

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



## ĐỒ ÁN THIẾT KẾ LUẬN LÝ (CO3091)

Báo cáo đồ án môn học

# HỆ THỐNG HỖ TRỢ ĐIỀU HƯỚNG XE TỰ HÀNH TRONG MÔI TRƯỜNG MÔ PHỎNG

GVHD: Thầy Phạm Hoàng Anh  
SV: Nguyễn Trọng Nhân - 1914446  
Lê Hoàng Minh Tú - 1915812  
Hồ Hữu Trọng - 1915672

Tp. Hồ Chí Minh, Tháng 12/2021



## Mục lục

<b>1 Lời nói đầu tiên:</b>	<b>2</b>
1.1 Mục tiêu: . . . . .	2
1.2 Thực trạng: . . . . .	2
1.3 Các tính năng: . . . . .	2
<b>2 Nội dung</b>	<b>3</b>
2.1 Tìm hiểu về hệ điều hành ROS . . . . .	3
2.1.1 ROS là gì . . . . .	3
2.1.2 Cài đặt ROS: . . . . .	3
2.2 Mô phỏng trên Gazebo . . . . .	3
2.2.1 Gazebo là gì? . . . . .	3
2.2.2 Cài đặt Gazebo: . . . . .	4
2.2.3 Không gian làm việc Gazebo: . . . . .	4
2.2.4 Các model và các map đã build: . . . . .	5
2.3 Phát hiện biển báo và phân biệt biển báo . . . . .	7
2.3.1 Phát hiện biển báo (sign detection): . . . . .	7
2.3.2 Phân biệt biển báo (sign classification) . . . . .	13
2.3.3 Áp dụng model vừa train vào hệ thống . . . . .	16
2.4 Nhận diện làn đường . . . . .	18
2.4.1 Tiền xử lí (Image Preprocessing) . . . . .	19
2.4.2 Ứng dụng kỹ thuật Canny Edge Detection . . . . .	20
2.4.3 Tạo vùng chú ý (Region-of-interest) . . . . .	22
2.4.4 Ứng dụng biến đổi Line Hough (Line Hough Transform) . . . . .	23
2.4.5 Phác họa các dữ liệu đã tính toán và cân chỉnh . . . . .	25
2.5 Tổng hợp hệ thống . . . . .	26
2.5.1 Một số định nghĩa quan trọng . . . . .	26
2.5.2 Sơ đồ giao tiếp giữa các Node/Topic và chức năng . . . . .	27
2.5.3 Sơ đồ xử lí của khối tổng hợp (Controller) . . . . .	28
<b>3 Tổng kết</b>	<b>29</b>
3.1 Video demo project của nhóm . . . . .	29
3.2 Thuận lợi và khó khăn . . . . .	29
3.2.1 Thuận lợi . . . . .	29
3.2.2 Khó khăn . . . . .	29
<b>4 Phân công công việc</b>	<b>31</b>
4.1 Bảng phân công công việc . . . . .	31
4.1.1 Link Trello quản lý công việc . . . . .	31
4.1.2 Link Github Project: . . . . .	31
<b>5 Tài liệu tham khảo</b>	<b>32</b>



## 1 Lời nói đầu tiên:

Khoa học Công nghệ ngày càng phát triển và được áp dụng vào từng ngóc ngách trong đời sống chúng ta. Việc này là không thể bàn cãi, và nhiệm vụ của người nghiên cứu công nghệ là có thể áp dụng những nghiên cứu của mình vào đời sống thực tế. Bởi học thuyết gắn với thực tiễn thì mới hữu ích.

### 1.1 Mục tiêu:

Để góp phần, đóng góp những công nghệ giúp ích cho đời sống của mọi người. Nhóm đã quyết định nghiên cứu công nghệ xe tự hành với mục tiêu giúp người dân có thể nghỉ ngơi trong quá trình di chuyển giữa hai địa điểm. Đồng thời hướng tới một mục tiêu xa hơn là đô thị siêu thông minh (nơi mà tất cả các phương tiện đều là xe tự hành) khi đó sẽ giảm tỉ lệ tai nạn giao thông xuống thấp nhất.

### 1.2 Thực trạng:

Để đánh giá tổng quan về đề tài này, đây là một đề tài khó. Bởi 3 yếu tố:

1. Để áp dụng trên thực tế, có rất rất nhiều yếu tố ảnh hưởng đến hành vi của xe (ánh sáng, độ trơn của mặt đường, hành vi của các phương tiện khác, ...)
2. Data dùng để train model là nguồn data mở trên mạng, nên về số lượng không được nhiều. Về chất lượng: chỉ phù hợp với biển báo của data đó (khó để áp dụng vào biển báo của Việt Nam ta).
3. Các ông lớn trên thế giới, như Tesla, tuy có đội ngũ lập trình viên xuất sắc nhưng các dòng tự hành của họ vẫn vướng vào các vụ kiện liên quan đến tai nạn giao thông.

Vì vậy để nghiên cứu và thực hiện đề tài này, trước tiên ta sẽ tiến hành trong môi trường lý tưởng trước, từ đó sẽ mở rộng, thêm các yếu tố ngoại vi gắn liền hơn với thực tế. Từ đó nhóm đã quyết định mô phỏng lại một đô thị thông minh. Trong thế giới mô phỏng mà nhóm đã thực hiện được:

- Có các đường nhựa, với 2 lane đường, và vạch phân cách giữa 2 lane đường là nét đứt (tăng độ khó cho việc bám line)
- Có các ngã ba, giúp xe có thể quẹo hay đi thẳng.
- Có 3 loại biển báo: stop, turn\_left, và turn\_right hỗ trợ điều hướng
- Có bầu trời và mây bay là yếu tố gây nhiễu khi nhận diện biển báo.

### 1.3 Các tính năng:

Mô hình xe tự hành của nhóm được phát triển với các tính năng cơ bản sau:

- Có khả năng phân biệt làn đường, và đi ở giữa làn đường đã phát hiện đó
- Có khả năng nhận diện biển báo bằng camera (sử dụng công nghệ nhận diện ảnh bằng Deep Learning với mạng CNN)



Nhóm cũng khẳng định rằng, đây chỉ là những tính năng cơ bản nhất để xe có thể áp dụng trong môi trường lý tưởng (không có người đi qua đi lại, không có vật cản, hay đường trơn), đồng thời nhóm cũng đề ra các hướng phát triển khả quan trong tương lai:

1. Có khả năng né các vật cản động (người, phương tiện) và vật cản tĩnh (đá lở, tuyết, ...).
2. Cải thiện độ chính xác và mở rộng số lượng biển báo nhận biết được

## 2 Nội dung

### 2.1 Tìm hiểu về hệ điều hành ROS

#### 2.1.1 ROS là gì



ROS (Robot operating system) là một hệ thống phần mềm chuyên dụng để lập trình và điều khiển robot, cung cấp các thư viện và công cụ để giúp các nhà phát triển phần mềm tạo ra các ứng dụng robot. ROS có những ưu điểm nổi bật như là:

1. ROS là hệ điều hành meta, mã nguồn mở, hoàn toàn miễn phí
2. ROS có thể được lập trình bằng nhiều ngôn ngữ khác nhau như C++, Python, ... (source code của nhóm được viết chủ yếu bằng Python)
3. ROS sử dụng mã (trình điều khiển và thuật toán) từ các dự án nguồn mở khác:
  - Trình mô phỏng dự án Player / Stage
  - Thư viện xử lý hình ảnh và tầm nhìn nhân tạo từ OpenCV
  - Thuật toán lập kế hoạch từ OpenRave
  - v.v

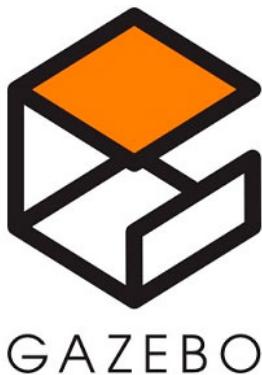
#### 2.1.2 Cài đặt ROS:

Hiện nay có khá nhiều phiên bản ROS như là ROS Kinetic, ROS Melodic, ROS Dashing,... Do phần lớn thành viên trong nhóm sử dụng hệ điều hành Ubuntu phiên bản 20.04 nên nhóm quyết định sử dụng ROS Noetic vì đây là phiên bản hỗ trợ tốt nhất dành cho Ubuntu 20.04 (Link tải [Ubuntu 20.04](#), [ROS Noetic](#))

## 2.2 Mô phỏng trên Gazebo

#### 2.2.1 Gazebo là gì?

Gazebo là một công cụ mô phỏng 3D. Gazebo có thể được áp dụng cho robot design, testing AI khi ở giai đoạn đầu của các nghiên cứu. Nó được nhiều người ưa chuộng bởi:



1. Gazebo cung cấp GUI rất thân thiện với người dùng, với các thanh menu, có thể giúp người dùng tương tác trực tiếp trên gazebo mà không cần thông qua edit file code.
2. Gazebo là một công cụ mã nguồn mở, không tính phí.
3. Gazebo có một cộng đồng lớn.

### 2.2.2 Cài đặt Gazebo:

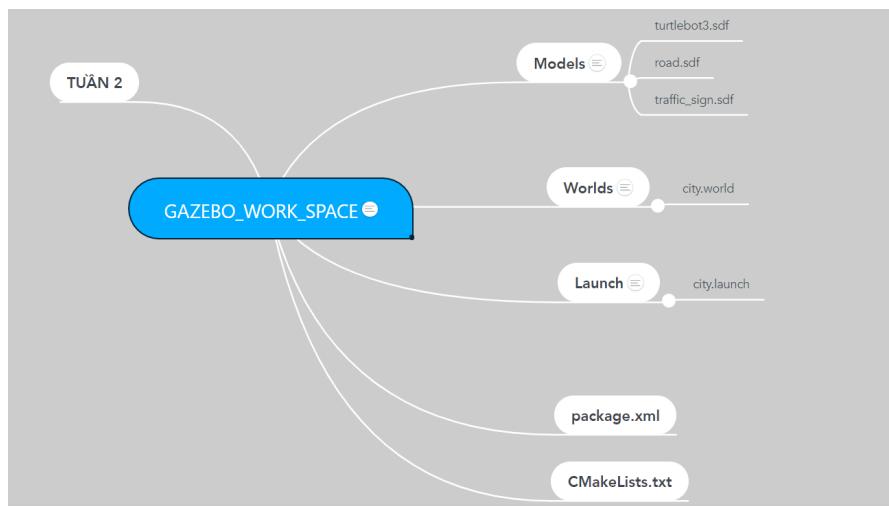
Như đã giới thiệu ở trên, Turtlebot chạy trên hệ điều hành ros, và ros tương thích tốt đối với Ubuntu, nên nhóm đã quyết định cài đặt **Gazebo phiên bản 11.0.0** trên Ubuntu để tiện cho việc mô phỏng. ([Link để tải Gazebo](#))

Ngoài ra, Gazebo chỉ là công cụ để hiện mô phỏng, nhưng để edit các model, 3d object như thêm màu, cắt, thay đổi hình bề mặt, ... ta cần 1 công cụ hỗ trợ khác cũng được nhiều người sử dụng không kém là **SketchUp Pro v21.0.339**. Đây là công cụ có tính phí, ta có thể tải ở ([link sau](#))

### 2.2.3 Không gian làm việc Gazebo:

Không gian làm việc của Gazebo cần có những thành phần sau:

- 2 file **package.xml** và **CMakeList.txt** để giúp cấu hình, set up bên trong
- **Thư mục Model:** Chứa các vật thể, object có hình dạng, màu sắc và collision (viền va chạm), và các thông số vật lý khác. Những object này sẽ được lưu trong file có đuôi **.sdf**. Các ví dụ về model có thể là:
  - vật tĩnh: nhà cửa, cây cối, biển báo, đường xá, ...
  - vật động: xe cộ, robot, con người, ...
- **Thư mục Worlds:** gồm các file **.world**, trong file này sẽ chứa các model nhỏ bên trong và vị trí của chúng xuất hiện trong một thế giới rộng hơn. Đây cũng là các map mà ta sẽ xây dựng để mô phỏng.
- **Thư mục Launchs:** gồm các file **.launch**, và sẽ được gọi trong terminal bằng lệnh **roslaunch**. Khi đó turtlebot của chúng ta sẽ được spawn trong thế giới đã định nghĩa trong file launch.



Hình 1: Không gian làm việc của Gazebo ([Link mind map](#))

#### 2.2.4 Các model và các map đã build:

##### 1. Các model:

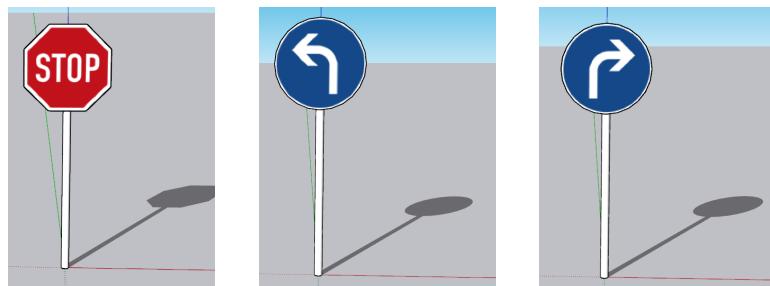
Như đã giới thiệu ở trên về công cụ SketchUp, nó có thể giúp ta thiết kế những model giống với thực tế như biển báo, ta có thể vẽ một cái trụ tròn cao, và một mặt phẳng hình tròn, đồng thời sơn lên bề mặt của hình tròn đó hình biển báo ta cần, như vậy là ta có được 1 model biển báo.

Đối với các model phức tạp hơn đòi hỏi sự chuyên nghiệp và tính thẩm mỹ, ta có thể tìm kiếm trên cộng đồng mã nguồn mở của [SketchUp 3D](#).

Một số model đã build:



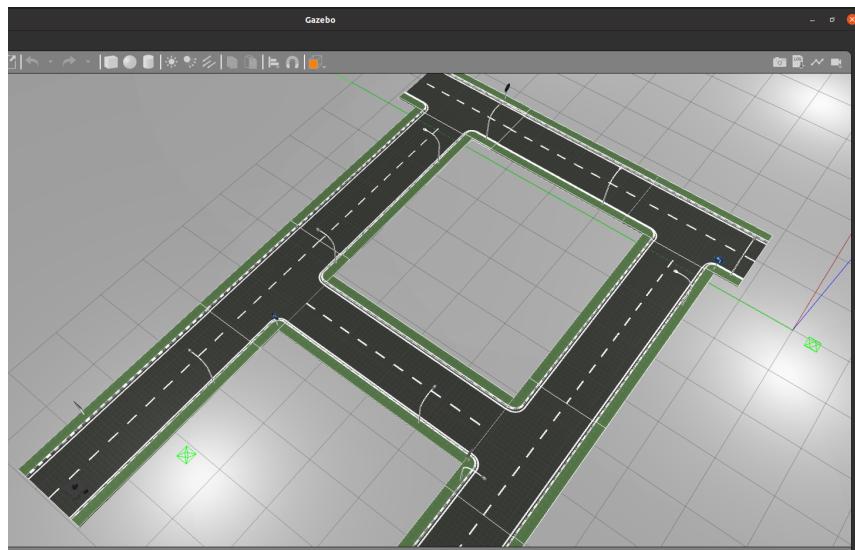
Hình 2: Các model mảng ghép đường



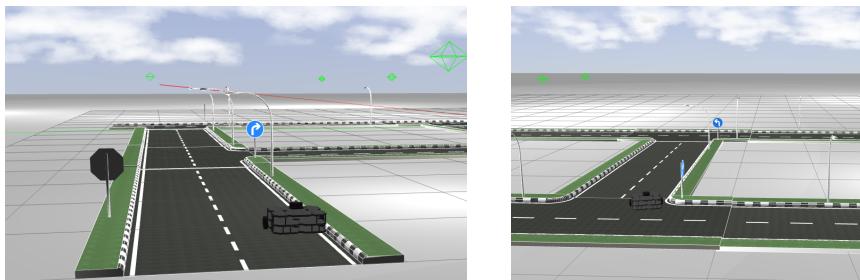
Hình 3: Các model biển báo

## 2. Các map:

Map mà nhóm đã sử dụng để chạy kiểm thử chương trình và demo ở bên dưới:

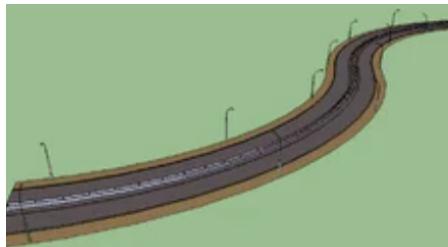


Hình 4: Tổng quan map



Hình 5: Một số khung cảnh từ map trên

Nhưng trước khi đi vào nhận diện đường có nét đứt trên, ta phải thử nghiệm việc bám lane trên map đường nét liền và có độ cong không quá lớn như hình 6.



Hình 6: Đường cong

### 2.3 Phát hiện biển báo và phân biệt biển báo

Để phân biệt được biển báo trên đường ta cần trải qua 2 step chính: **traffic sign detection** và **traffic sign classification**.

- **Traffic sign detection:** camera của turtlebot là một camera nhìn bao quát, trong góc nhìn đó có cả biển báo, phần lane đường, và một bộ phận gây nhiễu như bầu trời, cây cối, ... Traffic sign detection là việc giúp tìm ra vùng chú ý (interest place) đúng ngay vùng có biển báo và crop hình có biển báo đó ra, sau đó mới tiến hành phân biệt biển báo gì.
- **Traffic sign classification:** đây là bước quan trọng nhất, là yếu tố quyết định độ thành công của xe. Nhiệm vụ của classification là tìm ra được model có thể nhận biết được nhiều biển báo nhất, và độ chính xác không quá thấp.

#### 2.3.1 Phát hiện biển báo (sign detection):

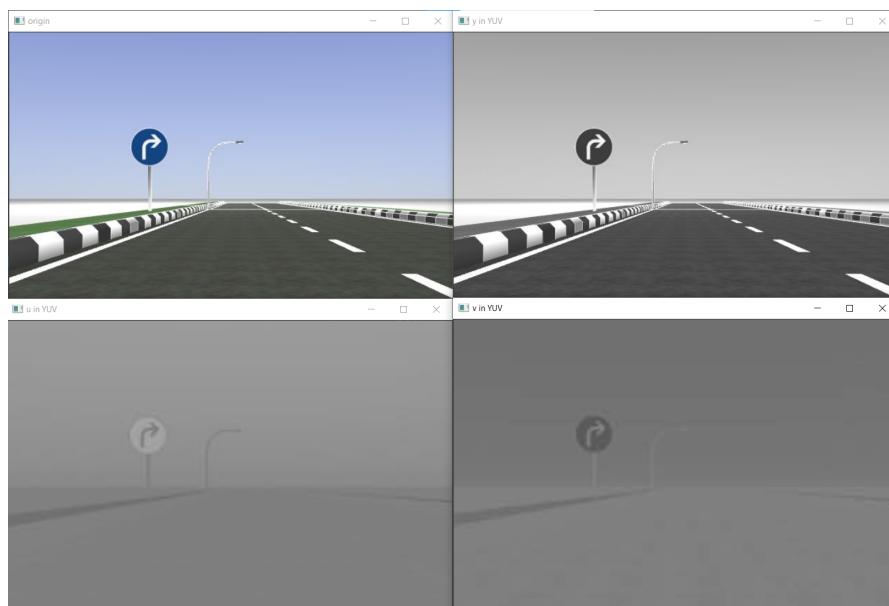
Hệ thống xử lý dữ liệu thông qua hình ảnh mà camera truyền về. Từ bức ảnh này ta có thể đọc ra các dữ liệu về làn đường, dữ liệu về vật cản và cả dữ liệu về biển báo. Như vậy ta cần thông qua một quy trình để nhận diện được biển báo trên bức ảnh đầy thông tin đó:

1. Chuyển ảnh về hệ **V** trong mô hình màu **YUV**.
2. Áp dụng **phân ngưỡng ảnh** (threshold)
3. Áp dụng **findContours()** để tìm vùng chú ý
4. Lọc các contour vừa tìm được, để tìm vùng chú ý tối ưu nhất.
5. Crop vùng chú ý từ bức tranh lớn ban đầu và đem cho model phân biệt biển báo

##### 2.3.1.1 Chuyển ảnh về hệ màu V trong mô hình màu YUV

Hình ảnh ta thu được từ camera là tấm hình với 3 kênh màu RGB (Red, Green, Blue), hình RGB thường được sử dụng trong các phần mềm photoshop bởi nó thân thiện với người dùng. Nhưng trong lĩnh vực thị giác máy tính, hệ màu **YUV** lại tối ưu hơn, bởi về sự mã hóa hiệu quả về màu sắc của nó giúp giảm băng thông hơn so với RGB.

Đi sâu hơn về mô hình màu YUV, màu YUV gồm 3 giá trị Y, U, và V. Trong đó Y biểu thị độ sáng (Luminosity), U và V biểu thị độ khác biệt về màu sắc (Color Difference). Đối với mắt người, thông thường kênh Y là kênh chứa nhiều thông tin nhất, nhưng ta đang cần giảm bớt lượng thông tin mà bức ảnh đem lại, thay vào đó là tập trung vào đúng khu màu của biển báo, nên ta sẽ sử dụng một trong hai kênh màu U hoặc V cho bước tiếp theo. Như trong hình 7, ta có thể thấy bức ảnh y còn chứa nhiều thông tin không cần thiết như vạch kẻ đường, vùng sọc ngựa vằn,..., trong khi đó ảnh u và v số thông tin hiển thị đã ít đi, nhưng vẫn tồn tại biển báo trong đó.



Hình 7: Chuyển ảnh từ rgb sang yuv và tách làm 3 kênh màu y, u, và v

Việc chuyển đổi về hệ YUV và chỉ lấy một kênh màu **V** này sẽ giúp giảm nhỏ lại mảng thông tin của bức ảnh, đồng thời tăng tốc độ xử lý fps của quá trình phát hiện ảnh lên.

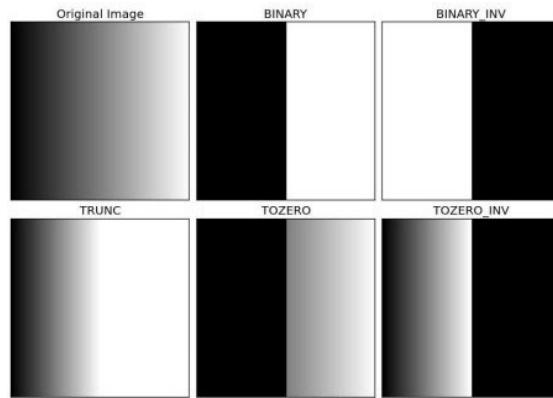
### 2.3.1.2 Phân ngưỡng ảnh (threshold)

Phân ngưỡng là gì? Phân ngưỡng đúng như tên gọi của nó, từ một ảnh xám và một ngưỡng thresh **T** cho trước, và ta sẽ duyệt hết mảng thông tin của một bức ảnh và so sánh từng giá trị với ngưỡng đó. Ta có hai loại phân ngưỡng:

1. **Threshold:** Hàm threshold đơn giản này cũng có nhiều phương thức gán giá trị khác nhau:

- THRESH\_BINARY:
  - Nếu giá trị pixel lớn hơn ngưỡng thì gán bằng maxval
  - Ngược lại gán bằng 0
- THRESH\_TRUNC
  - Nếu giá trị pixel lớn hơn ngưỡng thì gán giá trị bằng ngưỡng
  - Ngược lại giữ nguyên giá trị
- THRESH\_TOZERO

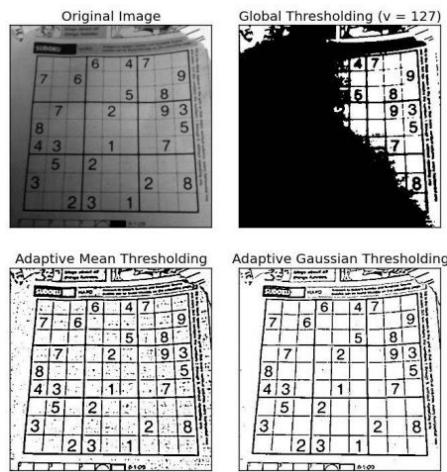
- Nếu giá trị pixel lớn hơn ngưỡng thì giữ nguyên giá trị
- Ngược lại gán bằng 0



Hình 8: Ví dụ về Threshold với  $T=127$

2. **Adaptive Threshold:** Bởi phương pháp threshold thông thường có nhược điểm là ánh sáng không đồng đều trong ảnh, nên **Adaptive Threshold** được ra đời. Phương pháp này sẽ tính giá trị trung bình của n điểm xung quang pixel đó và trừ cho một số nguyên C. Adaptive Threshold cũng có 2 phương thức thường được sử dụng:

- ADAPTIVE\_THRESH\_MEAN\_C: giá trị của pixel phụ thuộc vào các pixel lân cận
- ADAPTIVE\_THRESH\_GAUSSIAN\_C: giá trị của pixel cũng phụ thuộc vào các pixel lân cận, tuy nhiên được khử nhiễu

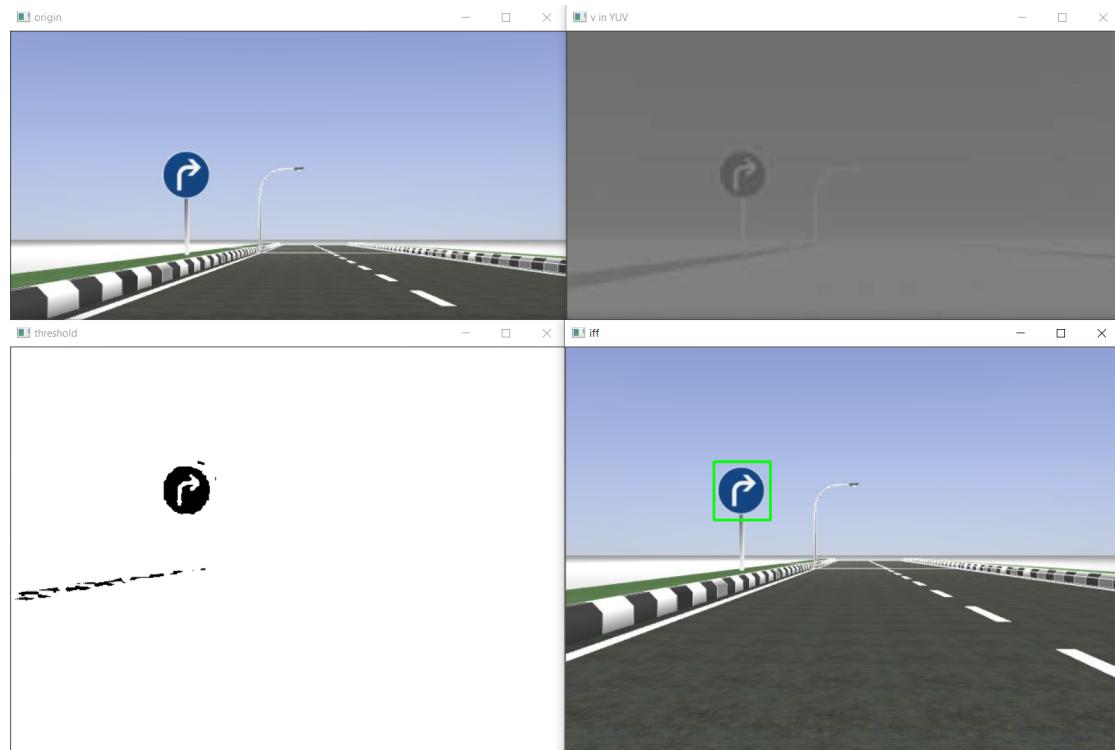


Hình 9: Ví dụ về Adaptive Threshold

Hiện tại, nhóm đang áp dụng **threshold** cơ bản với loại phân ngưỡng **THRESH\_BINARY** để phát hiện biển báo, với tác dụng nhằm tăng độ tương phản, với mục tiêu tách biệt khôi biển báo và nền phía sau. Do đây là project hiện tại chỉ được thực hiện trong môi trường mô phỏng

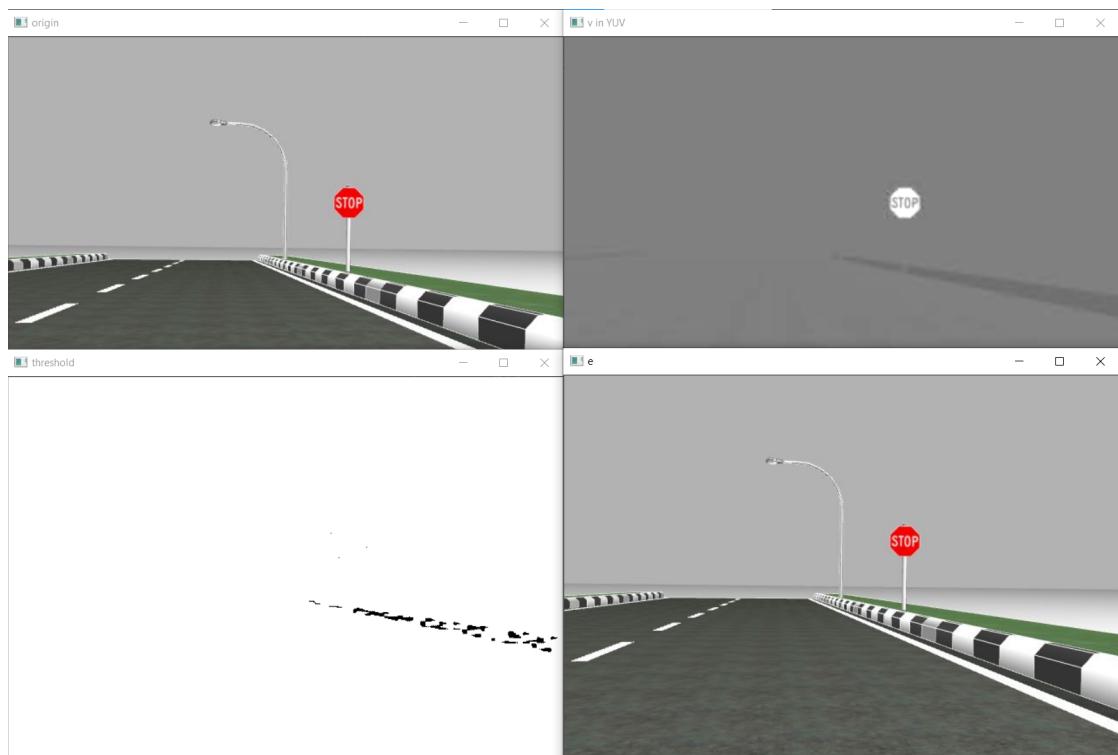
với ánh sáng giả lập khá đồng đều, nên nhóm ban đầu mới áp dụng **THRESH\_BINARY**, nhưng trong tương lai, nếu có sự hỗ trợ để dự án được tiếp tục, nhóm sẽ cải tiến bằng việc áp dụng **ADAPTIVE\_THRESH\_GAUSSIAN\_C** nhằm cải thiện khả năng nhận diện trong điều kiện thực tế có ánh sáng không đồng đều.

Vậy ta cần lựa chọn ngưỡng **T** phù hợp để có thể nhận biết tối ưu nhất biển báo. Bằng việc chạy vòng for từ **T=0** đến **T=255**, ta kiểm được giá trị **T=110** giúp ta nhận diện biển báo tốt nhất (đối với biển **turn left**, và biển **turn right** màu xanh dương như hình 10)



Hình 10: Phát hiện biển báo

Tuy nhiên, khi ta áp dụng ngưỡng **T=110** đó đối với biển báo **stop** màu đỏ, hình ảnh thu được sau khi qua threshold lại là một vùng đồng màu (như hình 11).



Hình 11: Không phát hiện được biển báo đỏ

Như vậy với ngưỡng 110, hệ thống ta sẽ bị mù màu đỏ. Và để giải quyết vấn đề trên, nhóm đã thử thay đổi ngưỡng  $T$  bằng việc chạy vòng for như cũ, và phát hiện với  $T=150$ , hệ thống có thể phát hiện màu đỏ nhưng không phát hiện màu xanh.

Tổng hợp cả hai ý trên, để nhận biết được biển báo, ta sẽ thông qua cả hai threshold  $T=110$  và  $T=150$  để nhận biết vùng của biển báo.

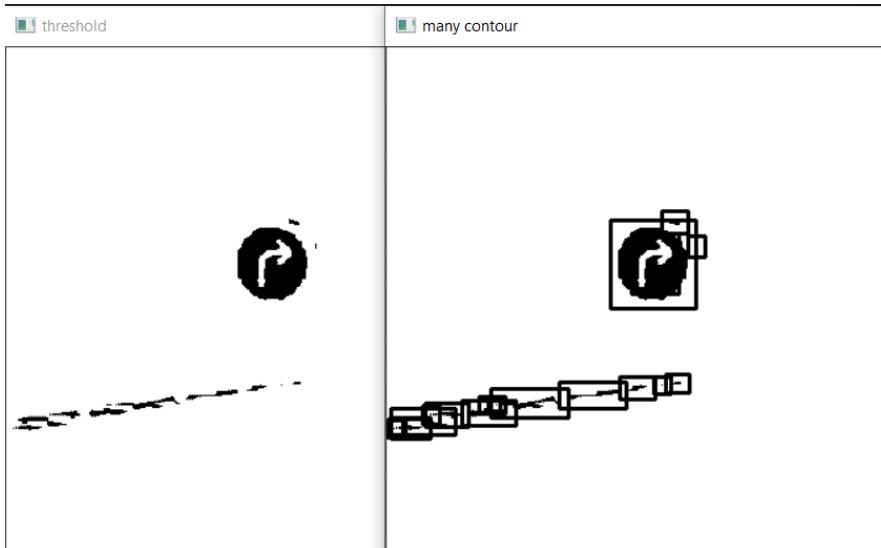
### 2.3.1.3 Áp dụng `findContours()` để tìm vùng chú ý và lọc các vùng không cần thiết

Như tên gọi của hàm `findContours()` ta sẽ đi tìm contour. Vậy contour là gì? Contour được hiểu nôm na là một đường cong liên kết toàn bộ những điểm liên tục theo đường biên mà có cùng màu sắc hay giá trị về cường độ.

Và khi áp dụng vào thực tế, ta lại gặp phải một vấn đề. Đó là có nhiều hơn 1 vùng được phát hiện như hình 12 (các contour được phát hiện được thể hiện trong khung màu đen).

Mà biển báo ta cần là một trong những contour được phát hiện trên. Khi đó để lọc và loại bỏ các vùng contour không mong muốn bằng cách:

- Tính diện tích của các contour, và chặn hai đầu của giá trị diện tích đó (loại bỏ các contour quá nhỏ hoặc quá lớn)
- Bởi biển báo có hình tròn, nên khung của biển báo sẽ là hình vuông, từ đó để loại bỏ các contour rác, ta có thể loại bỏ các contour có tỷ lệ chiều dài và chiều rộng quá lớn (loại các contour hình chữ nhật)

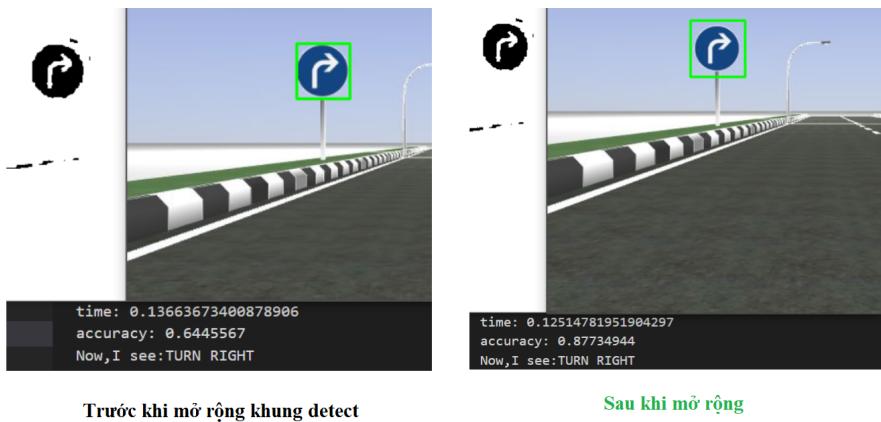


Hình 12: Nhiều hơn 1 contour được phát hiện

#### 2.3.1.4 Crop vùng chú ý sau khi đã tối ưu

Sau bước trên, ta đã kiểm thấy vùng có biển báo, tuy nhiên khi ta crop phần biển báo vừa in thì accuracy khi test với model sẽ thấp. Nghĩa là ví dụ sau khi loại bỏ các contour rác, ta đã có được contour tối ưu nhất chứa biển báo, nếu ta crop y nguyên phần ảnh có cùng kích thước với contour đó thì accuracy lúc thử nghiệm với model sẽ thấp.

Thay vào đó, ta sẽ crop rộng hơn một ít như hình 33 nhằm tăng độ chính xác cho model phân biệt biển báo.



Hình 13: Mở rộng khung detection

### 2.3.2 Phân biệt biển báo (sign classification)

Nhằm phân biệt biển báo thông qua xử lý ảnh, ta có thể áp dụng mô hình YOLO (You Only Look Once) hoặc CNN (Convolutional Neural Network). Theo như nhóm đã tìm hiểu, mô hình YOLO có tốc độ xử lý chậm hơn so với CNN. Mà đây là dự án về xe tự hành với mục tiêu áp dụng vào thực tiễn với tốc độ vừa (30-40km/h), nên yêu cầu về fps (frame per second) cao. Chính vì thế, nhóm đã sử dụng CNN (mạng nơ ron tích chập) để phân biệt các biển báo. Trước tiên, ta cần training ra một model theo các bước sau:

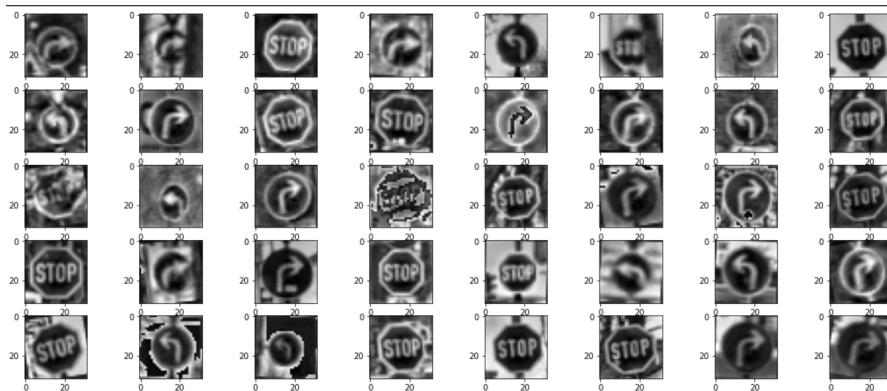
1. Thu thập tìm kiếm dataset
2. Thiết kế các layer trong model
3. Training, và check quality (Ta sẽ quay lại từ bước 2 hoặc có thể là bước 1, nếu kết quả validation không đạt như mong muốn).

#### 2.3.2.1 Thu thập, tìm kiếm dataset

Công việc quan trọng nhất, và có tầm quan trọng ảnh hưởng đến kết quả của model nhất đó là bước thu thập dữ liệu. Đây cũng là bước khó khăn đối với nhóm, bởi nhóm toàn bộ đều là sinh viên không đủ thời gian để tự đi chụp ảnh các biển báo thật ở ngoài đường Việt Nam, không đủ kinh phí để mua lại dataset của biển báo Việt Nam. Nên nhóm đã quyết định sử dụng các dataset từ các mã nguồn mở. Việc này cũng kèm theo những ưu, và nhược điểm sau:

- **Ưu điểm:** Các dataset này hoàn toàn miễn phí
- **Nhược điểm:** Không kiểm thấy dataset của biển báo Việt Nam, thay vào đó là dataset về biển báo Trung, Đức, ... Đồng thời ta cần loại bỏ đi những bức ảnh nằm ngoài phạm vi dự án.

Cuối cùng, nhóm đã thống nhất chọn tập dataset của Đức, tập biển báo này bao gồm 2405 tấm hình của mỗi loại biển báo như hình 14.



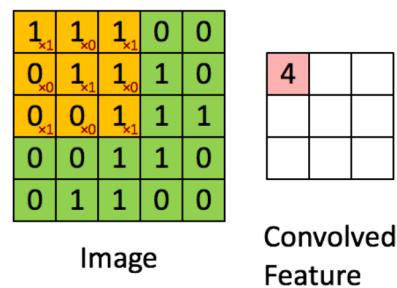
Hình 14: Tập dataset dùng để train model

### 2.3.2.2 Thiết kế các layer trong model

Trước hết CNN là gì?

Như đã giới thiệu sơ bộ ở trên CNN (Convolutional Neural Network) là Mạng nơ-ron tích chập, đây là một trong những mô hình deep learning được sử dụng phổ biến hiện nay. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay.

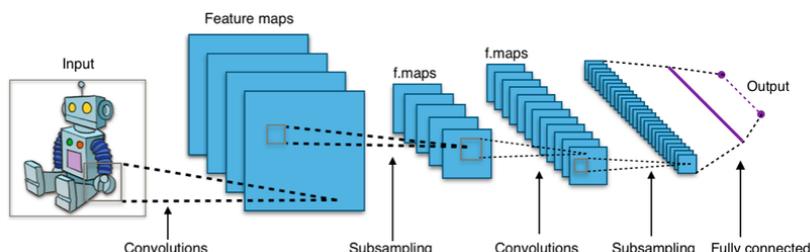
Trong CNNs có sử dụng quá trình trượt một window trong một ma trận ảnh như hình 16. Cụ thể ta sẽ trượt khối màu cam **3x3** trên một ma trận ảnh cần detect vật gì đó (ma trận ảnh sẽ lớn hơn khối màu cam). Kích thước của khối màu cam sẽ được gọi là **filter**. Và sau khi trượt hết ma trận ảnh, ta sẽ thu được Convolved Feature.



Hình 15: Hình ảnh minh họa về Sliding window

Mạng CNN là tập hợp của nhiều lớp Convolution chồng lên nhau. Ta có thể áp dụng các hàm nonlinear activation như ReLU, softmax, hàm optimizer như adam và hàm loss cụ thể là categorical\_crossentropy.

Khi đó, Các layer liên kết được với nhau thông qua cơ chế convolution. Như vậy, mỗi neuron ở lớp kế tiếp được sinh ra từ kết quả của filter áp đặt lên một vùng ảnh của neuron trước đó. Và trong mỗi một layer đó, ta lại sử dụng các thông số khác nhau, cùng với các layer khác nhau như (conv2D, MaxPooling, Flattenr, Dense, ....) Như vậy tùy thuộc vào các lớp layer và các thông số, ta sẽ đạt được model mong muốn.



Hình 16: Ví dụ về CNNs Layers

**Lưu ý:** Ta không thể tìm được một model có thể thỏa mãn tất cả các trường hợp được. Nên



ta chỉ có gắn tăng độ chính xác cao nhất có thể cho các model.

Ban đầu nhóm áp dụng theo một giải thuật có sẵn trên mã nguồn mở, nhưng không đạt được yêu cầu của nhóm. Model sau khi train gặp tình trạng **overfitting** (nghĩa là model lúc testing trên các tập dữ liệu mới thì không thể classify ra đúng biển báo, và dù kết quả ra sai, nhưng accuracy mà model thông báo lại rất cao)

Chính vì thế nhóm tuy vẫn giữ nguyên các lớp layer, nhưng thay đổi các thông số như filter và của layer **Dense**. Cụ thể ta có bảng tóm tắt model sau (hình 17):

### ▼ Summary Model

```
✓ 0 giây
▶ model[0].summary()

Model: "sequential_2"

+-----+
| Layer (type)        | Output Shape | Param # |
+=====+=====+=====+
| conv2d_2 (Conv2D)    | (None, 32, 32, 32) | 832      |
| max_pooling2d_2 (MaxPooling 2D) | (None, 16, 16, 32) | 0         |
| flatten_2 (Flatten)   | (None, 8192)   | 0         |
| dense_4 (Dense)      | (None, 370)    | 3031410  |
| dense_5 (Dense)      | (None, 3)      | 1113     |
+=====+=====+=====+
Total params: 3,033,355
Trainable params: 3,033,355
Non-trainable params: 0
```

Hình 17: Tóm tắt các layer trong model

#### 2.3.2.3 Training và check quality

Dây là bước kiểm thử của model, đầu tiên ta sẽ xuất ra các giá trị xác suất lúc train model (hình 19)

```
Tệp Chính sửa Xem Chèn Thời gian chạy Công cụ Trợ giúp Chính sửa lần gần đây nhất vào 11 tháng 12
+ Mã + Văn bản
Model
Epoch 1/5
1214/1214 [=====] - 8s 6ms/step - loss: 0.0770 - accuracy: 0.9769 - val_loss: 0.0099 - val_accuracy: 0.9958 - lr: 7.7378e-04
Epoch 2/5
1214/1214 [=====] - 8s 6ms/step - loss: 0.0097 - accuracy: 0.9977 - val_loss: 0.0066 - val_accuracy: 0.9708 - lr: 7.3509e-04
Epoch 3/5
1214/1214 [=====] - 8s 6ms/step - loss: 0.0121 - accuracy: 0.9970 - val_loss: 0.0325 - val_accuracy: 0.9875 - lr: 6.9834e-04
Epoch 4/5
1214/1214 [=====] - 8s 6ms/step - loss: 3.9376e-04 - accuracy: 1.0000 - val_loss: 0.0024 - val_accuracy: 1.0000 - lr: 6.6342e-04
Epoch 5/5
1214/1214 [=====] - 8s 6ms/step - loss: 0.0140 - accuracy: 0.9967 - val_loss: 0.0177 - val_accuracy: 0.9917 - lr: 6.3025e-04

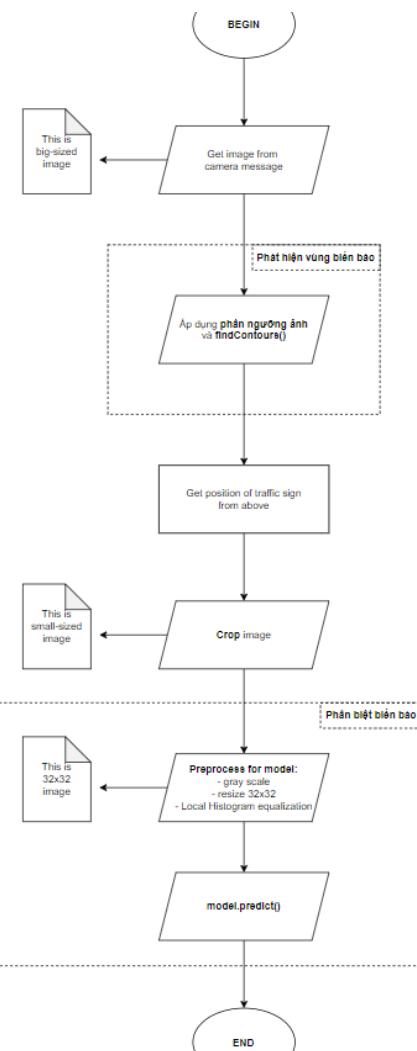
Model with filters 5x5, epochs=5, training accuracy=1.00000, validation accuracy=1.00000
```

Hình 18: training model

Và ta sẽ chạy thử với các bức ảnh ta chụp từ camera của turtlebot trong Gazebo Simulation. Nhóm đã testing với các điều kiện về ánh sáng. Bấm vào [Link](#) sau để xem chi tiết về việc kiểm tra độ chính xác model của nhóm. Khi đó, nếu model vừa train có độ chính xác thấp, hay bị overfitting, Ta sẽ quay lại bước **thu thập dataset** hoặc bước **thiết kế layer** trong model.

### 2.3.3 Áp dụng model vừa train vào hệ thống

Dây là bước cuối cùng, ta sẽ kết hợp những gì đã làm ở cả **sign detection** và **sign classification**. Để dễ hiểu hơn, ta có thể xem ở flowchart sau hoặc để xem hình ảnh rõ hơn ta có thể xem ở [link này](#):



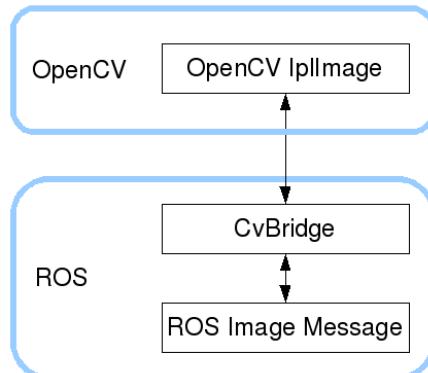
Hình 19: Flowchart cho quá trình phát hiện biển báo

Đây là tóm tắt lại quy trình ở flowchart trên:

1. Get image from camera message
2. Áp dụng "Phát hiện biển báo" để tìm ra vị trí của biển báo trong bức hình
3. Crop khung hình chứa biển báo
4. Áp dụng `model.predict()` sau khi đã train xong model từ "Phân biệt biển báo"

#### 2.3.3.1 Get image from camera message

Đây là lĩnh vực liên quan đến **ROS**. Cụ thể, camera từ turtlebot sẽ gửi đi (publish) một message, message này chính là mã hóa của bức ảnh được lưu dưới dạng ma trận. Mã hóa này thì ngôn ngữ lập trình không thể đọc hiểu được. Nên để lấy được tấm ảnh từ message trên ta phải convert nó về dạng ảnh.



Hình 20: Thư viện CvBridge

Trong Python, có một thư viện là **CvBridge**, đây là cây cầu để chuyển format giữa message của ros và format dạng ảnh mà OpenCV có thể đọc được (xem hình 20).

#### 2.3.3.2 Áp dụng phát hiện biển báo

Như đã trình bày ở phần **Phân biệt biển báo**, ta sẽ áp dụng các bước: (Để rõ hơn, hãy quay lại phần [2.3.1 Phát hiện biển báo \(sign detection\):](#))

1. Chuyển ảnh về hệ V trong mô hình màu **YUV**.
2. Áp dụng **phân ngưỡng ảnh** (threshold)
3. Áp dụng `findContours()` để tìm vùng chú ý
4. Lọc các contour vừa tìm được, để tìm vùng chú ý tối ưu nhất.
5. Crop vùng chú ý từ bức tranh lớn ban đầu và đem cho model phân biệt biển báo

Sau khi bước này thực hiện xong ta sẽ nhận được vị trí của biển báo trong bức hình gốc.



### 2.3.3.3 Áp dụng model.predict

Dây là bước sau khi ta thực hiện **Phân biệt biển báo** và nhận được model. Khi này, ta sẽ import model vào folder chứa code cho tiện việc truy vấn và sử dụng, đồng thời ta tuân theo 2 bước sau:

1. Tiền xử lý ảnh đầu vào
2. Sử dụng **model.predict**

Từ các bức ảnh đã crop ở trên, ta sẽ đem chúng đi tiền xử lý qua các bước sau:

- **Gray-scale:** Làm xám ảnh lại.

Do ảnh ban đầu có định dạng là RGB là một tập hợp gồm các điểm, mà mỗi điểm là bộ 3 màu red, green và blue, như vậy sẽ chứa các thông tin không cần thiết. Việc gray-scale giúp cho ta giảm bớt số lượng thông tin dư thừa, đồng thời phù hợp với model chúng ta train (lúc train model ta sử dụng ảnh gray-scale, nên khi ta predict ta cũng phải sử dụng ảnh gray-scale),

- **resize to 32x32**

Tùy thuộc vào vị trí đứng của xe, mà biển báo nhìn thấy được trong hình sẽ to hay nhỏ, nên ta cần resize lại về cùng 1 kích thước. Khi đó ta mới có thể đem vào model.predict.

- **Local Histogram Equalization**

Dây là phương pháp tăng độ tương phản cho ảnh khi ở dạng gray-scale. Nhờ vào phương pháp này, ta có thể tăng độ chính xác khi predict.

Sau khi tiền xử lý, ta đã có bức ảnh tốt nhất với kích thước 32x32, khi này, ta sẽ sử dụng model của chính chúng ta train để nhận biết biển báo bằng việc sử dụng hàm **model.predict()** và gán giá trị trả về vào một biến **prediction**. Khi này ta có thể biết được các thông tin sau:

- **Biển báo nào:** Ta sử dụng **np.argmax(prediction)**, hàm này sẽ trả về số tương ứng với loại của biển báo. Cụ thể đối với project này của nhóm, các số sẽ tương ứng với:

1. STOP SIGN
2. TURN LEFT SIGN
3. TURN RIGHT SIGN

- **Xác suất khi phân biệt biển báo đó:** Ta sử dụng **np.amax(prediction)**, hàm này sẽ trả về % xác suất khi phân biệt biển báo với giá trị từ 0 đến 1.

## 2.4 Nhận diện làn đường

Trước khi đi vào chi tiết cụ thể, ta có một quy trình như sau để có thể nhận diện được làn đường trên một bức ảnh, hoặc một video.

1. Tiền xử lý hình ảnh với **Gray Scale** và **Gaussian Blur**
2. Sử dụng **Canny Edge Detection**
3. Tạo vùng chú ý (Region-of-interest hay còn được gọi là tạo Masking)
4. Sử dụng **Line Hough Transform** cho vùng ảnh đã được tạo masking **Region-of-interest**.

5. Ngoại suy các đường được tìm thấy từ **Line Hough Transform** để có thể xây dựng được phương trình của làn trái đường trái và phải
6. Phác họa các đường thẳng, thông tin đã được tính toán và xây dựng lên video/ hình ảnh đầu vào để trực quan hóa dữ liệu cần thiết đã được xử lý.

Sau đây chúng ta sẽ tiến hành phân tích kĩ hơn cho từng bước trên, về mặt lí thuyết cũng như là ứng dụng để có thể phát hiện ra được làn đường cần phát hiện như ta mong muốn

#### 2.4.1 Tiền xử lí (Image Preprocessing)

- **Grayscale:** Hình ảnh RGB chứa nhiều dữ liệu có thể không cần thiết cho quá trình xử lý. Khi thực hiện chuyển đổi hình ảnh RGB thành thang màu Xám (Grayscale), chúng ta sẽ loại bỏ nhiều thông tin không cần thiết để xử lý.

Đối với hình ảnh tỷ lệ RGB, đối với mỗi thành phần, tức là R, G, B, hình ảnh có các cường độ khác nhau. Hình ảnh RGB được thể hiện bằng 3 kênh. mỗi kênh thường bao gồm 8 bit. Do đó, rất nhiều dữ liệu để lưu trữ và hoặc thao tác.

Khi chuyển đổi hình ảnh thành hình ảnh có thang màu Xám (Grayscale), thì mặc dù chúng ta sẽ mất tất cả thông tin liên quan đến màu sắc, nhưng các sự vật sẽ được thể hiện bằng tông màu Xám chỉ với một kênh duy nhất (vẫn sẽ là một thành phần duy nhất). Như vậy, giảm được nhiều thông tin không cần thiết và chúng ta cũng sẽ tiết kiệm được nhiều sức mạnh tính toán. Đồng thời việc tính toán trên ma trận 2D (Gray) dễ dàng hơn và nhanh hơn so với ma trận 3D (RGB).

Khi ứng dụng Grayscale cho hệ thống, ta nhận được hình ảnh như sau:



Hình 21: Kết quả trả về của hệ thống khi sử dụng grayscale

- **Gaussian Blur:** Một kĩ thuật làm mờ (làm mịn) một hình ảnh bằng cách sử dụng chức năng Gaussian để giảm mức độ nhiễu. Nó có thể được coi là một bộ lọc giúp giảm nhiễu hình ảnh và các chi tiết có thể bỏ qua trong hình ảnh, từ đó loại bỏ các thành phần tần số rất cao vượt quá tần số được liên kết với bộ lọc gradient được sử dụng. Nếu không, chúng có thể gây ra các cảnh sai được phát hiện. Việc Gaussian Blur này thường đạt được bằng cách tích chập ma trận hình ảnh với Gaussian Kernel. Ở đây dựa vào thư viện khuyên dùng, chúng ta sẽ sử dụng Gaussian Kernel 5x5.

Khi ứng dụng Gaussian Blur cho hệ thống, ta nhận được hình ảnh như sau:



Hình 22: Kết quả trả về của hệ thống khi sử dụng Gaussian Blur

#### 2.4.2 Ứng dụng kĩ thuật Canny Edge Detection

Ta cần phát hiện các cạnh để phát hiện làn đường vì độ tương phản giữa làn đường và mặt đường xung quanh cung cấp thông tin hữu ích về việc phát hiện các vạch của làn đường.

Phát hiện cạnh Canny (Canny Edge Detection) là một thuật toán sử dụng các độ dốc ngang và dọc của các giá trị pixel của hình ảnh để phát hiện các cạnh. Về mặt chi tiết, thuật toán có thể được mô tả như sau.

Canny Edge Detection là một thuật toán phát hiện cạnh phổ biến. Nó được phát triển bởi John F. Canny. Nó là một thuật toán nhiều giai đoạn và chúng ta sẽ đi qua từng giai đoạn.

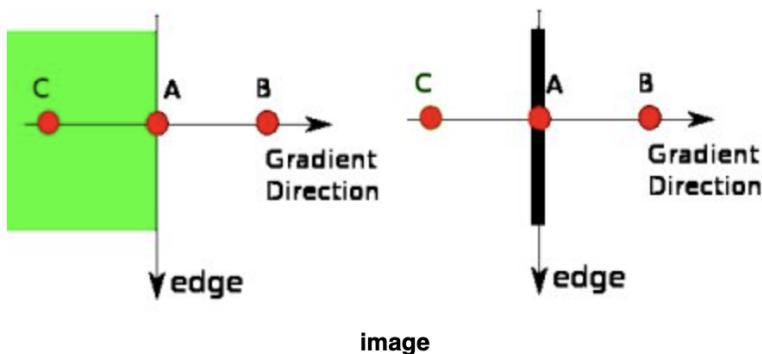
1. **Noise Reduction:** Vì tính năng phát hiện cạnh dễ bị nhiễu trong ảnh, nên bước đầu tiên là loại bỏ nhiễu trong ảnh bằng bộ lọc Gaussian 5x5.
2. **Finding Intensity Gradient of the Image:** Ảnh được làm mịn sau đó được lọc bằng Sobel Kernel theo cả hướng ngang và dọc để thu được đạo hàm bậc nhất theo hướng ngang

(G<sub>x</sub>) và hướng dọc (G<sub>y</sub>). Từ đó, chúng ta có thể tìm thấy độ dốc và hướng của cạnh cho mỗi pixel như sau:

- $\text{edge\_gradient}(G) = \sqrt{G_x^2 + G_y^2}$

- $\text{angle}(\theta) = \arctan(G_y/G_x)$

3. **Non-maximum Suppression:** Sau khi nhận được độ lớn và hướng của gradient, quá trình quét toàn bộ hình ảnh được thực hiện để loại bỏ bất kỳ pixel không mong muốn nào có thể không tạo thành cạnh. Đối với điều này, tại mỗi pixel, pixel được kiểm tra xem nó có phải là cực đại cục bộ trong vùng lân cận của nó theo hướng gradient hay không. Chẳng hạn như xét hình bên dưới:

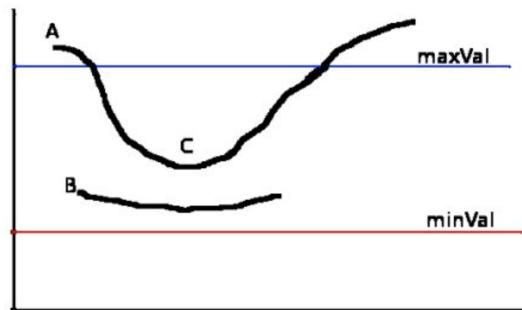


Hình 23: Hình minh họa cơ chế phát hiện cạnh

Theo như hình trên thì điểm A đang nằm trên phương thẳng đứng của một cạnh dọc. Chính vì thế điểm A sẽ kiểm tra điểm B và C đang nằm trên phương của gradient để xem giá trị của điểm A có phải là cực đại cục bộ trong vùng lân cận hay không. Nếu chính xác là cực đại cục bộ trong vùng lân cận thì chuyển sang giai đoạn kế tiếp. Ngược lại thì giá trị của điểm (hoặc pixel đó) sẽ được chỉnh về giá trị là 0 (màu đen).

4. **Hysteresis Thresholding:** Giai đoạn này quyết định đâu là cạnh thực sự là cạnh và đâu không thực sự là cạnh. Đối với điều này, ta cần hai giá trị ngưỡng, minVal và maxVal. Bất kỳ cạnh nào có cường độ gradient lớn hơn maxVal chắc chắn là cạnh và những cạnh dưới minVal chắc chắn không phải là cạnh, do đó, bị loại bỏ. Những giá trị cạnh nằm giữa hai ngưỡng này được phân loại các cạnh hoặc không cạnh dựa trên khả năng kết nối của chúng. Nếu chúng được kết nối với các pixel "chắc chắn", chúng được coi là một phần của các cạnh. Nếu không, chúng cũng bị loại bỏ.

Có thể thấy, cạnh A nằm trên maxVal, do đó được coi là "cạnh chắc chắn". Mặc dù cạnh C nằm dưới maxVal, nhưng nó được nối với cạnh A, vì vậy đó cũng được coi là cạnh hợp lệ và chúng ta có được đường cong/ cạnh đầy đủ. Nhưng cạnh B, mặc dù nó nằm trên minVal và nằm trong cùng vùng với cạnh C, nhưng nó không được kết nối với bất kỳ "cạnh chắc chắn" nào, vì vậy nó sẽ bị loại bỏ. Vì vậy điều rất quan trọng là chúng ta phải chọn minVal và maxVal cho phù hợp để có kết quả chính xác.



Hình 24: Hình ảnh thể hiện việc kiểm tra cạnh dựa trên minVal/ maxVal

Sau các bước tính toán trên, chương trình sẽ trả cho chúng ta kết quả tương tự như hình sau: Khi ứng dụng Grayscale cho hệ thống, ta nhận được hình ảnh như sau:



Hình 25: Kết quả trả về của hệ thống khi sử dụng Canny Edge Detection

#### 2.4.3 Tạo vùng chú ý (Region-of-interest)

Vùng chú ý (Region-of-interest) đối với camera của xe tự hành chỉ là hai làn đường hiện thời trong vùng quan sát của nó. Chúng ta có thể lọc ra các pixel không liên quan bằng cách tạo một vùng đa giác và loại bỏ tất cả các pixel khác không có trong đa giác.

Sau khi lọc và loại bỏ những pixel không nằm trong vùng chú ý, ta thu được kết quả như sau:



Hình 26: Hình ảnh ứng dụng Region-of-interest

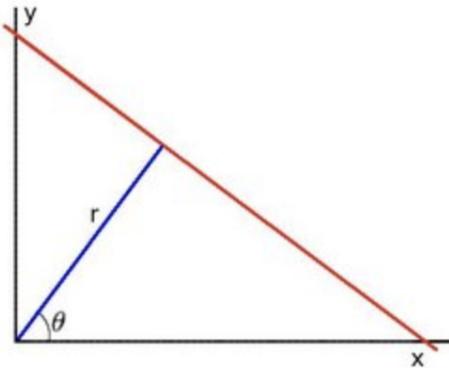
#### 2.4.4 Ứng dụng biến đổi Line Hough (Line Hough Transform)

Sau các bước trên, chúng ta đã tạo được tập hợp các cạnh trên hình, có thể là đường cong, cũng có thể là đường thẳng. Tuy nhiên ở trong việc phát hiện làn đường thì ta chỉ cần sử dụng đường thẳng ở đại đa số cái trường hợp (vì đối khi có trường hợp đường cong). Đây là lúc mà phép biến đổi Hough trở nên hữu dụng.

Như chúng ta đã biết thì đường thẳng trong không gian gian ảnh có thể được thể hiện bởi 2 biến, chẳng hạn như:

- Hệ thống tọa độ Cartesian: Tham số  $(m, b)$
- Hệ thống tọa độ Cực (Polar): Tham số  $(r, \theta)$

Giả sử như trong tọa độ笛卡尔 các ta có đường thẳng  $y = mx + b$  và góc  $\theta$  như hình sau:

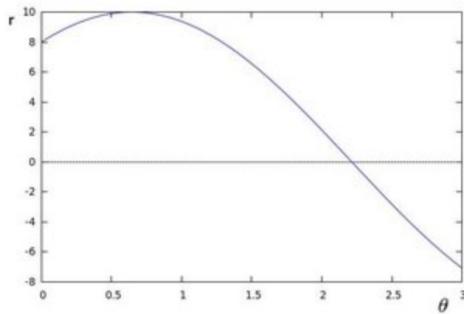


Hình 27: Đường thẳng  $y = mx + b$  trong tọa độ Cartesian

Để dễ thực hiện phép biến đổi Hough (Hough Transform) ta có thể chuyển đường thẳng trên sang toạ độ cực:  $y = (-\frac{\cos \theta}{\sin \theta}x + \frac{r}{\sin \theta})$

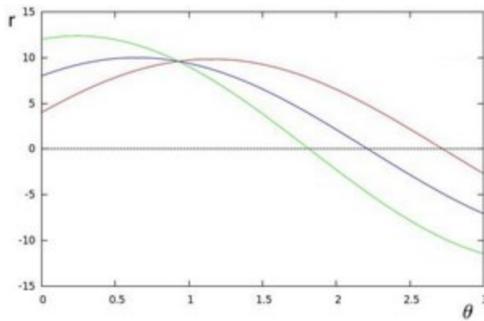
Sắp xếp lại biểu thức trên ta có được  $r = x \cos \theta + y \sin \theta$ . Một cách tổng quát chúng ta sẽ có các tính chất sau:

- Mỗi cặp  $(x_0, y_0)$  cố định, chúng ta có thể định nghĩa một tập hợp các đường thẳng đi qua điểm đó:  $r_\theta = x_0 \cos \theta + y_0 \sin \theta$ . Có nghĩa là mỗi cặp  $(r_\theta, \theta)$  đại diện cho mỗi đường thẳng đi qua điểm  $(x_0, y_0)$ .
- Giả sử chúng ta có  $r > 0$  và  $0 < \theta < 2\pi$  với  $(x_0, y_0) = (8, 6)$ , chúng ta có được một tập các đường đi qua điểm đó được thể hiện trên toạ độ cực như sau (có dạng sinusoid).



Hình 28: Hình vẽ thể hiện tập hợp các đường thẳng trên toạ độ cực đi qua  $(x_0, y_0)$  cố định

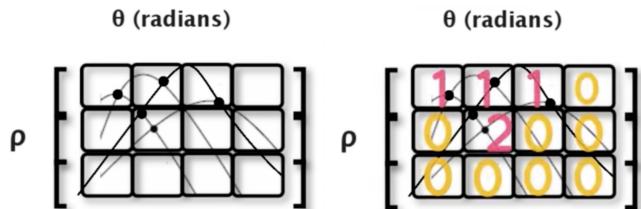
- Tương tự, chúng ta sẽ phác họa tập hợp các đường đi qua  $(x_0, y_0)$  qua toạ độ cực, lần lượt với cặp  $(4, 9)$  và  $(12, 3)$ . Chúng ta có được đồ thị sau. Ta sẽ thu được giao điểm duy nhất  $(\theta, r) = (0.925, 9.6)$ . Tức là điểm này nếu chuyển qua hệ toạ độ Cartesian sẽ là đường thẳng đi qua 3 điểm  $(8, 6), (4, 9), (12, 3)$ .



Hình 29: Hình vẽ thể hiện tập hợp các đường thẳng trên toạ độ cực đi qua 3 cặp  $(x_0, y_0)$  cố định

- Điều đó có nghĩa là, một cách tổng quát chúng ta có thể phát hiện được lằn đường bằng cách tìm kiếm giao điểm của các đồ thị trong toạ độ cực. Các đồ thị giao nhau càng nhiều thì nghĩa là đường thẳng/lằn đường được đại diện bởi nhiều điểm, thì độ chính xác suất cho việc đó là một đường thẳng đúng nghĩa càng cao. Từ đó chúng ta có thể tự quy định

*threshold*, hay gọi là số điểm giao nhau cần thiết để xác định đường đường thẳng, hay trong bài toán này là phương trình của làn đường.



Hình 30: Hình ảnh ứng dụng Hough Transform

- Đó là cách mà phép biến đổi Line Hough thực hiện. Phép biến đổi này sẽ theo dõi số lượng điểm giao nhau giữa các đường cong tại mỗi điểm trong ảnh. Nếu số lượng giao điểm ở có giá trị lớn hơn *threshold*, thì nó sẽ khởi tạo một đường thẳng ứng với  $(\varphi, r_\theta)$ .
- Ứng dụng vào chương trình, ta có được kết quả:



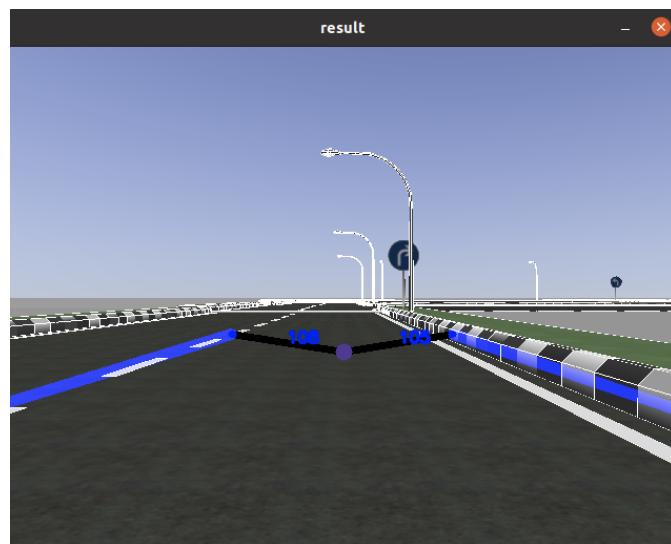
Hình 31: Hình ảnh ứng dụng Hough Transform

#### 2.4.5 Phác họa các dữ liệu đã tính toán và cân chỉnh

Các đường thẳng ta thu được thông qua phép biến đổi Line Hough đôi khi chỉ là những đường thẳng mà không phải là làn đường (chẳng hạn như vạch kẻ ngang phía trước). Đồng thời đôi khi những đường thẳng bị rời rạc nhau mà không nối lại thành 1 làn duy nhất, chính vì thế ta sẽ xử lí/ lọc những tín hiệu nhiễu đó và phác họa lại thông tin làn đường cuối cùng. Đồng thời tính toán khoảng cách giữa làn trái và phải để có thể thuận lợi cho việc điều chỉnh tốc độ rẽ trái, phải, thẳng, lùi trên đường.s

1. Xử lí các đường chấn ngang (không phải làn đường): Bởi vì các đường được phát hiện có phương trình đường thẳng riêng, dựa vào phương trình đường thẳng  $y = mx + b$ , ta có thể loại bỏ đường ngang theo tham số m ( $m = \tan \alpha$ ).
2. Xử lí các đường rời rạc: Dối với những đường thẳng bị tách rời rạc nhưng có xu hướng gần và nối liền nhau, ta sẽ tính trung bình để tạo ra được một đường thẳng mới duy nhất. Và đó sẽ là làn đường được thể hiện như hình.
3. Xử lí khoảng cách trái/ phải: Tiến hành đo khoảng cách giữa một điểm trung tâm của camera với làn trái và phải để xe có thể ước lượng và cân chỉnh tốc độ không bị lấn đường.

Sau đây là hình ảnh đã được xử lí qua các bước trên. Thông tin trên ảnh bao gồm làn đường và khoảng cách trái/ phải

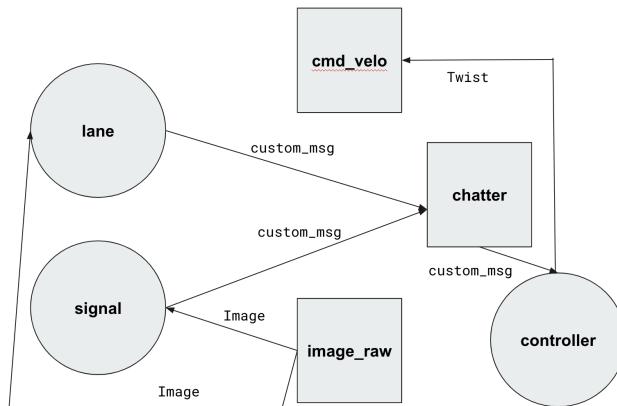


Hình 32: Hình ảnh kết quả trả về cuối cùng

## 2.5 Tổng hợp hệ thống

### 2.5.1 Một số định nghĩa quan trọng

- Node: Node là một tệp thực thi sử dụng ROS để giao tiếp với các Node khác.
- Topic: Topic là các bus được đặt tên mà thông qua đó, các Node trao đổi cái messages. Nhìn chung, các Node không nhận thức rằng chúng đang giao tiếp với ai. Thay vào đó, các Node quan tâm đến dữ liệu subscribe vào Topic liên quan; các Node tạo dữ liệu publish đến các Topic liên quan.
- Message (msg): Như đã nói, các Node giao tiếp với nhau bằng việc publish các message lên Topic. Message là một cấu trúc dữ liệu đơn giản, bao gồm các trường dữ liệu. Các kiểu dữ liệu cơ bản (int, float, bool, etc.) được hỗ trợ, cũng như các array của các kiểu dữ liệu này.



Hình 33: Hình ảnh kết quả trả về cuối cùng

### 2.5.2 Sơ đồ giao tiếp giữa các Node/Topic và chức năng

#### 1. Sơ đồ giao tiếp

#### 2. Giới thiệu về các Node/ Topic:

Trong sơ đồ giao tiếp, hình tròn tượng trưng cho các *node*, và hình vuông tượng trưng cho các *topic*. Chữ cái nằm trên mũi tên đại diện cho kiểu dữ liệu message. Trong đó:

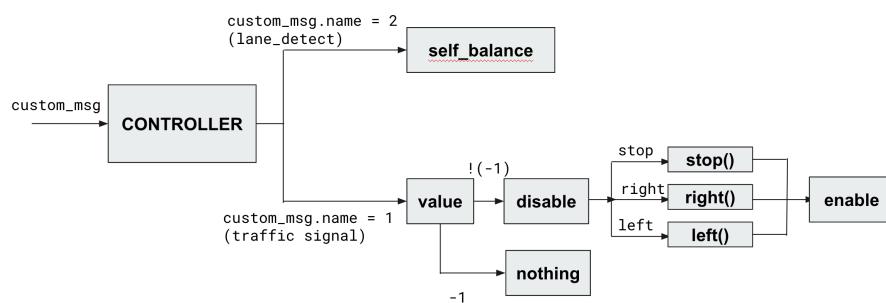
- node "lane": Được dùng để nhận diện làn đường và tính toán khoảng cách so với làn trái, phải. Sau đó sẽ cập nhật giá trị tính toán được cập nhật vào trong message *custom\_msg*, sau đó gửi lên topic "chatter" để lưu trữ.
- node "signal": Được dùng để phát hiện và nhận diện biển báo. Sau đó giá trị nhận biển báo sẽ được cập nhật vào trong message *custom\_msg*, và gửi lên topic "chatter" để lưu trữ.
- node "controller": Lấy giá trị *custom\_msg* từ topic "chatter" rồi xử lý, sau đó gửi tốc độ (định dạng message là *Twist*) lên topic *cmd\_velo* để lưu trữ.
- topic "cmd\_velo": Là nơi lưu trữ các message có dạng *Twist*.
- topic "image\_raw": Là nơi lưu trữ các message có dạng *Image*.
- topic "chatter": Là nơi lưu trữ các message có dạng là *custom\_msg*.
- message "custom\_msg": Là tệp thông tin truyền nhân giữa các node mà người dùng tự định nghĩa. Mục tiếp theo sẽ giải thích chi tiết cụ thể message mà nhóm định nghĩa
- message "Image": Là tệp thông tin truyền nhận giữa các node được xây dựng sẵn để truyền nhận dữ liệu ảnh.
- message "Twist": Là tệp thông tin truyền nhận giữa các node được xây dựng sẵn để truyền nhận dữ liệu tốc độ.

3. **Định dạng phương thức message trao đổi dữ liệu:** Để dễ dàng hơn trong việc truyền gửi thông tin phù hợp với dữ liệu đang có. Ta có thể tự định nghĩa ra gói message cho riêng mình. Ở đây, để phù hợp với bài toán đặt ra thì nhóm đã đề ra cấu trúc cho gói "*custom\_msg*" như sau:  $custom\_msg = \{int64 name, int64 num1, int64 num2\}$ . Trong đó:

- (a) Tín hiệu gửi từ node "signal" sẽ được gán name = 1, và num1 = -1, 0, 1, 2, num2 = -1. Trong đó ta không sử dụng num2. Và ở num1 thì -1 ứng với không phát hiện được gì, 0 tương ứng stop, 1 tương ứng left và 2 tương ứng right.
- (b) Tín hiệu gửi từ node "lane" sẽ được gán name = 2, num1 là khoảng cách so với làn trái và num2 là khoảng cách so với làn phải. Khoảng cách có giá trị là -1 khi không phát hiện được làn đường.

### 2.5.3 Sơ đồ xử lí của khối tổng hợp (Controller)

#### 1. Sơ đồ xử lí



Hình 34: Hình ảnh kết quả trả về cuối cùng

#### 2. Giải thích các khối

##### • self\_balance():

- Là khối giúp xe chuyển động thẳng, đồng thời tự cân bằng khoảng cách giữa 2 làn đường.
- Cách hoạt động: **self\_balance()** nhận 2 tham số *distance\_left* và *distance\_right* làm thông số chính để điều khiển xe. Các tham số *distance\_left* và *distance\_right* được lấy từ node *lane\_detect* qua *custom\_msg*. Trường hợp cả hai tham số truyền vào đều lớn hơn 0 (tức là nhận diện được cả 2 làn đường), nếu khoảng cách giữa xe và làn đường bất kỳ có giá trị nhỏ hơn 80px, xe sẽ tự động rẽ hướng ngược lại 1 góc nhỏ  $\approx 0.1rad$ , liên tục điều chỉnh cho tới khi thỏa khoảng cách giữa xe và cả 2 làn lớn hơn 80px. Trường hợp có một tham số bé hơn 0 (tức không xác định được làn đường do một số yếu tố ngoại cảnh), nếu khoảng cách giữa xe và làn đường xác định được lớn hơn 130px, xe sẽ tự động rẽ hướng ngược lại 1 góc nhỏ  $\approx 0.1rad$ , liên tục điều chỉnh cho tới khi thỏa khoảng cách giữa xe và làn đó nhỏ hơn 130px

##### • turn\_right():

- Là khối giúp xe chuyển động rẽ phải.
- Cách hoạt động: tạo bộ đếm thời gian, với tốc độ khoảng 0.2px/s, tốc độ góc 0.25rad/s, nhóm xác định thời gian cần để xe rẽ phải là vào khoảng 7.5s.

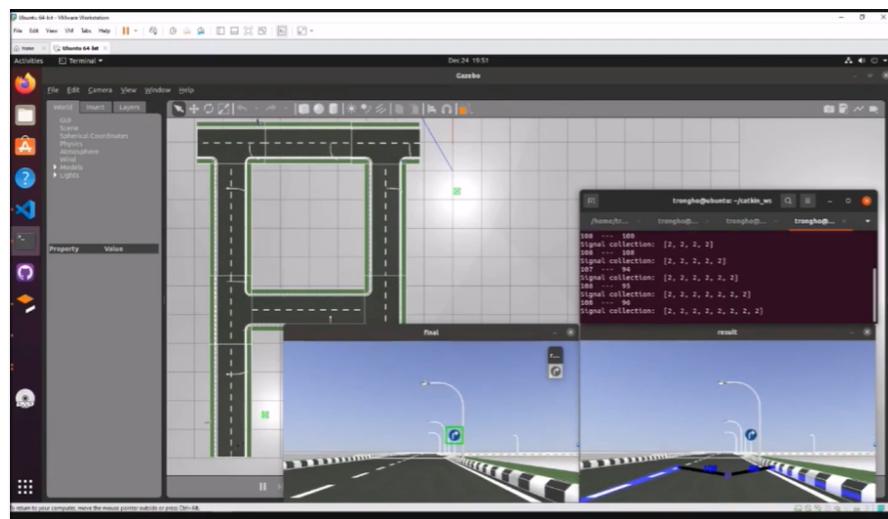
##### • turn\_left():

- Là khối giúp xe chuyển động rẽ trái.

- Cách hoạt động: tương tự như với rẽ phải, có điều chỉnh phù hợp để xe đến đúng vị trí.
- **stop():**
  - Là khởi động chuyển động xe.

## 3 Tổng kết

### 3.1 Video demo project của nhóm



Hình 35: Video demo (bấm vào [Link YouTube](#) sau)

### 3.2 Thuận lợi và khó khăn

#### 3.2.1 Thuận lợi

- Đây là một đề tài hay, đem lại nguồn động lực tìm tòi cho sinh viên đặc biệt là có nguồn kinh phí hạn hẹp do có thể chạy trên mô phỏng.
- Có sẵn nguồn tài liệu tham khảo trên mạng để tra cứu nếu cần thiết. Đặc biệt là tài liệu về Deep Learning và Computer Vision.

#### 3.2.2 Khó khăn

- Các thành viên đều lần đầu tiếp xúc về lĩnh vực Computer Vision và Deep Learning nên thời gian đầu tìm hiểu rất khó khăn (Kể cả đọc hiểu code lẫn lý thuyết). Đồng thời cũng khá mới lạ với hệ điều hành ROS.
  - ⇒ Xem các video để duy trì nguồn cảm hứng về mặt tinh thần cũng như liên tục bổ sung về mặt kiến thức.



- Hạn chế về thiết bị phần cứng. Cụ thể là việc chạy mô phỏng khiến máy trở nên rất nóng, nên khá mất thời gian cho việc chờ máy dịu lại rồi mới có thể làm tiếp để tránh tình trạng giật lag.
  - ⇒ **Sử dụng quạt đứng để giúp tản nhiệt cho laptop**
- Nguồn tài liệu vì quá nhiều nên dễ bị loãng kiến thức.
  - ⇒ **Chắt lọc từ những nguồn uy tín**
- Đề tài quá rộng, còn thời gian hạn chế.
  - ⇒ **Giảm lại phạm vi đề tài, giảm số lượng biến báo xuống để có thể tăng độ chính xác và độ thành công khi thực hiện chương trình.**
- Gặp phải một số lỗi cài đặt và khá mất thời gian để sửa (vì yêu cầu độ tương thích hệ điều hành, cũng như là thiết bị).
  - ⇒ **Chia nhau tìm hiểu và lưu lại những lỗi hay gặp để giảm thiểu thời gian mắc lỗi và sửa chữa.**



## 4 Phân công công việc

### 4.1 Bảng phân công công việc

Họ và tên	Nhiệm vụ
1. Nguyễn Trọng Nhân	<ul style="list-style-type: none"><li>Phát triển module phát hiện làn đường.</li><li>Tạo ra gói message mới (custom_msg) để giao tiếp.</li><li>Thiết kế mô hình giao tiếp và tổng hợp các module.</li><li>Viết báo cáo.</li><li>Soạn slide.</li></ul>
2. Lê Hoàng Minh Tú	<ul style="list-style-type: none"><li>Phát triển module phát hiện và nhận diện biển báo.</li><li>Thiết kế map và biển báo giao thông trong Gazebo Simulation.</li><li>Viết báo cáo.</li><li>Soạn slide.</li></ul>
3. Hồ Hữu Trọng	<ul style="list-style-type: none"><li>Hiện thực khối chức năng tổng hợp các Module thành phần (Controller)</li><li>Quay video demo</li><li>Viết báo cáo</li><li>Soạn slide.</li></ul>

#### 4.1.1 Link Trello quản lý công việc

Link: [Trello Project Management](#)

#### 4.1.2 Link Github Project:

Link: [Project Github](#)



## 5 Tài liệu tham khảo

### Tài liệu

- [1] GazeboSim.org “<http://gazebosim.org/tutorials>”, *Gazebo tutorials*
- [2] 3dWarehouse.sketchup.com “<https://3dwarehouse.sketchup.com/>”, *Sketch Up open source*
- [3] wiki.ros.org “<http://wiki.ros.org/ROS/Tutorials>”, *ROS tutorials*
- [4] wiki.ros.org ”[http://wiki.ros.org/cv\\_bridge/Tutorials](http://wiki.ros.org/cv_bridge/Tutorials)”, *CVBridge in ROS*
- [5] Valentyn Sichkar ”<https://www.kaggle.com/valentynsichkar/traffic-sign-dataset>”, *Traffic Signs Dataset*
- [6] Valentyn Sichkar ”<https://www.kaggle.com/valentynsichkar/traffic-signs-cnn-model>”, *Traffic Signs CNN model* Cập nhật lần cuối: 14/07/2021
- [7] vohungvi ”<https://thigiacmaytinh.com/ly-thuyet-phan-nguong-anh>”, *Lý thuyết về phân ngưỡng ảnh (threshold)* Cập nhật lần cuối: 12/07/2015
- [8] TopDev ”<https://topdev.vn/thuat-toan-convolutional-neural-network/>”, *Thuật toán convolutional neural network*
- [9] opencv.org ”[https://docs.opencv.org/3.4/da/d22/tutorial\\_py\\_canny](https://docs.opencv.org/3.4/da/d22/tutorial_py_canny)”, *Canny Edge Detection*
- [10] opencv.org ”[https://docs.opencv.org/3.4/d9/db0/tutorial\\_hough\\_lines](https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines)”, *Hough Lines*
- [11] <https://www.kdnuggets.com/2017/07/road-lane-line-detection-using-computer-vision-models>”, *Road Line Detection*