi trường mô phỏng như laser xác định khoảng cách, máy ảnh (bao gồm cả góc rộng), cảm biến kiểu Kinect.

Chương 2

Xây dựng và mô phỏng hệ thống

2.1 Giải thuật phát hiện làn đường

2.1.1 Thresholding

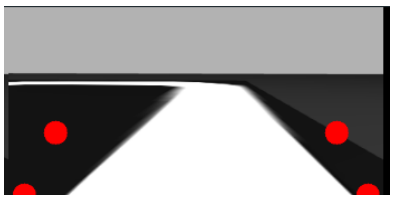
Kỹ thuật Thresholding (Phân ngưỡng ảnh) theo màu sắc được thực hiện để phát hiện màu sắc của làn đường. Kỹ thuật Thresholding không được áp dụng trực tiếp trên hình ảnh từ dữ liệu camera của robot mà phải qua một số chuyển đổi. Hình ảnh có định dạng BGR (Blue Green Red) được chuyển sang định dạng HSV (Hue Saturation Value) để phát hiện màu sắc dễ dàng hơn. Không gian màu HSV thể hiện cách màu sắc đáp ứng với ánh sáng. Biểu diễn dưới dạng toán học, HSV bao gồm ba ma trận cho "Hue", "Saturation" và "Value" với giá trị dao động trong khoảng lần lượt là 0-179, 0-255 và 0-255. Ứng dụng chính của HSV là phân vùng ảnh dựa trên màu sắc để phân tách làn đường ra khỏi môi trường xung quanh. Kỹ thuật Thresholding sẽ chuyển hình ảnh thành một ảnh nhị phân, trong đó màu sắc của làn đường được phát hiện sẽ có màu trắng và phần còn lại của hình ảnh sẽ có màu đen.

2.1.2 Warping

Hình ảnh có được từ dữ liệu camera của robot bị ảnh hưởng bởi góc nhìn. Vì vậy, ranh giới của làn đường không thể được nhận diện chính xác sử dụng hình ảnh này. Kỹ thuật Warping (Nắn ảnh) được sử dụng để điều chỉnh hình ảnh. Kỹ thuật Warping được áp dụng nhằm lấy được góc nhìn làn đường theo hướng từ trên xuống bằng cách thay đổi góc nhìn của hình ảnh. Để sử dụng Warping, một bộ bốn điểm được đặt lên hình ảnh. Giá trị của các điểm này sẽ quyết định Region of Interest (Vùng quan tâm - ROI). Việc này được thực hiện bằng cách truyền toạ độ các điểm vào các hàm của thư viện OpenCV. Kết quả nhận được sẽ là một hình ảnh được cắt theo Region of Interest với góc nhìn từ trên xuống. Warping là một trong những kỹ thuật tiền xử lý ảnh phổ biến nhất được sử dụng trong các hệ thống phát hiện làn đường vì nó cung cấp một hình ảnh mà các tính toán tiếp theo có thể được áp dụng lên một cách dễ dàng hơn so với hình ảnh nguyên bản lấy được từ dữ liệu camera. Ý tưởng cơ bản là có được một hình chữ nhật khi đoạn đường đang thẳng. [1]

2.1.3 Pixel Summation

Để tìm được độ cong của đường, việc cộng các pixel trên ảnh được thực hiện thông qua các bước:

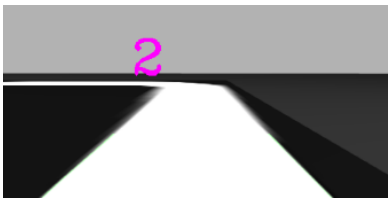


Hình 2.1: *Region of Interest*

Hình 2.2: *Ảnh qua bước Warping*

- Hình ảnh đã qua các bước Thresholding và Warping là một ảnh nhị phân, tức là ảnh chỉ có các pixel đen và trắng.
- Bước tiếp theo bao gồm định vị và lập bản đồ (mapping) làn đường trên hình ảnh đã xử lý bằng cách sử dụng biểu đồ tần suất (histogram). Trong lĩnh vực xử lí ảnh, biểu đồ tần suất được dùng để thống kê số lần xuất hiện các mức sáng trong ảnh.
- Trong trường hợp này, hình ảnh được phân ngưỡng dưới dạng nhị phân nên biểu đồ tần suất sẽ biểu diễn các điểm đen (điểm không) và trắng (điểm khác không). Các pixel khác không thể hiện vị trí của làn đường, vị trí của pixel bắt đầu từ góc dưới trái của ảnh theo trục x thể hiện vị trí bắt đầu của làn đường.
- Có khả năng một số pixel trong hình ảnh bị ảnh hưởng bởi nhiễu. Để tránh được chúng trong quá trình tính toán, một giá trị ngưỡng được thiết lập để quyết định một cột bất kỳ có phải là một phần của làn đường hay là nhiễu.
- Giá trị trung bình của biểu đồ tần suất của một phần tư dưới cùng của hình ảnh sẽ được xem là điểm gốc (base point) hay trung tâm của làn đường. Giá trị trung bình của biểu đồ tần suất của cả hình ảnh sẽ được xem là điểm giữa (middle point). Vậy lấy điểm giữa trừ đi điểm gốc sẽ cho ta một giá trị độ cong có thể được hiệu chỉnh để sử dụng cho các tính toán tiếp theo như tìm ra góc rẽ cho xe/robot.[2]

Hình 2.3: *Ảnh qua bước mapping*



Hình 2.4: *Giá trị độ cong*

2.2 Xây dựng bộ điều khiển cho robot

2.2.1 Lấy dữ liệu từ camera

Subscribe vào topic */camera/image* để lấy dữ liệu trực tiếp từ camera của robot

```
class Test_Img:
    def __init__(self):
        self.cv_bridge = CvBridge()
        self.image_sub = rospy.Subscriber("/camera/image",Image
    ,self.callback)
```

Mỗi khi nhận dữ liệu từ camera, gọi hàm *callback* để xử lý ảnh nhận được, cần chuyển ảnh từ định dạng message của ROS thành hình ảnh có thể xử lý với OpenCV và thay đổi kích thước thành 480x240 pixel trước khi tiến hành xử lý để tìm được giá trị độ cong của làn đường

```
def callback(self,data):
    try:
        cv_image = self.cv_bridge.imgmsg_to_cv2(data,"bgr8")
    except CvBridgeError as e:
        print(e)
    cv_image = cv2.resize(cv_image,(480,240))
```

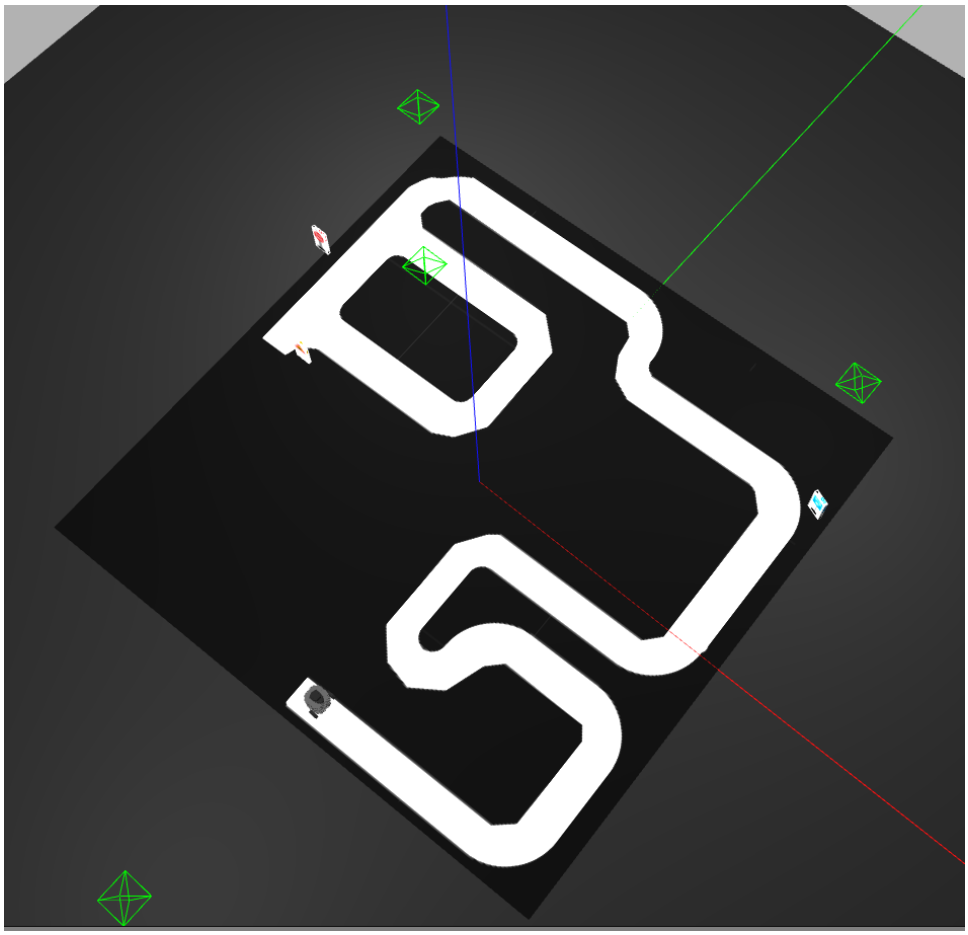
2.2.2 Di chuyển theo độ cong của làn đường

Xây dựng hàm điều khiển việc di chuyển của robot theo giá trị độ cong của làn đường. Góc lệch khi di chuyển của robot cho phù hợp với giá trị độ cong của làn đường được tìm bằng cách thử nghiệm các giá trị góc lệch khác nhau cho đến khi tối ưu.

```
def move(self,curveVal):
        cmd_vel = Twist()
        cmd_vel.linear.x = 0.1
        cmd_vel.angular.z = 0.0

        NegativeCurve = False
        if curveVal>0: NegativeCurve = True

        if curveVal<0.2 and curveVal>-0.2: curveVal=0.0
        elif curveVal<0.4 and curveVal>-0.4: curveVal=0.123
        elif curveVal<0.5 and curveVal>-0.5: curveVal=0.123
        elif curveVal<0.6 and curveVal>-0.6: curveVal=0.123
        elif curveVal<0.7 and curveVal>-0.7: curveVal=0.263
        elif curveVal<0.8 and curveVal>-0.8: curveVal=0.263
        elif curveVal<0.9 and curveVal>-0.9: curveVal=0.263
        elif curveVal<1.0 and curveVal>-1.0: curveVal=0.263
        elif curveVal<1.2 and curveVal>-1.2: curveVal=0.525
        elif curveVal<1.5 and curveVal>-1.5: curveVal=0.613
        else: curveVal=0.349

        if NegativeCurve == True:
            cmd_vel.angular.z = -curveVal
        else:
            cmd_vel.angular.z = curveVal
        self.cmd_vel_pub.publish(cmd_vel)
```

2.3 Mô phỏng

Thế giới dùng trong mô phỏng với làn đường và các biển báo



Hình 2.5: *Thế giới dùng trong mô phỏng*

Chương 3

Nhận diện biển báo

3.1 Xây dựng model CNN hỗ trợ nhận diện biển báo

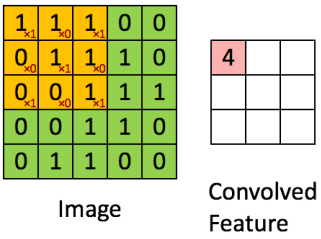
3.1.1 CNN model

CNN là gì ?

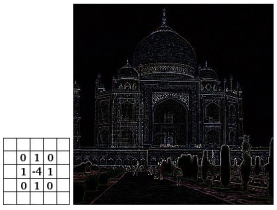
CNN - Convolutional Neural Network là một trong những mô hình Deep Learning tiên tiến. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay. CNN được sử dụng nhiều trong các bài toán nhận dạng các object trong ảnh.

Convolutional là gì?

Convolutional là một cửa sổ trượt (Sliding Windows) trên một ma trận.



Các convolutional layer có các parameter(kernel) đã được học để tự điều chỉnh lấy ra những thông tin chính xác nhất mà không cần chọn các feature.



Trong hình ảnh ví dụ trên, ma trận bên trái là một hình ảnh trắng đen được số hóa. Ma trận có kích thước 5×5 và mỗi điểm ảnh có giá trị 1 hoặc 0 là giao điểm của dòng và cột.

Convolution hay tích chập là nhân từng phần tử trong ma trận 3. Sliding Window hay còn gọi là kernel, filter hoặc feature detect là một ma trận có kích thước nhỏ như trong ví dụ trên là 3×3.

Convolution hay tích chập là nhân từng phần tử bên trong ma trận 3×3 với ma trận bên trái. Kết quả được một ma trận gọi là Convoled feature được sinh ra từ việc nhận ma trận Filter với ma trận ảnh 5×5 bên trái.

Cấu trúc mạng của CNN

Mạng CNN là một tập hợp các lớp Convolution chồng lên nhau và sử dụng các hàm nonlinear activation như ReLU và tanh để kích hoạt các trọng số trong các node. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo.

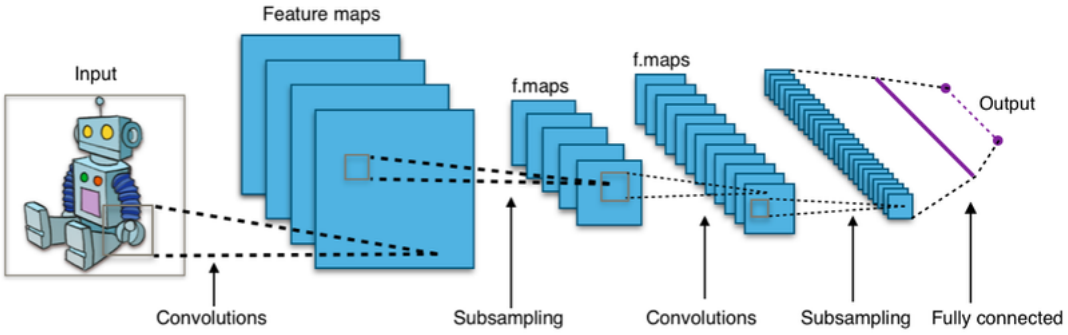
Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo. Trong mô hình mạng truyền ngược (feedforward neural network) thì mỗi neural đầu vào (input node) cho mỗi neural đầu ra trong các lớp tiếp theo.

Mô hình này gọi là mạng kết nối đầy đủ (fully connected layer) hay mạng toàn vẹn (affine layer). Còn trong mô hình CNNs thì ngược lại. Các layer liên kết được với nhau thông qua cơ chế convolution.

Layer tiếp theo là kết quả convolution từ layer trước đó, nhờ vậy mà ta có được các kết nối cục bộ. Như vậy mỗi neuron ở lớp kế tiếp sinh ra từ kết quả của filter áp đặt lên một vùng ảnh cục bộ của neuron trước đó.

Mỗi một lớp được sử dụng các filter khác nhau thông thường có hàng trăm hàng nghìn filter như vậy và kết hợp kết quả của chúng lại. Ngoài ra có một số layer khác như pooling/subsampling layer dùng để chắt lọc lại các thông tin hữu ích hơn (loại bỏ các thông tin nhiễu).

Trong quá trình huấn luyện mạng (traning) CNN tự động học các giá trị qua các lớp filter dựa vào cách thức mà bạn thực hiện. Ví dụ trong tác vụ phân lớp ảnh, CNNs sẽ cố gắng tìm ra thông số tối ưu cho các filter tương ứng theo thứ tự raw pixel > edges > shapes > facial > high-level features. Layer cuối cùng được dùng để phân lớp ảnh.



Trong mô hình CNN có 2 khía cạnh cần quan tâm là tính bất biến (Location Invariance) và tính kết hợp (Compositionality). Với cùng một đối tượng, nếu đối tượng này được chiếu theo các gốc độ khác nhau (translation, rotation, scaling) thì độ chính xác của thuật toán sẽ bị ảnh hưởng đáng kể.

Pooling layer sẽ cho bạn tính bất biến đối với phép dịch chuyển (translation), phép quay (rotation) và phép co giãn (scaling). Tính kết hợp cục bộ cho ta các cấp độ biểu diễn thông tin từ mức độ thấp đến mức độ cao và trừu tượng hơn thông qua convolution từ các filter.

Đó là lý do tại sao CNNs cho ra mô hình với độ chính xác rất cao. Cũng giống như cách con người nhận biết các vật thể trong tự nhiên.

3.1.2 Keras

Keras là một open source cho Neural Network được viết bởi ngôn ngữ Python. Nó là một library được phát triển vào năm 205 bởi Francois Chollet, là một kỹ sư nghiên cứu Deep Learning. Keras có thể sử dụng chung với các thư viện nổi tiếng như Tensorflow, CNTK, Theano. Một số ưu điểm của Keras như:

- Dễ sử dụng, dùng đơn giản hơn Tensor, xây dựng model nhanh.
- Run được trên cả CPU và GPU.
- Hỗ trợ xây dựng CNN , RNN hoặc cả hai. Với những người mới tiếp cận đến Deep như mình thì mình chọn sử dụng Keras để build model vì nó đơn giản,dễ nắm bắt hơn các thư viện khác.

3.1.3 Xây dựng model CNN

[3] Trong Keras có hỗ trợ 2 cách dựng models là Sequential model và Function API. Ở đây chung ta sẽ sử dụng Sequential model.

Trong dự án này chung ta sẽ xây dựng model CNN hỗ trợ cho việc nhận diện 4 loại biển báo

Công cụ sử dụng

Import library

```
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
import cv2
from sklearn.model_selection import train_test_split
import pickle
import os
import pandas as pd
import random
from keras.preprocessing.image import ImageDataGenerator
```

Khai báo biến

```
path = 'myData'  # folder with all the class folders
labelFile = 'labels.csv' #file with all class folders
batch_size_val= 50 # file with all name of classes
```

```
steps_per_epoch_val=2000
epochs_val=10
imageDimesions = (32,32,3)
testRatio = 0.2 #if 1000 image 20% of remaining 800 will be 200 for validation
validationRatio = 0.2
```

```
count = 0
images = []
classNo = []
myList = os.listdir(path)
print("Total Classes Detected:",len(myList))
noOfClasses=len(myList)
print("Importing Classes.....")
for x in range (0,len(myList)):
    myPicList = os.listdir(path+"/"+str(count))
    for y in myPicList:
        curImg = cv2.imread(path+"/"+str(count)+"/"+y)
        images.append(curImg)
        classNo.append(count)
    print(count,end =" ")
    count +=1
print(" ")
images = np.array(images)
classNo = np.array(classNo)
```

```
Total Classes Detected: 4
Importing Classes.....
0 1 2 3
```

```
X_train, X_test, y_train, y_test = train_test_split(images, classNo,test_size=testRatio)
X_train, X_validation, y_train, y_validation = train_test_split(X_train,y_train,test_size=validationRatio)
```

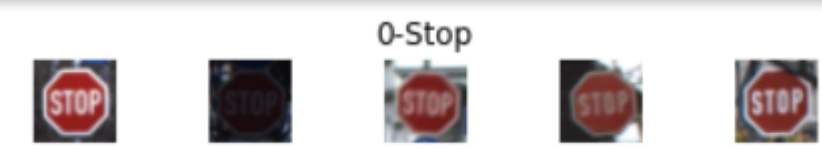
```
print("Data Shapes")
print("Train",end = "");print(X_train.shape,y_train.shape)
print("Validation",end = "");print(X_validation.shape, y_validation.shape)
print("Test",end="");print(X_test.shape,y_test.shape)
assert (X_train.shape[0]==y_train.shape[0]),"The number of image in not equal to the number of label in training set"
assert(X_validation.shape[0]==y_validation.shape[0]), "The number of images in not equal to the number of lables in validation set"
```

```
assert(X_test.shape[0]==y_test.shape[0]), "The number of images
    in not equal to the number of lables in test set"
assert(X_train.shape[1:]==(imageDimesions)),"  The dimesions of
   the Training images are wrong "
assert(X_validation.shape[1:]==(imageDimesions)),"  The
   dimesionas of the Validation images are wrong "
assert(X_test.shape[1:]==(imageDimesions)),"  The dimesionas of
   the Test images are wrong"
```

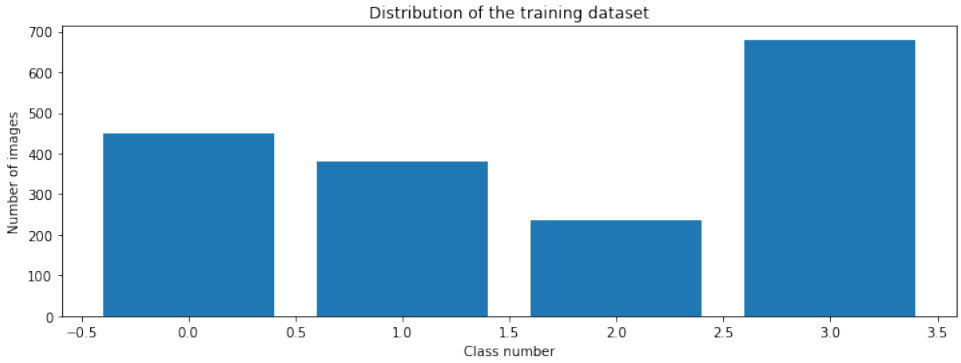
```
Data Shapes
Train(1746, 32, 32, 3) (1746,)
Validation(437, 32, 32, 3) (437,)
Test(546, 32, 32, 3) (546,)
```

```
data=pd.read_csv(labelFile)
print("data shape ",data.shape,type(data))
```

```
num_of_samples = []
cols = 5
num_classes = noOfClasses
fig, axs = plt.subplots(nrows=num_classes,ncols=cols,figsize
   =(5,300))
fig.tight_layout()
for i in range(cols):
    for j,row in data.iterrows():
        x_selected = X_train[y_train == j]
        axs[j][i].imshow(x_selected[random.randint(0,len(
   x_selected)-1),:,:],cmap=plt.get_cmap("gray"))
        axs [j][i].axis("off")
        if i == 2:
            axs [j][i].set_title(str(j)+ "-"+row["Name"])
            num_of_samples.append(len(x_selected))
```




```
print(num_of_samples)
plt.figure(figsize=(12, 4))
plt.bar(range(0, num_classes), num_of_samples)
plt.title("Distribution of the training dataset")
plt.xlabel("Class number")
plt.ylabel("Number of images")
plt.show()
```




```
def grayscale(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img


def equalize(img):
    img = cv2.equalizeHist(img)
    return img
```

```

def preprocessing(img):
    img = grayscale(img)  # CONVERT TO GRAYSCALE
    img = equalize(img)  # STANDARDIZE THE LIGHTING IN AN IMAGE
    img = img / 255  # TO NORMALIZE VALUES BETWEEN 0 AND 1 INSTEAD OF 0 TO 255
    return img


X_train = np.array(list(map(preprocessing, X_train)))  # TO IRETATE AND PREPROCESS ALL IMAGES
X_validation = np.array(list(map(preprocessing, X_validation)))
X_test = np.array(list(map(preprocessing, X_test)))
cv2.imshow("GrayScale Images",X_train[random.randint(0,len(X_train)-1)]) # TO CHECK IF THE TRAINING IS DONE PROPERLY
```

```
X_train = X_train.reshape(X_train.shape[0],X_train.shape[1],X_train.shape[2],1)
X_validation = X_validation.reshape(X_validation.shape [0],X_validation.shape[1],X_validation.shape[2],1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1],X_test.shape[2],1)
```

```
dataGen = ImageDataGenerator(width_shift_range=0.1,# 0.1 = 10% IF MORE THAN 1 E.G 10 THEN IT REFFERS TO NO. OF  PIXELS EG 10 PIXELS
                             height_shift_range=0.1,
                             zoom_range=0.2,  # 0.2 MEANS CAN GO FROM 0.8 TO 1.2
                             shear_range=0.1,  # MAGNITUDE OF SHEAR ANGLE
                             rotation_range=10)  # DEGREES
dataGen.fit(X_train)
batches = dataGen.flow(X_train, y_train,batch_size=20)  # REQUESTING DATA GENRATOR TO GENERATE IMAGES  BATCH SIZE = NO. OF IMAGES CREAED EACH TIME ITS CALLED
X_batch, y_batch = next(batches)
```

```
fig, axs = plt.subplots(1, 15, figsize=(20, 5))
fig.tight_layout()

for i in range(15):
    axs[i].imshow(X_batch[i].reshape(imageDimesions[0],imageDimesions[1]))
    axs[i].axis('off')
    plt.show()

```

```
y_train = to_categorical(y_train, noOfClasses)
y_validation = to_categorical(y_validation, noOfClasses)
y_test = to_categorical(y_test, noOfClasses)
```

```
def myModel():
    no_Of_Filters = 60
    size_of_Filter = (5, 5)  # THIS IS THE KERNEL THAT MOVE AROUND THE IMAGE TO GET THE FEATURES.
    # THIS WOULD REMOVE 2 PIXELS FROM EACH BORDER WHEN USING 32 32 IMAGE
    size_of_Filter2 = (3, 3)
    size_of_pool = (2, 2)  # SCALE DOWN ALL FEATURE MAP TO GERNALIZE MORE, TO REDUCE OVERFITTING
    no_Of_Nodes = 500  # NO. OF NODES IN HIDDEN LAYERS
    model = Sequential()
    model.add((Conv2D(no_Of_Filters, size_of_Filter, input_shape=(imageDimesions[0],imageDimesions[1],1), activation='relu')))  # ADDING MORE CONVOLUTION LAYERS = LESS FEATURES BUT CAN CAUSE ACCURACY TO INCREASE
    model.add((Conv2D(no_Of_Filters, size_of_Filter, activation='relu')))
    model.add(MaxPooling2D(pool_size=size_of_pool))  # DOES NOT EFFECT THE DEPTH/NO OF FILTERS

    model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu')))
    model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu')))
    model.add(MaxPooling2D(pool_size=size_of_pool))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(no_Of_Nodes, activation='relu'))
    model.add(Dropout(0.5))  # INPUTS NODES TO DROP WITH EACH UPDATE 1 ALL 0 NONE
    model.add(Dense(noOfClasses, activation='softmax'))  # OUTPUT LAYER
    # COMPILE MODEL
    model.compile(Adam(lr=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
model = myModel()
print(model.summary())
history = model.fit_generator(dataGen.flow(X_train, y_train, batch_size=batch_size_val),steps_per_epoch=steps_per_epoch_val, epochs=epochs_val,validation_data=(X_validation, y_validation), shuffle=1)
```

```
################################ PLOT
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 28, 28, 60)        1560
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 24, 24, 60)        90060
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 60)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 10, 10, 30)        16230
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 30)          8130
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 30)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 30)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 480)               0
_________________________________________________________________
dense_3 (Dense)              (None, 500)               240500
_________________________________________________________________
dropout_4 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 2004
=================================================================
Total params: 358,484
Trainable params: 358,484
Non-trainable params: 0
```

```
None
Epoch 1/10
2000/2000 [==============================] - 395s 198ms/step - loss: 0.0778 - accuracy: 0.9717 - val_loss: 2.7207e-05 - val_acc
uracy: 1.0000
Epoch 2/10
2000/2000 [==============================] - 381s 190ms/step - loss: 0.0100 - accuracy: 0.9970 - val_loss: 9.0834e-06 - val_acc
uracy: 1.0000
Epoch 3/10
2000/2000 [==============================] - 372s 186ms/step - loss: 0.0067 - accuracy: 0.9980 - val_loss: 0.0021 - val_accurac
y: 1.0000
Epoch 4/10
2000/2000 [==============================] - 373s 187ms/step - loss: 0.0058 - accuracy: 0.9984 - val_loss: 2.9897e-07 - val_acc
uracy: 1.0000
Epoch 5/10
2000/2000 [==============================] - 372s 186ms/step - loss: 0.0051 - accuracy: 0.9986 - val_loss: 1.3367e-08 - val_acc
uracy: 1.0000
Epoch 6/10
2000/2000 [==============================] - 379s 190ms/step - loss: 0.0044 - accuracy: 0.9988 - val_loss: 2.7279e-10 - val_acc
uracy: 1.0000
Epoch 7/10
2000/2000 [==============================] - 374s 187ms/step - loss: 0.0044 - accuracy: 0.9988 - val_loss: 0.0000e+00 - val_acc
uracy: 1.0000
Epoch 8/10
2000/2000 [==============================] - 371s 186ms/step - loss: 0.0055 - accuracy: 0.9987 - val_loss: 0.0000e+00 - val_acc
uracy: 1.0000
Epoch 9/10
2000/2000 [==============================] - 376s 188ms/step - loss: 0.0040 - accuracy: 0.9992 - val_loss: 0.0000e+00 - val_acc
uracy: 1.0000
Epoch 10/10
2000/2000 [==============================] - 380s 190ms/step - loss: 0.0050 - accuracy: 0.9988 - val_loss: 2.7279e-10 - val_acc
uracy: 1.0000
```

```
plt.figure(1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training','validation'])
plt.title('loss')
plt.xlabel('epoch')
plt.figure(2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training','validation'])
plt.title('Acurracy')
plt.xlabel('epoch')
plt.show()
score = model.evaluate(X_test, y_test, verbose=0)
print('Test Score:', score[0])
print('Test Accuracy:', score[1])
```

Lưu model dưới dạng file.

```
model.save("my_model")
model.save_weights("weights.h5")
```

3.1.4 Nhận diện bằng openCV

```
import cv2
import numpy as np
from scipy.stats import itemfreq
import pickle
```

Load model.

```
##################################################33

# ######################################3
font = cv2.FONT_HERSHEY_SIMPLEX
from tensorflow import keras
my_model = keras.models.load_model("my_model")
my_model.load_weights("weights.h5")
```

Chuyển ảnh về dạng COLOR_BGRGRAY để tăng khả năng xử lý

```
def grayscale(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img
```

Cân bằng sáng. Tham khảo giải thuật cân bằng sáng của opencv. // link

```
def equalize(img):
    img = cv2.equalizeHist(img)
    return img
```

Chuyển các giá trị pixel của ảnh nằm trong khoảng từ 0 đến 1 thay vì 0 đến 255.

Đối với hầu hết dữ liệu hình ảnh, giá trị pixel là số nguyên có giá trị từ 0 đến 255.

Mạng nơron xử lý đầu vào bằng cách sử dụng các giá trị trọng số nhỏ và đầu vào có giá trị số nguyên lớn có thể làm gián đoạn hoặc làm chậm quá trình học. Do đó, bạn nên chuẩn hóa các giá trị pixel để mỗi giá trị pixel có giá trị từ 0 đến 1.

Nó hợp lệ cho hình ảnh có giá trị pixel trong phạm vi 0-1 và hình ảnh có thể được xem bình thường.

```
def preprocessing(img):
    img = grayscale(img)
    img = equalize(img)
    img = img/255
    return img
```

Tạo hệ thống class name

```
def getClassName(classNo):
    if classNo == 0: return 'Stop'
    elif classNo == 1: return 'Right'
    elif classNo == 2: return 'Left'
    elif classNo == 3: return 'ahead only'

##################################33
```

```
clicked = False
def onMouse(event, x, y, flags, param):
    global clicked
    if event == cv2.EVENT_LBUTTONUP:
        clicked = True
```

Lấy ảnh từ camera và xử lý.

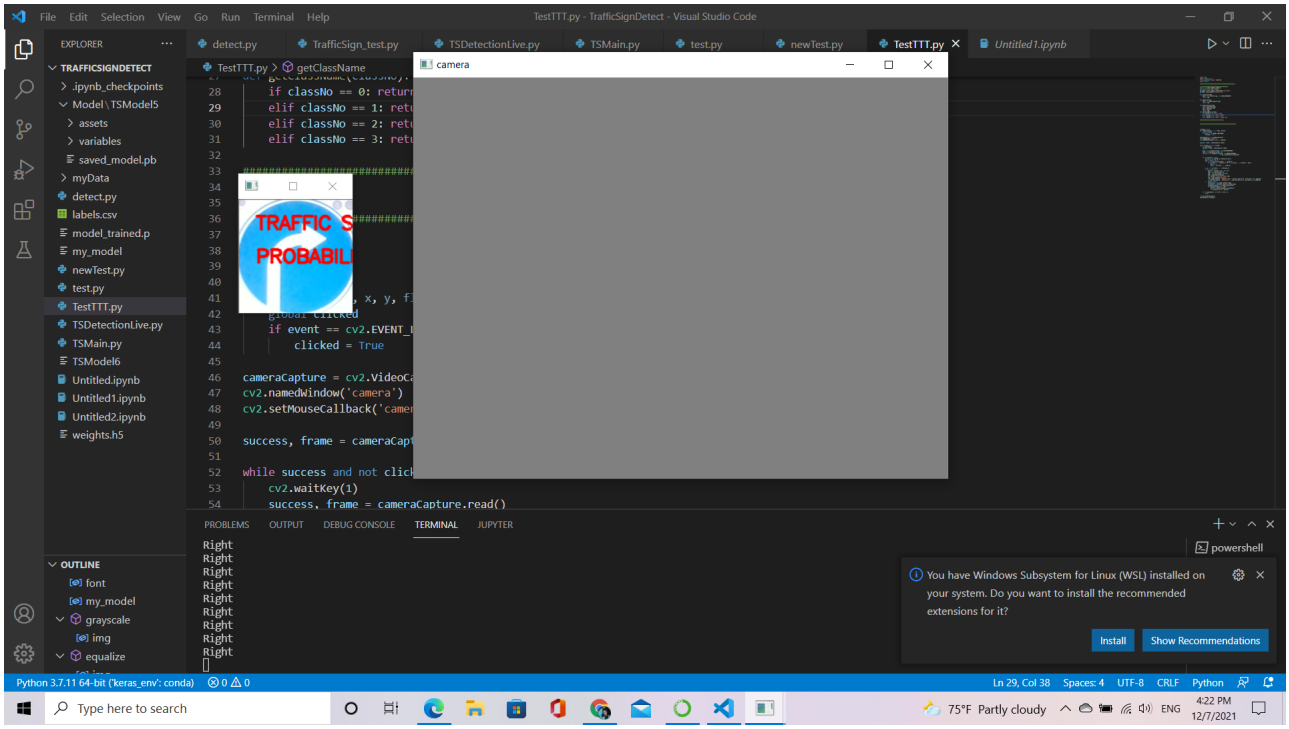
Để tăng độ chính xác chung ta có sử dụng thêm 1 số chức năng của opencv: Sử dụng cv2.HoughCircles để lọc hình tròn trong ảnh. Và chỉ lọc lấy hình tròn lớn nhất.

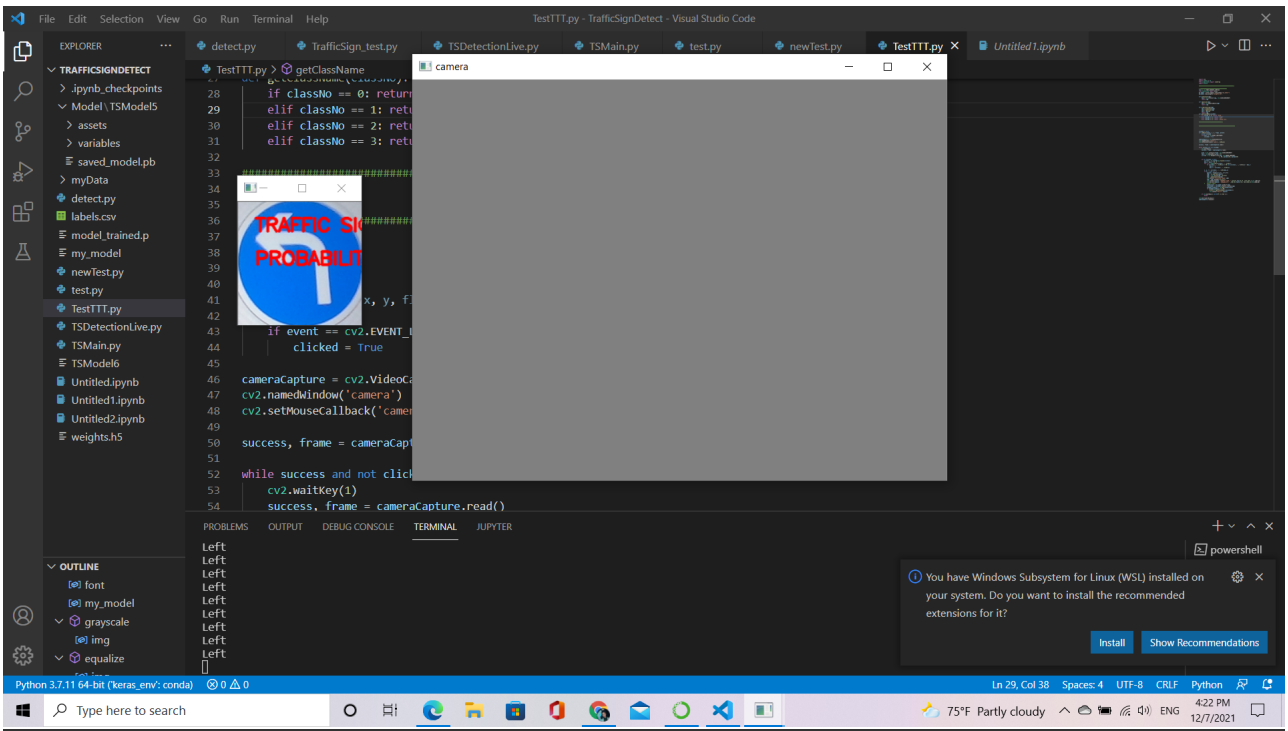
Rồi sử dụng model.predict(image) để nhận diện.

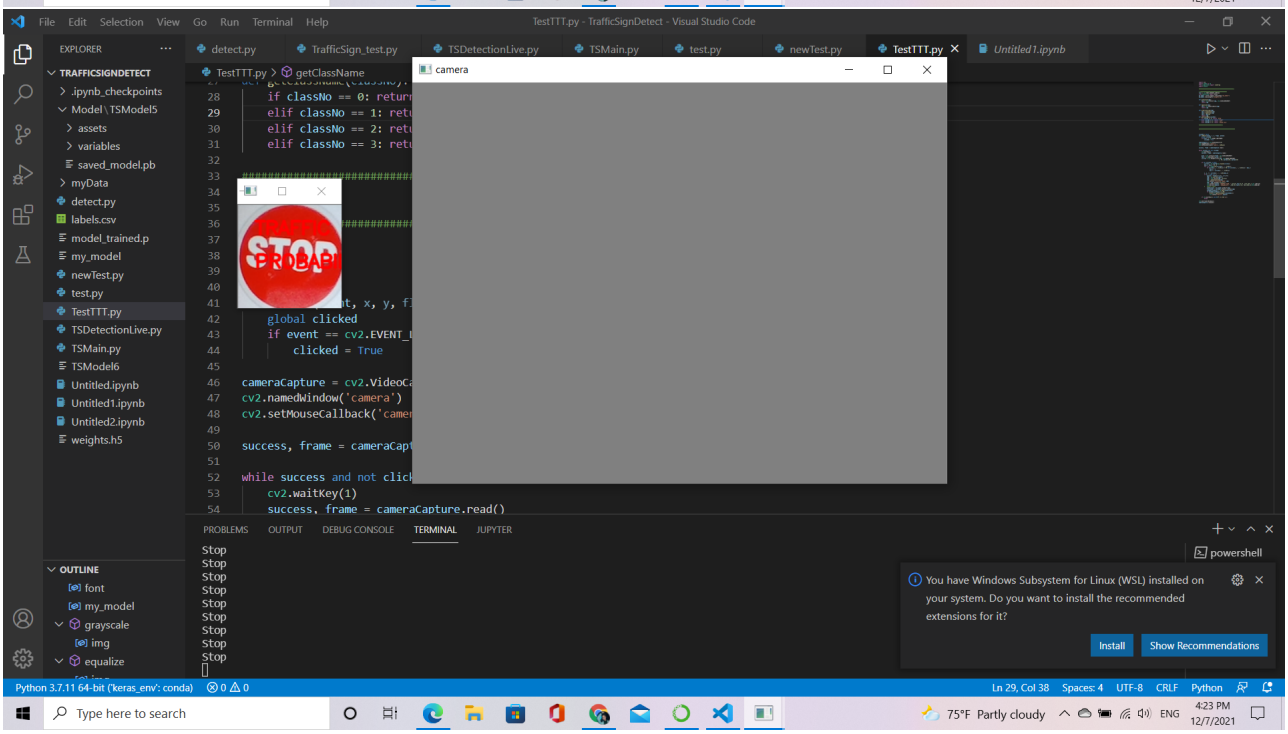
```
cameraCapture = cv2.VideoCapture(0)
cv2.namedWindow('camera')
cv2.setMouseCallback('camera', onMouse)

success, frame = cameraCapture.read()

while success and not clicked:
    cv2.waitKey(1)
    success, frame = cameraCapture.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    img = cv2.medianBlur(gray, 37)
    circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT,
                               1, 50, param1=120, param2=40)

    if not circles is None:
        circles = np.uint16(np.around(circles))
        max_r, max_i = 0, 0
        for i in range(len(circles[:, :, 2][0])):
            if circles[:, :, 2][0][i] > 50 and circles[:, :,
   2][0][i] > max_r:
                max_i = i
                max_r = circles[:, :, 2][0][i]

        x, y, r = circles[:, :, :][0][max_i]
        if y > r and x > r:
            square = frame[y-r:y+r, x-r:x+r]
            img = np.asarray(square)
            img = cv2.resize(img, (32,32))
            img = preprocessing(img)
            cv2.imshow("Processed Image",img)
            img = img.reshape(1,32,32,1)
            cv2.putText(square, "TRAFFIC SIGN: ",(20,35),font
   ,0.75, (0,0,255),2,cv2.LINE_AA)
            cv2.putText(square, "PROBABILITY: ",(20,75),font
   ,0.75, (0,0,255),2,cv2.LINE_AA)
        #   PREDICT IMAGE
            predictions = my_model.predict(img)
```

```
            classIndex = my_model.predict_classes(img)
            probabilityValue = np.amax(predictions)
            if probabilityValue > 0.99:
                print(str(getClassName(classIndex)))
                cv2.imshow("Result",square)

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

cv2.destroyAllWindows()
cameraCapture.release()
```

Demo trên webcam laptop.




TestTTT.py - TrafficSignDetect - Visual Studio Code
camera
TRAFFIC SI
PROBABILIT
Left
Left
Left
Left
Left
Left
Left
Left
Left
You have Windows Subsystem for Linux (WSL) installed on your system. Do you want to install the recommended extensions for it?
Install
Show Recommendations
Python 3.7.11 64-bit ('keras_env': conda)
Ln 29, Col 38
Spaces: 4
UTF-8
CRLF
Python
Type here to search
75°F Partly cloudy
4:22 PM
12/7/2021


TestTTT.py - TrafficSignDetect - Visual Studio Code
camera
STOP
PROBABI
Stop
Stop
Stop
Stop
Stop
Stop
Stop
Stop
Stop
You have Windows Subsystem for Linux (WSL) installed on your system. Do you want to install the recommended extensions for it?
Install
Show Recommendations
Python 3.7.11 64-bit ('keras_env': conda)
Ln 29, Col 38
Spaces: 4
UTF-8
CRLF
Python
Type here to search
75°F Partly cloudy
4:23 PM
12/7/2021

Chương 4

Kết luận và hướng phát triển

4.1 Đánh giá kết quả thực hiện

- Đã tìm hiểu, phân tích và xây dựng mô hình của robot; đánh giá được kết quả xây dựng thông qua mô phỏng trên Gazebo.
- Cơ bản hiểu lý thuyết và ứng dụng được thư viện xử lý ảnh OpenCV cho Python.

4.2 Những mặt hạn chế

- Robot di chuyển chưa thật sự tối ưu ở các góc rẽ
- Việc tích hợp phần nhận diện biển báo vào bộ điều khiển robot còn gặp nhiều khó khăn

4.3 Hướng phát triển

- Tối ưu hóa việc di chuyển của robot
- Hoàn thiện phần nhận diện biển báo và đưa ra quyết định hướng di chuyển của robot
- Kiểm tra kết quả mô phỏng trên robot thật, đáp ứng yêu cầu đề ra của đề tàifb

Tài liệu tham khảo

[1] Vighnesh Devane, Ganesh Sahane, Hritish Khairmode, and Gaurav Datkhile. Lane detection techniques using image processing. In *ITM Web of Conferences*, volume 40, pages 1–4, Jan. 2021.

[2] M.Hassan. Self-driving car using raspberry pi. In *computer-vision.zone*, 2020.

[3] M.Hassan. Traffic signs classification using convolution neural networks cnn. In *computer-vision.zone*, February 2020.