

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

XE TỰ HÀNH VỚI ROS VÀ GIẢ LẬP GAZEBO

GV ra đề và hướng dẫn: Trần Thanh Bình

SV thực hiện: Nguyễn Quốc Trọng - 1915676
Nguyễn Anh Văn - 1915886
Nguyễn Quốc Hùng - 1913610
Lê Mậu Phúc Khang - 1913693

Tp. Hồ Chí Minh, Tháng 11/2021

LỜI MỞ ĐẦU

Ngày nay, nền khoa học kỹ thuật đang phát triển như vũ bão, rất nhiều các phát minh, sản phẩm công nghệ mới ra đời. Các robot tự động đang là xu hướng công nghệ tương lai vì sự tiện dụng, khả năng hoạt động chính xác trong các điều kiện khắc nghiệt với con người. Một yêu cầu cơ bản của các robot tự động là khả năng hoạt động theo chức năng mong muốn mà không cần sự can thiệp của con người trong quá trình làm việc. Các đề tài về giao thông đang được chú ý đến trong các diễn đàn về công nghệ. Trong đó thông tin về các hệ thống như : hệ thống xe hỗ trợ hầm phanh tự động, hệ thống nhận biết vật cản cho xe, các hệ thống về nhận diện đường đi cũng như là điều khiển tự động lái của xe, các đề tài này hiện tại đang được chú trọng phát triển một cách mạnh mẽ. Bên cạnh đó xe tự hành cũng là một đề tài nổi trội trong lĩnh vực này.

Xe tự hành là một hệ thống tự hoạt động không cần đến sự điều khiển của con người. Xe tự hành có thể phân tích xung quanh mà không cần đến bất kỳ tương tác nào của con người và đưa ra quyết định mà không cần đến sự can thiệp của con người trong quá trình đấy. Một số cảm biến được sử dụng trên xe để xác định đường đi và tín hiệu của môi trường xung quanh. Xe tự hành giảm được chi phí do ít hao phí nhiên liệu hơn, tăng tính an toàn, tăng tính di động, tăng sự hài lòng của khách hàng và đó là lý do tại sao nó có nhiều lợi thế hơn so với xe truyền thống. Lợi ích lớn nhất của việc sử dụng xe tự lái là ít tai nạn giao thông hơn đáng kể. Hơn 90% tất cả các vụ tai nạn là do lỗi của con người ở một mức độ nào đó, bao gồm mất tập trung, lái xe kém hoặc ra quyết định kém. Với việc ô tô tự lái đưa ra quyết định và giao tiếp với nhau, số vụ tai nạn sẽ giảm.

Chúng em xin cảm ơn sự hướng dẫn, chia sẻ tận tình của Thầy Trần Thanh Bình giúp chúng em hoàn thành đề tài này. Trong thời lượng hạn chế của môn học, với kỹ năng và kinh nghiệm ít ỏi, chúng em không thể tránh khỏi những sai sót, rất mong được sự góp ý của thầy và các bạn để đề tài được hoàn thiện hơn

MỤC LỤC

LỜI MỞ ĐẦU	1
DANH SÁCH HÌNH VẼ	3
1 Giới thiệu chung	4
2 Mục tiêu và nhiệm vụ	4
2.1 Mục tiêu của đề tài	4
2.2 Nhiệm vụ của đề tài	4
3 Cơ sở lý thuyết	5
3.1 Xe tự hành là gì?	5
3.2 Tổng quan về ROS	6
3.3 Xử lý hình ảnh với OpenCV	8
4 Hiện thực yêu cầu	8
4.1 Ý tưởng thực hiện	8
4.2 Mô tả hệ thống	8
4.3 Hiện thực mô hình trong Gazebo	9
4.3.1 Bản đồ hệ thống	9
4.3.2 Phương tiện tự hành	10
4.4 Hiện thực nhận diện biển báo	11
4.4.1 Blob Detection sử dụng OpenCV	12
4.4.2 Kiểm tra hình ảnh thông qua model	14
4.5 Hiện thực phát hiện làn đường	17
4.6 Hiện thực điều khiển xe	26
5 Kết quả và đánh giá	29
5.1 Kết quả	29
5.2 Ưu điểm	30
5.3 Hạn chế	30
5.4 Phương hướng phát triển	30

Danh sách hình vẽ

1	Phân loại xe tự hành	5
2	Cấu tạo xe tự hành	6
3	Thành phần ROS	7
4	Kiến trúc hệ thống	9
5	Bản đồ hoàn thiện	10
6	Các loại biển báo	10
7	Mô hình xe trong Gazebo	11
8	Mô hình xe trong Rviz	11
9	Mô hình xe dạng TF	11
10	Các bước nhận diện biển báo	12
11	Phát hiện các đốm màu	12
12	Nhận diện biển báo rẽ phải	17
13	Nhận diện biển báo dừng lại	17
14	Hình ảnh ban đầu	18
15	Lọc các cạnh trong hình	19
16	Cắt hình	20
17	Line segments	21
18	Vẽ 2 làn đường	24
19	Vẽ thêm đường xác định góc	25
20	Kết quả mô phỏng	29
21	Sơ đồ kết nối các node	29



1 Giới thiệu chung

Robot đã và đang xuất hiện trong cuộc sống của chúng ta từ lâu và ngày càng trở thành một phần không thể thiếu trong cuộc sống hiện đại. Chúng đã góp phần mình vào công cuộc lao động, chính robot đang làm nên một cuộc cách mạng về lao động,khoa học, và đang phục vụ đắc lực cho các ngành khoa học như: Khoa học quân sự,khoa học giáo dục, các ngành dịch vụ, giải trí. Do việc phải học online, các thành viên không gặp nhau được trực tiếp nền trong đề tài này, nhóm em sẽ thực hiện thiết kế một robot tự hành bám đường trong mô phỏng Gazebo.

2 Mục tiêu và nhiệm vụ

2.1 Mục tiêu của đề tài

Mục tiêu của nhóm khi thực hiện đề tài là trước hết là muôn phát huy tiềm năng của chủ đề robot trong thời đại hiện nay. Đồng thời tạo ra một sản phẩm bám sát theo công nghệ hiện tại.

Ngoài ra, quá trình làm đề tài là khoảng thời gian quý báu để nhóm thực hiện đề tài và có thể tự kiểm tra lại những kiến thức đã học ở trường từ thầy cô. Đồng thời có thể phát huy sự sáng tạo của mình cho các ứng dụng tự động trong mảng robot và hơn thế nữa có khả năng giải quyết những vấn đề và yêu cầu mà ý tưởng ban đầu đã đặt ra.

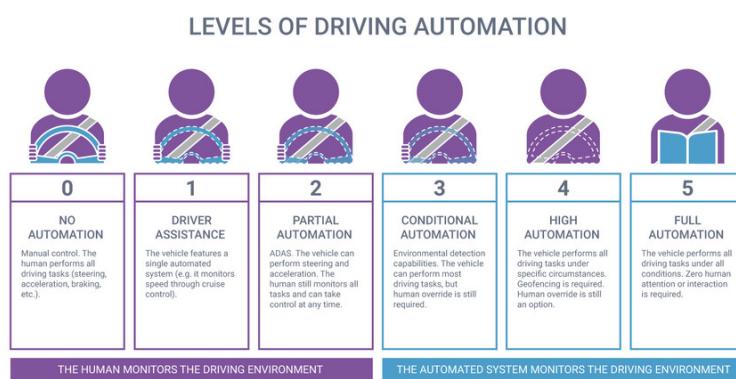
2.2 Nhiệm vụ của đề tài

- Sử dụng phần mềm giả lập **Gazebo** và **ROS** để hiện thực hệ thống
- Hiện thực phát hiện làn đường cho xe tự hành
- Hiện thực phát hiện biển báo
- Hiện thực điều khiển xe di chuyển tự động theo đúng các biển báo và làn đường phát hiện được

3 Cơ sở lý thuyết

3.1 Xe tự hành là gì?

- Xe tự hành là phương tiện giao thông có khả năng tự điều khiển nhờ hệ thống phần cứng hỗ trợ mà không có hoặc có ít sự can thiệp của con người.
- Phân loại xe tự hành: Theo The Society of Automotive Engineers (SAE), tùy thuộc vào mức độ can thiệp của con người xe tự hành được phân chia thành các loại như hình dưới đây.



Hình 1: Phân loại xe tự hành

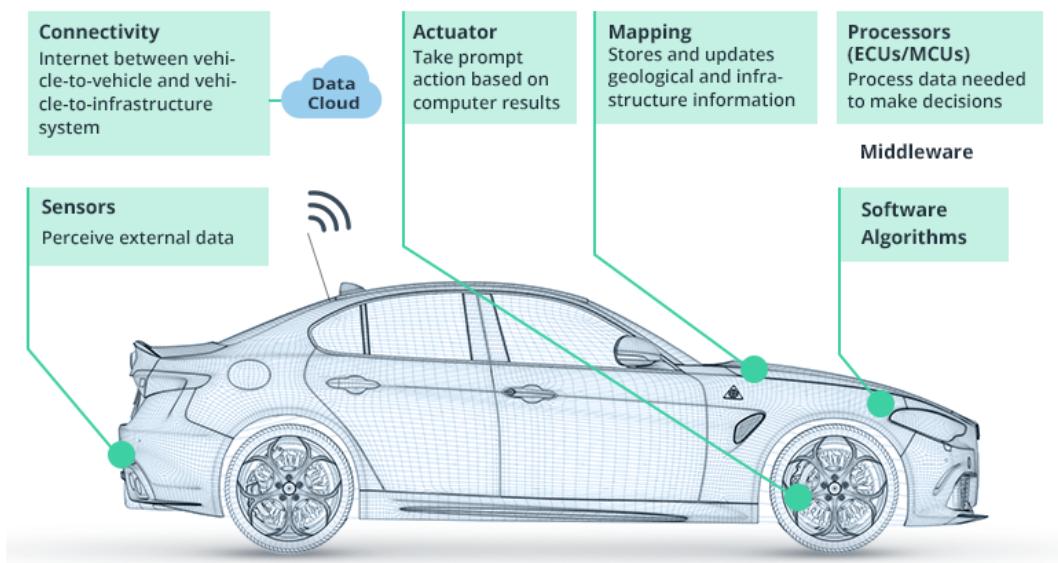
- Cấu tạo của xe tự hành:

- Bộ xử lý trung tâm
- Hệ thống máy ảnh và cảm biến
- Hệ thống thiết bị truyền động
- Các thiết bị hỗ trợ khác như gps v.v

- Phương thức hoạt động xe tự hành:

Xe tự hành hoạt động dựa trên hệ thống các máy ảnh, cảm biến và bộ xử lý tinh vi, mạnh mẽ.

Bản đồ về môi trường xung quanh luôn được tạo và duy trì trong suốt quá trình hoạt động. Cảm biến thăm dò giúp phát hiện các phương tiện xung quanh. Máy ảnh ghi lại hình ảnh biển báo, đèn giao thông, các phương tiện trên đường, người đi bộ cùng với các vật thể khác. Cảm biến nắp phát hiện và xác định phạm vi ánh sáng hỗ trợ tính toán, phát hiện mép đường cùng với hệ thống vạch kẻ đường v.v. Từ dữ liệu thu được từ hệ thống trên, bộ xử lý trung tâm tiến hành tính toán, xử lý dữ liệu đầu vào và vạch ra đường đi và gửi hướng dẫn đến bộ truyền động của ô



Hình 2: Cấu tạo xe tự hành

tô, bộ phận này sẽ kiểm soát gia tốc, phanh và đánh lái.

Các quy tắc được mã hóa cứng, thuật toán tránh chướng ngại vật, mô hình dự đoán và nhận dạng đối tượng giúp phần mềm tuân theo các quy tắc giao thông và điều hướng chướng ngại vật.

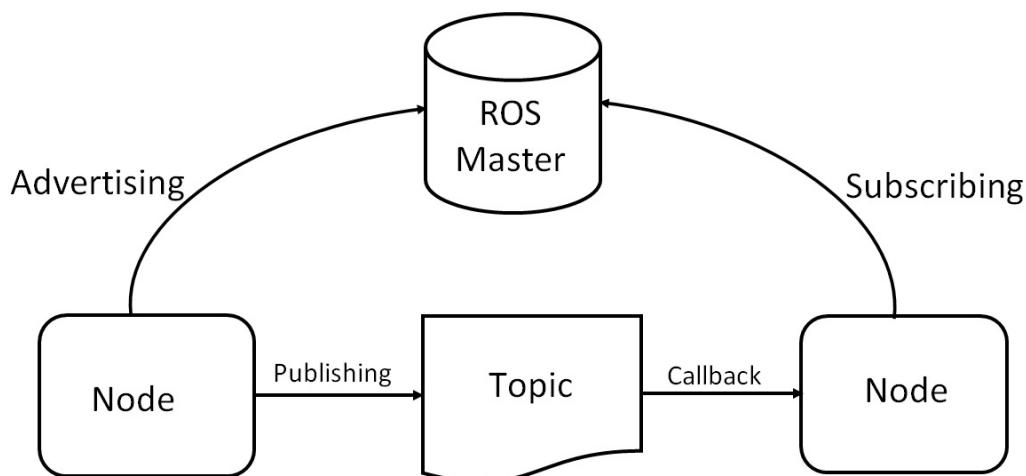
3.2 Tổng quan về ROS

Robot operating system (ROS) là một hệ thống phần mềm có tính linh hoạt và chuyên dụng cao dùng để lập trình và điều khiển robot. ROS bao gồm các thư viện, công cụ hỗ trợ lập trình, công cụ đồ họa, các công cụ hỗ trợ giao tiếp điều khiển trực tiếp với phần cứng cũng như các thư viện hỗ trợ việc lấy dữ liệu từ các cảm biến và các thuật toán phổ biến trong lập trình điều khiển robot.

ROS giúp đơn giản và tiết kiệm thời gian việc lập trình các hành vi phức tạp của robot. ROS như là một mạng lưới điểm (node) được kết nối bởi nhiều node mà mỗi node có nhiệm vụ riêng tương ứng với các thành phần cấu tạo nên robot. Mỗi node riêng có thể được tùy chỉnh phát triển và lập trình theo ý tưởng của người phát triển mà không cần quan tâm đến việc phải thống nhất sử dụng cùng một ngôn ngữ lập trình, điều đó có nghĩa là node này bạn phát triển trên ngôn ngữ C++ nhưng node khác bạn có thể phát triển trên ngôn ngữ Python.

ROS bao gồm các cấu thành bởi các thành phần như sau:

- Nodes: là đơn vị cơ bản nhằm hỗ trợ giao tiếp với các thành phần cấu thành nên robot. Vì dụ như một con robot thường có các node như Laser scanner, Camera,



Hình 3: Thành phần ROS

Node gửi vận tốc đến thiết bị điều khiển động cơ. Các nodes này có thể giao tiếp và tương tác với nhau qua Master.

- Master: đóng vai trò kết nối các node với nhau. Do đó, master luôn được khởi động đầu tiên bằng lệnh roscore sau đó thì bạn có thể gọi bất kỳ các node nào trong hệ thống. Sau khi gọi xong, các node có thể kết nối và tương tác với nhau.
- Parameter Server: Là một cấu trúc nhiều tham số có thể truy cập trong lúc chạy ROS. Các node sử dụng cấu trúc này nhằm lưu trữ và truy xuất các thông số trong thời gian chạy. Do nó không có hiệu suất cao nên thường dùng với kiểu dữ liệu tĩnh, chẳng hạn như các thông số cấu hình, thời gian hệ thống.
- Message: Đây là cấu trúc dữ liệu được các node sử dụng để trao đổi với nhau tương tự như các kiểu dữ liệu double, int trong các ngôn ngữ lập trình. Các node tương tác với nhau bằng cách send và receive ROS message.
- Topics: là phương pháp giao tiếp trao đổi dữ liệu giữa hai node, nó bao gồm nhiều cấp bậc thông tin mà chúng có thể giao tiếp thông qua ROS message. Hai phương thức trong topic bao gồm publish và subscribe.
- Services: Là một giao tiếp trao đổi dữ liệu/ thông tin giữa hai node thông qua phương thức request và response. Thường được áp dụng trong trường hợp việc thực hiện một lệnh cần nhiều thời gian xử lý nên dữ liệu tính toán được lưu ở server và sẽ dùng khi cần xử lý.
- Bags: là một định dạng tệp trong ROS dùng để lưu trữ dữ liệu message với phần mở rộng là .bag có vai trò quan trọng và có thể được xử lý, phân tích bởi các công cụ mô phỏng trong ROS như Rviz.



3.3 Xử lý hình ảnh với OpenCV

OpenCV là tên viết tắt của Open Source Computer Vision Library – có thể được hiểu là một thư viện mã nguồn mở cho máy tính. Cụ thể hơn OpenCV là kho lưu trữ các mã nguồn mở được dùng để xử lý hình ảnh, phát triển các ứng dụng đồ họa trong thời gian thực...

OpenCV cho phép cải thiện tốc độ của CPU khi thực hiện các hoạt động real time. Nó còn cung cấp một số lượng lớn các mã xử lý phục vụ cho quy trình của thị giác máy tính hay các learning machine khác.

OpenCV được sử dụng cho đa dạng nhiều mục đích và ứng dụng khác nhau bao gồm:

- Hình ảnh street view
- Kiểm tra và giám sát tự động
- Robot và xe hơi tự lái
- Phân tích hình ảnh y học

4 Hiện thực yêu cầu

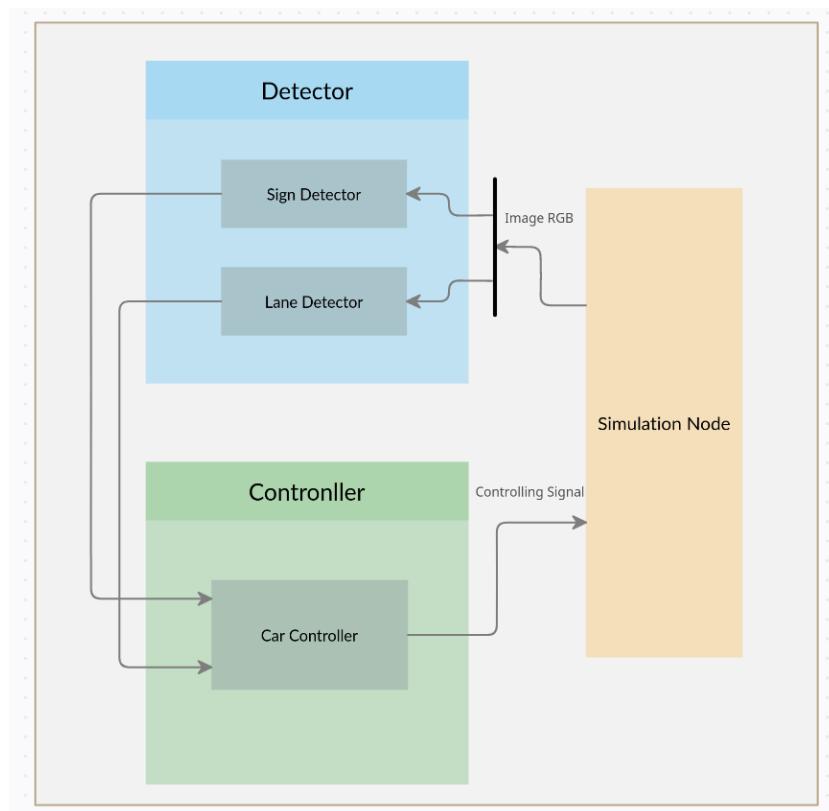
4.1 Ý tưởng thực hiện

Đề tài yêu cầu thiết kế và triển khai một hệ thống điều khiển cho xe tại một mô phỏng tương đương với thế giới thực. Trong mô phỏng này, mô hình xe của nhóm chúng em sẽ được trang bị một camera để thu các hình ảnh phía trước của mình và dựa vào đó sẽ đưa ra các quyết định điều khiển cho xe (góc lái, tốc độ). Để làm được điều này nhóm sẽ dùng hai 2 giải thuật xử lý hình ảnh để phát hiện làn đường và nhận diện các biển báo.

- Hệ thống nhận biết vạch kẻ đường đưa ra quyết định điều khiển xe chạy đúng theo làn đường nhận biết được.
- Hệ thống nhận biết cách biển báo giao thông bao gồm: đi thẳng, rẽ trái, rẽ phải và dừng lại. Đồng thời đưa ra các quyết định điều khiển đối với các biển báo trên một cách đúng đắn.

4.2 Mô tả hệ thống

Thông qua việc tìm hiểu các công cụ giả lập cũng như đề xuất từ phía giảng viên hướng dẫn, nhóm đã chọn Gazebo là công cụ để mô phỏng. Giả lập này sẽ sử dụng môi



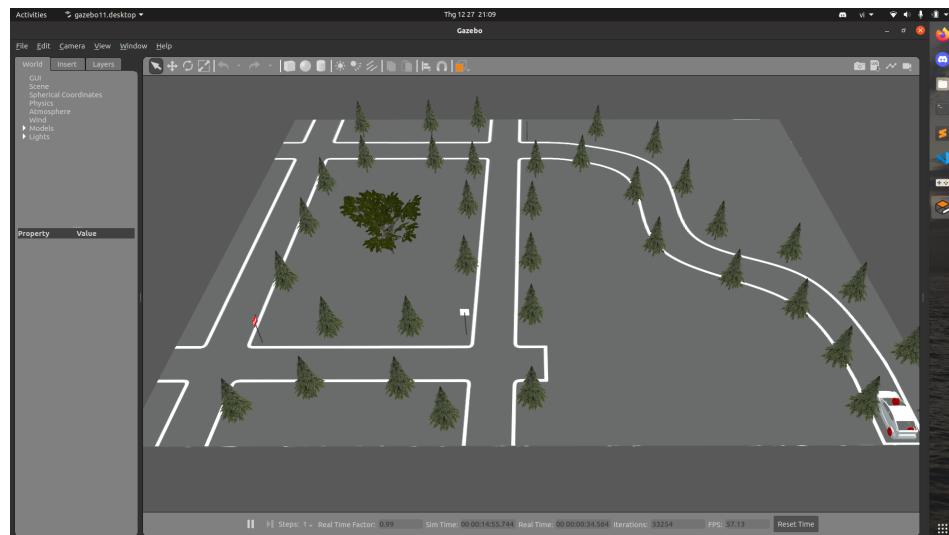
Hình 4: Kiến trúc hệ thống

trường ROS để trao đổi thông tin với phần mềm điều khiển xe. Giả lập Gazebo và phần mềm điều khiển đóng vai trò là các "node" trong hệ thống ROS, giao tiếp với nhau thông qua cơ chế publisher - subscriber. Phần mềm giả lập sẽ liên tục truyền hình ảnh thu được phía trước mũi xe vào hệ thống ROS, đồng thời cũng liên tục nhận lại và thực thi các lệnh điều khiển (bao gồm góc lái và tốc độ). Hình dưới mô tả kiến trúc chung của hệ thống được chúng tôi thiết kế được: 1 mạng lane detector (phát hiện làn đường) và một mạng sign detector (phát hiện biển báo) từ hình ảnh lấy được từ mô phỏng. Trình điều khiển sẽ dựa vào kết quả xử lý hình ảnh để đưa ra lệnh điều khiển cho trình mô phỏng.

4.3 Hiện thực mô hình trong Gazebo

4.3.1 Bản đồ hệ thống

Trong đồ án này, nhóm đã xây dựng một bản đồ đơn giản để thể hiện lại làn đường và các loại biển báo cơ bản (rẽ trái, rẽ phải, đi thẳng và dừng lại). Bản đồ được nhóm vẽ với công cụ paint và có thể được thay thế dễ dàng, biển báo được sử dụng có thể thêm bớt nhanh chóng bằng việc sử dụng các model biển báo đã tạo. Kết quả được hiển thị bên dưới



Hình 5: Bản đồ hoàn thiện

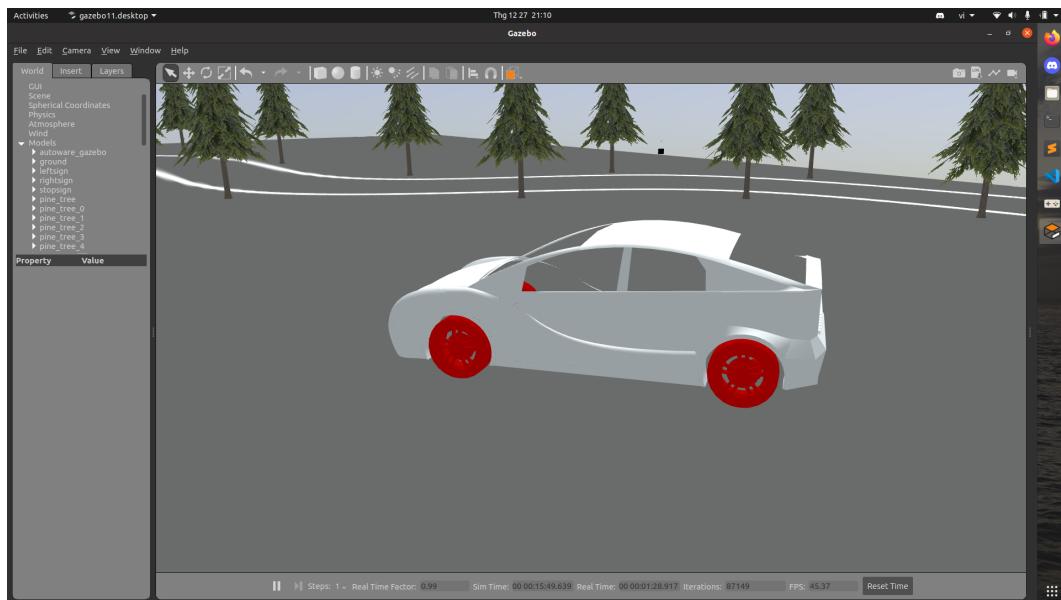


Hình 6: Các loại biển báo

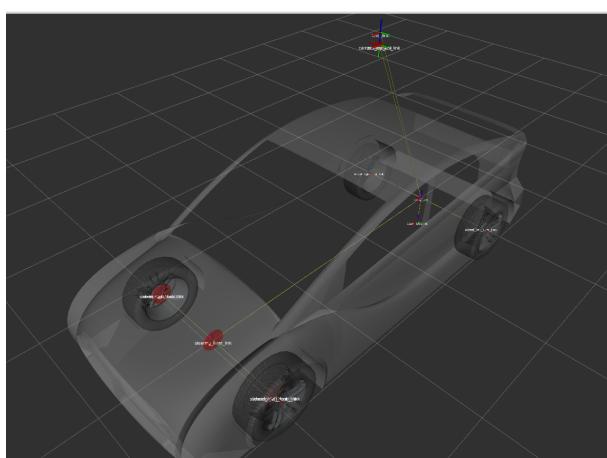
4.3.2 Phương tiện tự hành

Một chiếc xe mô phỏng 4 bánh (hình 7) được trang bị camera phía trước xe cho phép nó ghi nhận lại thông tin hình ảnh phía trước của xe. Đồng thời, tạo một kết nối giữa các bánh xe lại với nhau để điều khiển tốc độ; một kết nối giữa 2 bánh trước để điều khiển về góc lái.

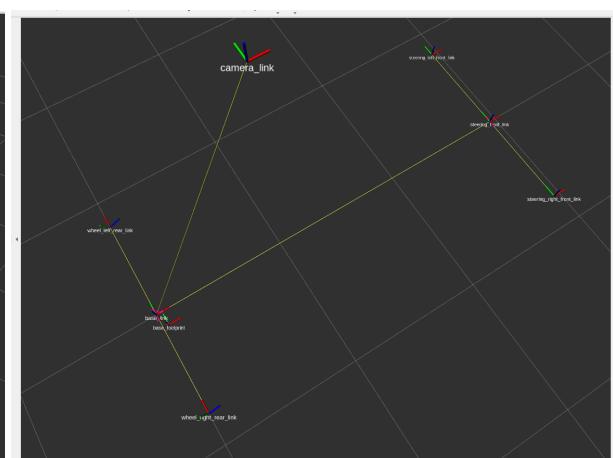
Ở đây, nhóm đã dùng lại mô hình xe tự hành của Yukihiro Saito tại [link](#)



Hình 7: Mô hình xe trong Gazebo



Hình 8: Mô hình xe trong Rviz



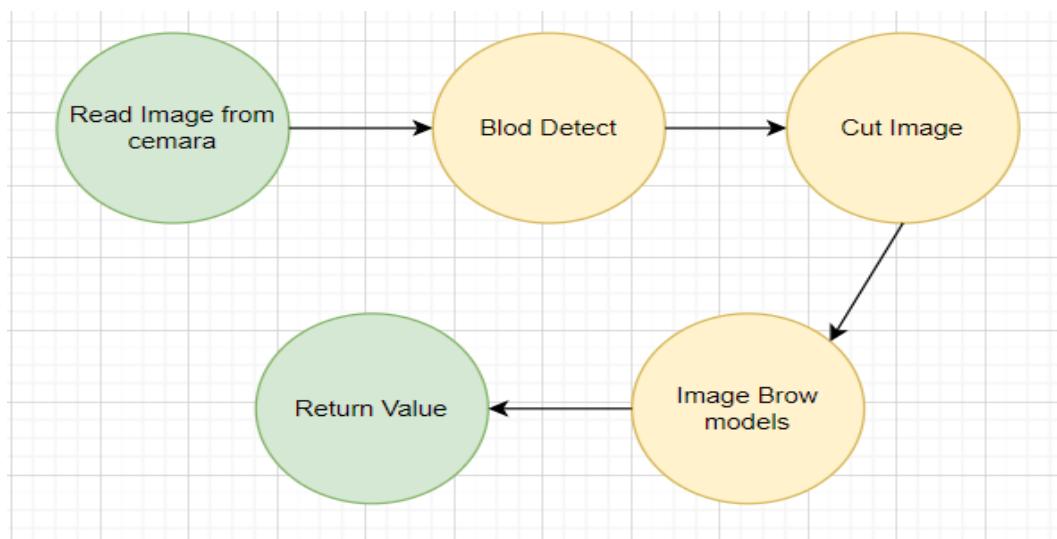
Hình 9: Mô hình xe dạng TF

4.4 Hiện thực nhận diện biển báo

Như chúng ta đã biết có rất nhiều loại biển báo giao thông khác nhau như biển báo giới hạn tốc độ, biển cấm đi vào, tín hiệu giao thông, rẽ trái hoặc phải, trẻ em băng qua đường, cấm xe nặng vượt qua, v.v. Nhận diện biển báo giao thông là quá trình nhận biết nội dung biển báo và xác định biển báo đó thuộc loại nào.

Trong project này chúng tôi đã sử dụng một mô hình deep neural network có thể phân loại các biển báo giao thông có trong hình ảnh thành các loại khác nhau được train từ tập dữ liệu công khai có sẵn tại Kaggle. Tải xuống tập dữ liệu [Traffic Signs Dataset](#). Với mô hình này, chúng ta có thể đọc và hiểu các biển báo giao thông, một nhiệm vụ rất quan trọng đối với tất cả các phương tiện tự hành.

Để có thể nhận diện biển báo với hình ảnh được đọc từ camera của xe mô phỏng, nhóm đã thực hiện theo các bước dưới đây:

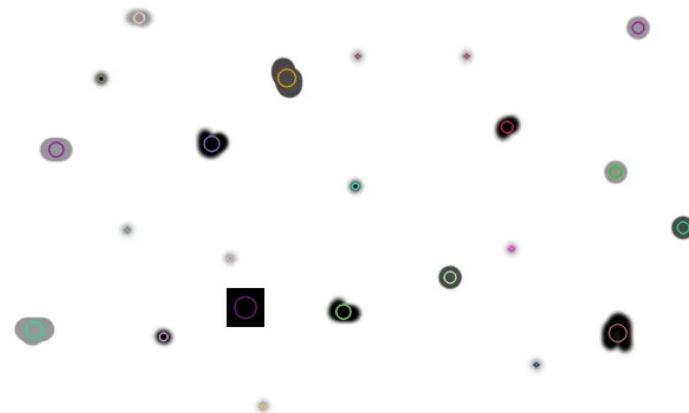


Hình 10: Các bước nhận diện biển báo

4.4.1 Blob Detection sử dụng OpenCV

- **Blob là gì?**

Blob là một nhóm các pixel được kết nối trong một hình ảnh có chung một số thuộc tính (ví dụ: giá trị thang độ xám). Trong hình bên dưới, các vùng kết nối tối là các đốm màu và mục tiêu của việc phát hiện đốm màu là xác định và đánh dấu các vùng này



Hình 11: Phát hiện các đốm màu

- **Cách hoạt động của Blob detection**

Thuật toán này được điều khiển bởi các tham số bên dưới và có các bước sau.



- **Thresholding:** Chuyển đổi các hình ảnh nguồn thành một số hình ảnh nhị phân bằng cách tạo ngưỡng cho hình ảnh nguồn với các ngưỡng bắt đầu từ **minThreshold**. Các ngưỡng này được tăng dần theo **thresholdStep** cho đến khi **maxThreshold**.
- **Grouping:** Trong mỗi ảnh nhị phân, các pixel trắng được kết nối được nhóm lại với nhau.
- **Merging:** Tâm của các đốm màu nhị phân trong ảnh nhị phân được tính toán và các đốm màu nằm gần **minDistBetweenBlobs** sẽ được hợp nhất.
- **Center & Radius Calculation:** Tâm và bán kính của các đốm màu hợp nhất mới được tính toán sẽ được trả về.
- **Tinh chỉnh các tham số:** Với từng tình huống cụ thể, các tham số như **Circularity** (dáng tròn), **Convexity** (tính lồi), **Inertia Ratio** (tỉ lệ quán tính) có thể được tinh chỉnh để lọc loại đốm màu mà chúng ta muốn.

Để hiện thực Blob detector, sau khi đã kiểm tra và tinh chỉnh các thông số, dưới đây sẽ là đoạn mã code cho phần này, giá trị trả về sẽ được sử dụng trong phần tiếp theo:

```
1 def detect_keypoints(image):  
2     # Set our filtering parameters  
3     # Initialize parameter setting using cv2.SimpleBlobDetector  
4     params = cv2.SimpleBlobDetector_Params()  
5  
6     # Set Area filtering parameters  
7     params.filterByArea = True  
8     params.minArea = 100  
9  
10    # Set Circularity filtering parameters  
11    params.filterByCircularity = True  
12    params.minCircularity = 0.87  
13  
14    # Set Convexity filtering parameters  
15    params.filterByConvexity = True  
16    params.minConvexity = 0.02  
17  
18    # Set inertia filtering parameters  
19    params.filterByInertia = True
```



```
20     params.minInertiaRatio = 0.01
21
22     # Create a detector with the parameters
23     detector = cv2.SimpleBlobDetector_create(params)
24
25     # Detect blobs
26     keypoints = detector.detect(image)
27
28     return keypoints
```

Code 1: Blob detector

4.4.2 Kiểm tra hình ảnh thông qua model

Bằng việc sử dụng model đã được training sẵn cùng với phân vùng hình ảnh đã được xác định ở bước trên, ta tiến hành đối chiếu để có thể xác định được loại biển báo nào, sau đó xác định được kết quả là một con số định danh cho loại biển báo đó:

```
1 #Load trained model
2 sign_model = tf.keras.models.load_model(model_path)
3
4 #use Blob detector with initial image as draw
5 keypoints = None
6 keypoints = detect_keypoints(proc_image)
7
8 input_data_list = []
9 rects = []
10 range_images = []
11
12 if "KeyPoint" not in str(keypoints):
13     pass
14 else:
15     blank = np.zeros((1, 1))
16     draw = cv2.drawKeypoints(draw, keypoints, blank, (0, 0, 255), cv2.
17                             DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
18
19     for keypoint in keypoints:
20         x = keypoint.pt[0]
```



```
20     y = keypoint.pt[1]
21     center = (int(x), int(y))
22     radius = int(keypoint.size / 2)
23
24     # Bounding box:
25     im_height, im_width, _ = draw.shape
26     pad = int(0.4*radius)
27     tl_x = max(0, center[0] - radius - pad)
28     tl_y = max(0, center[1] - radius - pad)
29     br_x = min(im_width-1, tl_x + 2 * radius + pad)
30     br_y = min(im_height-1, tl_y + 2 * radius + pad)
31
32     rect = ((tl_x, tl_y), (br_x, br_y))
33     crop = image[tl_y:br_y, tl_x:br_x]
34     range_image = abs(tl_x - tl_y)
35
36     image_fromarray = IM.fromarray(crop, 'RGB')
37     resize_image = image_fromarray.resize((30, 30))
38     expand_input = np.expand_dims(resize_image, axis=0)
39     input_data = np.array(expand_input)
40     input_data = input_data/255
41
42     input_data_list.append(input_data)
43     rects.append(rect)
44     range_images.append(range_image)
45
46 if len(range_images) != 0 and max(range_images) > 165 and len(input_data_list)
47 != 0:
48
49     preds = sign_model.predict(input_data_list)
50     preds = np.argmax(preds, axis=1)
51     preds = preds.tolist()
52
53     for i in range(len(preds)):
54         if preds[i] == 14:
55             flag = 1
56             cv2.rectangle(draw, rects[i][0], rects[i][1], (0, 0, 255), 2)
```



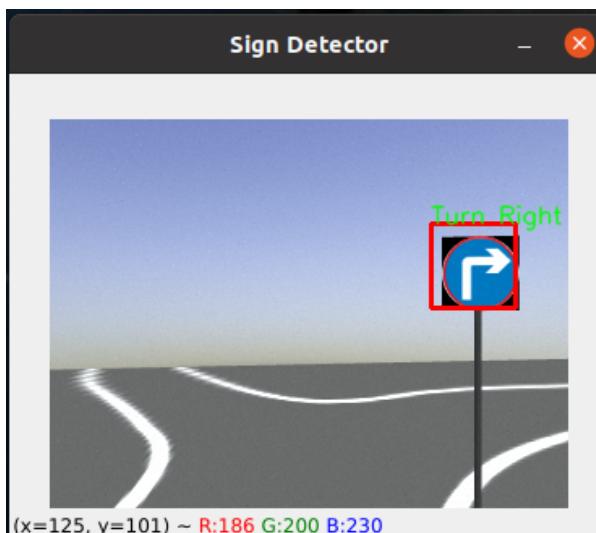
```
56     draw = cv2.putText(draw, "Stop Sign", rect[i][0], cv2.  
57         FONT_HERSHEY_SIMPLEX,  
58             0.5, (0, 255, 0), 1, cv2.LINE_AA)  
  
59  
60     elif preds[i] == 33:  
61         flag = 2  
62         cv2.rectangle(draw, rect[i][0], rect[i][1], (0, 0, 255), 2)  
63         draw = cv2.putText(draw, "Turn Right Sign", rect[i][0], cv2.  
64             FONT_HERSHEY_SIMPLEX,  
65                 0.5, (0, 255, 0), 1, cv2.LINE_AA)  
  
66  
67     elif preds[i] == 34:  
68         flag = 3  
69         cv2.rectangle(draw, rect[i][0], rect[i][1], (0, 0, 255), 2)  
70         draw = cv2.putText(draw, "Turn Left Sign", rect[i][0], cv2.  
71             FONT_HERSHEY_SIMPLEX,  
72                 0.5, (0, 255, 0), 1, cv2.LINE_AA)  
  
73  
74     elif preds[i] == 35:  
75         flag = 4  
76         cv2.rectangle(draw, rect[i][0], rect[i][1], (0, 0, 255), 2)  
77         draw = cv2.putText(draw, "Go Straight Sign", rect[i][0], cv2.  
78             FONT_HERSHEY_SIMPLEX,  
79                 0.5, (0, 255, 0), 1, cv2.LINE_AA)  
  
80     else:  
81         pass  
  
82  
83     if flag == 1:  
84         command_pub.publish("STOP")  
85     elif flag == 2:  
86         command_pub.publish("TURN_RIGHT")  
87     elif flag == 3:  
88         command_pub.publish("TURN_LEFT")  
89     elif flag == 4:  
90         command_pub.publish("GO_STRAIGHT")  
91     else:  
92         pass
```

```
89     flag = 0
90
91     input_data_list.clear()
92
93     rects.clear()
94
95     range_images.clear()

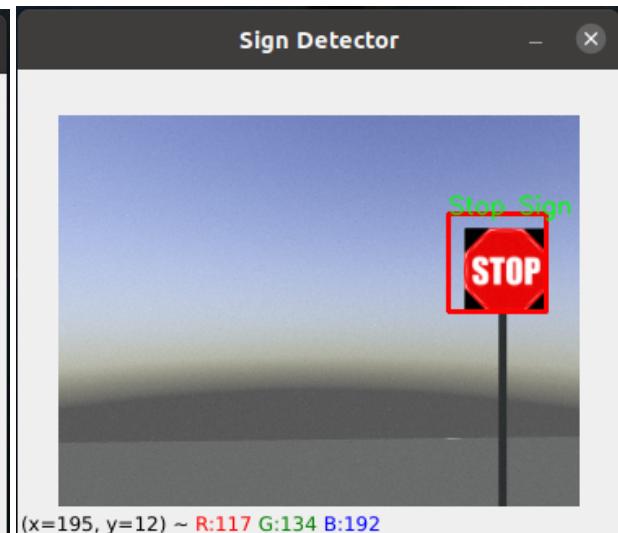
94     cv2.imshow("Sign Detector", draw)
95     cv2.waitKey(2)
```

Code 2: Kiểm tra hình ảnh với model

Kết quả thu được nếu nhận thấy có biển báo sẽ là:



Hình 12: Nhận diện biển báo rẽ phải



Hình 13: Nhận diện biển báo dừng lại

4.5 Hiện thực phát hiện làn đường

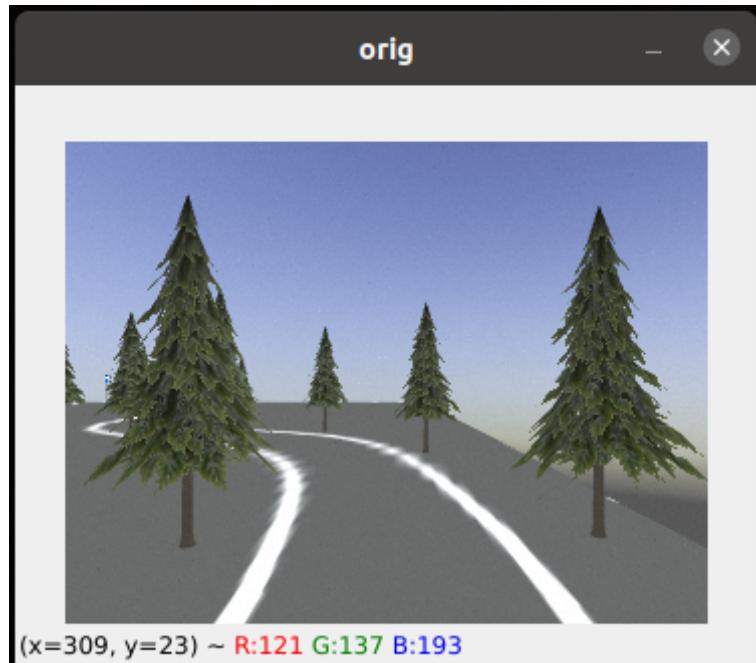
Hệ thống phát hiện làn đường gồm 2 thành phần đó là phần nhận biết làn đường(lane detection) và lập kế hoạch di chuyển, đánh lái (steering).

Ở phần nhận biết làn đường, nhóm đã sử dụng giải thuật Hough Transform - là một phương pháp trích xuất tính năng để phát hiện các hình dạng đơn giản như hình tròn, đường thẳng, v.v. trong một hình ảnh. Trong OpenCV, phát hiện line bằng Hough Transform được thực hiện trong hàm HoughLines và HoughLinesP (Probabilistic Hough Transform). Hàm này nhận các đối số sau:

- **edge**: Đầu ra của bộ dò cạnh.
- **lines**: Một vectơ để lưu trữ tọa độ của điểm bắt đầu và kết thúc của dòng.
- **rho**: Tham số độ phân giải ρ tính bằng pixel.

- **theta:** Độ phân giải của tham số Θ tính bằng radian.
- **threshold:** Số điểm giao nhau tối thiểu để phát hiện một đường.

Để áp dụng vào giải quyết vấn đề trong đồ án lần này, nhóm đã làm các bước như sau. Bắt đầu là hình ảnh đầu vào camera của xe mà nhóm thu nhận được:



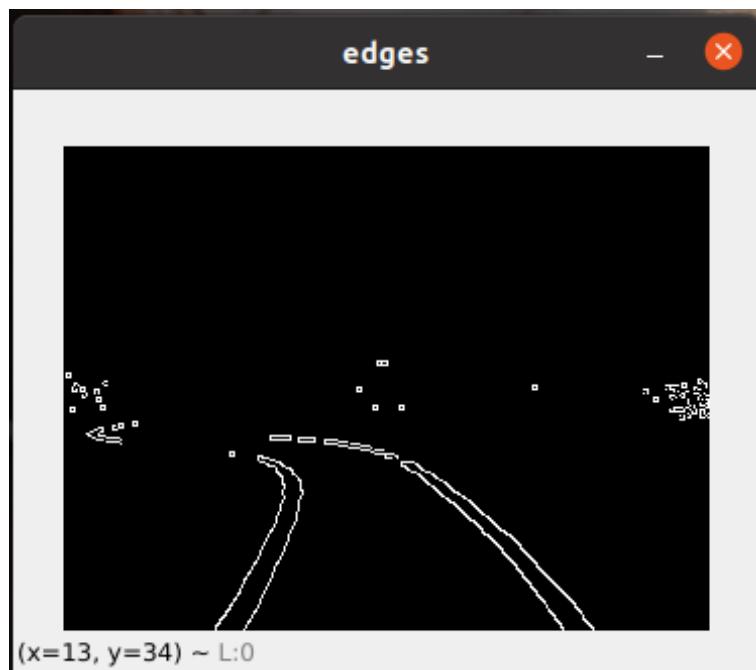
Hình 14: Hình ảnh ban đầu

Tiếp theo, chúng ta cần phát hiện các cạnh trong mặt nạ màu trắng để tạo ra một vài đường phân biệt đại diện các làn đường có trong trình mô phỏng. Chức năng Canny edge detection là một lệnh mạnh mẽ để phát hiện các cạnh trong hình ảnh.

Trong đoạn mã dưới đây, đầu tiên chuyển từ không gian gian màu BGR sang hệ màu GRAY, tiếp theo, ta tạo một mặt nạ màu trắng bằng lệnh `inRange` với khoảng từ (200-255). Cuối cùng, sử dụng lệnh Canny với tham số đầu tiên là mặt nạ màu trắng từ bước trước. Tham số thứ hai và thứ ba là phạm vi dưới và trên để phát hiện cạnh, mà OpenCV đề xuất là (100, 200) hoặc (200, 400), vì vậy chúng tôi đang sử dụng (200, 400).

```
1 def detect_edges(frame):  
2     # Color space conversion  
3     img_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  
4     # Detecting white colors  
5     mask_white = cv2.inRange(img_gray, 200, 255)  
6     # detect edges  
7     edges = cv2.Canny(mask_white, 200, 400)  
8     return edges
```

Code 3: Phát hiện cạnh trong hình



Hình 15: Lọc các cạnh trong hình

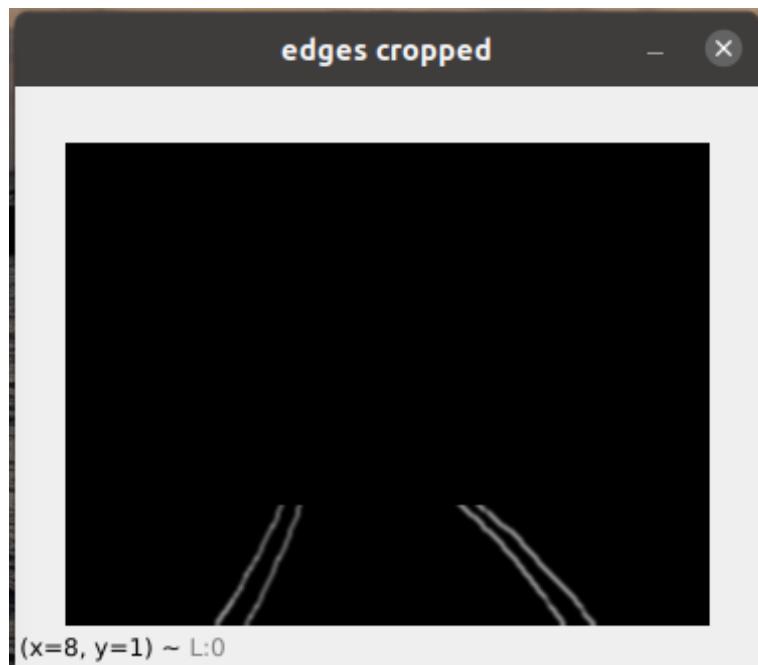
Từ hình ảnh trên, chúng ta thấy rằng chúng ta đã phát hiện ra khá nhiều vùng màu trắng KHÔNG phải là vạch phân làn của chúng ta. Quan sát kỹ hơn sẽ thấy rằng tất cả chúng đều nằm ở nửa trên của màn hình nhưng khi thực hiện điều hướng làn đường, chúng ta chỉ quan tâm đến việc phát hiện các vạch kẻ làn đường gần xe hơn, nơi cuối màn hình. Vì vậy, chúng tôi sẽ chỉ đơn giản là cắt bỏ nửa trên.

```
1 def region_of_interest(canny):
2     ysize = canny.shape[0]
3     xsize = canny.shape[1]
4     # Smoothing for removing noise
5     gray_blur = cv2.GaussianBlur(canny, (5, 5), 0)
6     # Region of Interest Extraction
7     mask_roi = np.zeros_like(gray_blur)
8     left_bottom = [0, ysize]
9     right_bottom = [xsize, ysize]
10    apex_left = [(0, (3*ysize/4))]
11    apex_right = [(xsize, (3*ysize/4))]
12    mask_color = 255
13    roi_corners = np.array(
```



```
14     [[left_bottom, apex_left, apex_right, right_bottom]], dtype=np.int32)
15     cv2.fillPoly(mask_roi, roi_corners, mask_color)
16     masked_image = cv2.bitwise_and(gray_blur, mask_roi)
17
18     return masked_image
```

Code 4: Cô lập khu vực quan tâm



Hình 16: Cắt hình

Trong hình ảnh hai làn đường thu được ở trên, đối với con người thì chúng ta có thể nhìn thấy bốn đường biểu thị cho hai làn đường. Tuy nhiên đối với máy tính, chúng chỉ là một loạt pixel màu trắng trên nền đen. Vì thế chúng ta sẽ sử dụng Hough Transform để tạo ra các đường thẳng từ hàng loạt các pixel đường như tạo thành đường thẳng với các thông số như sau:

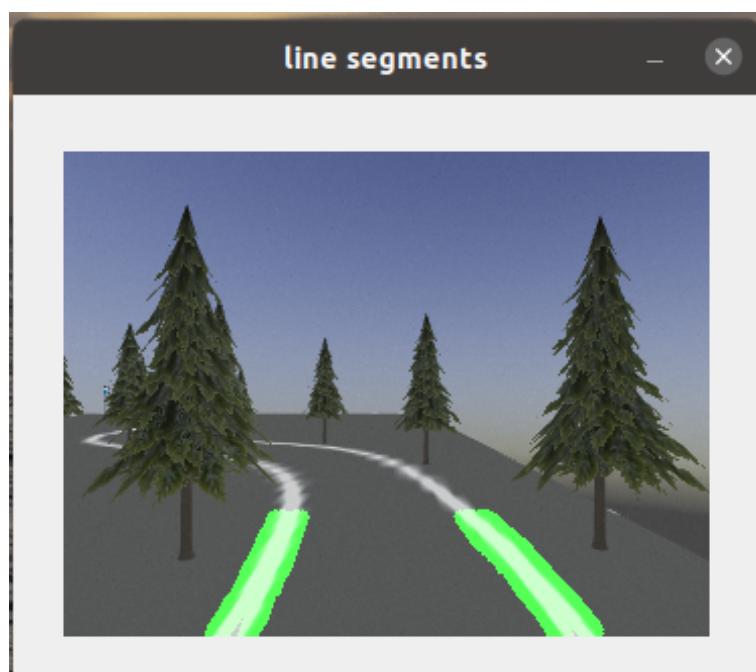
- **rho** là độ chính xác khoảng cách tính bằng pixel.
- **angle** là độ chính xác của góc tính bằng radian.
- **min_threshold** là số phiếu bầu cần thiết để được coi là một đoạn thẳng. Nếu một line có nhiều phiếu bầu hơn, Hough Transform coi chúng có nhiều khả năng đã phát hiện ra một đoạn dòng
- **minLineLength** là độ dài tối thiểu của đoạn thẳng tính bằng pixel. Hough Transform sẽ không trả về bất kỳ đoạn đường thẳng nào ngắn hơn độ dài tối thiểu này.

- **maxLineGap** là pixel tối đa mà hai đoạn thẳng có thể được phân tách và vẫn được coi là một đoạn thẳng.

Sau các thử nghiệm thì nhóm đã chọn ra được các thông số tối ưu cho giải thuật này, chi tiết được thể hiện trong đoạn code bên dưới:

```
1 def detect_line_segments(cropped_edges):  
2     rho = 1 # precision in pixel, i.e. 1 pixel  
3     angle = np.pi / 180 # degree in radian, i.e. 1 degree  
4     min_threshold = 10 # minimal of votes  
5     line_segments = cv2.HoughLinesP(cropped_edges, rho, angle, min_threshold,  
6                                     np.array([]), minLineLength=10, maxLineGap=4)  
7  
8     return line_segments
```

Code 5: Phát hiện line segments



Hình 17: Line segments

Từ các đoạn thẳng có được từ bước trên, đoạn code dưới đây sẽ chuyển chúng thành một hoặc hai đường thẳng. Nếu tất cả các hệ số góc của đường đều < 0 : thì chỉ phát hiện làn đường bên trái. Nếu tất cả các hệ số góc của đường đều > 0 : thì chỉ phát hiện làn đường bên phải.

make_points là một hàm trợ giúp cho hàm **average_slope_intercept**, hàm này nhận slope và intercept của đường và trả về các điểm cuối của đoạn thẳng.



```
1 def average_slope_intercept(frame, line_segments):
2     lane_lines = []
3     if line_segments is None:
4         return lane_lines
5
6     height, width, _ = frame.shape
7     left_fit = []
8     right_fit = []
9
10    boundary = 1/3
11    # left lane line segment should be on left 2/3 of the screen
12    left_region_boundary = width * (1 - boundary)
13    # right lane line segment should be on left 1/3 of the screen
14    right_region_boundary = width * boundary
15
16    for line_segment in line_segments:
17        for x1, y1, x2, y2 in line_segment:
18            if x1 == x2:
19                continue
20            fit = np.polyfit((x1, x2), (y1, y2), 1)
21            slope = fit[0]
22            intercept = fit[1]
23            if slope < 0:
24                if x1 < left_region_boundary and x2 < left_region_boundary:
25                    left_fit.append((slope, intercept))
26            else:
27                if x1 > right_region_boundary and x2 > right_region_boundary:
28                    right_fit.append((slope, intercept))
29
30    left_fit_average = np.average(left_fit, axis=0)
31    if len(left_fit) > 0:
32        lane_lines.append(make_points(frame, left_fit_average))
33
34    right_fit_average = np.average(right_fit, axis=0)
35    if len(right_fit) > 0:
36        lane_lines.append(make_points(frame, right_fit_average))
```



```
37
38     return lane_lines
39
40 def make_points(frame, line):
41     height, width, _ = frame.shape
42     slope, intercept = line
43     y1 = height # bottom of the frame
44     y2 = int(y1 * 3 / 4) # make points from middle of the frame down
45
46     # bound the coordinates within the frame
47     x1 = max(-width, min(2 * width, int((y1 - intercept) / slope)))
48     x2 = max(-width, min(2 * width, int((y2 - intercept) / slope)))
49     return [[x1, y1, x2, y2]]
```

Việc cuối cùng trong phát hiện làn đường là tính toán góc lệch hiện tại dựa trên tọa độ các làn đường đã tìm được ở các bước trên. Chi tiết code được thể hiện bên dưới:

```
1 def compute_steering_angle(frame, lane_lines):
2     if len(lane_lines) == 0:
3         return -90
4
5     height, width, _ = frame.shape
6     if len(lane_lines) == 1:
7         x1, _, x2, _ = lane_lines[0][0]
8         x_offset = x2 - x1
9     else:
10        _, _, left_x2, _ = lane_lines[0][0]
11        _, _, right_x2, _ = lane_lines[1][0]
12        mid = int(width / 2)
13        x_offset = (left_x2 + right_x2) / 2 - mid
14
15     # find the steering angle, which is angle between navigation direction to
16     # end of center line
17
18     y_offset = int(3*height / 4)
19
20     # angle (in radian) to center vertical line
21     angle_to_mid_radian = math.atan(x_offset / y_offset)
22     # angle (in degrees) to center vertical line
```



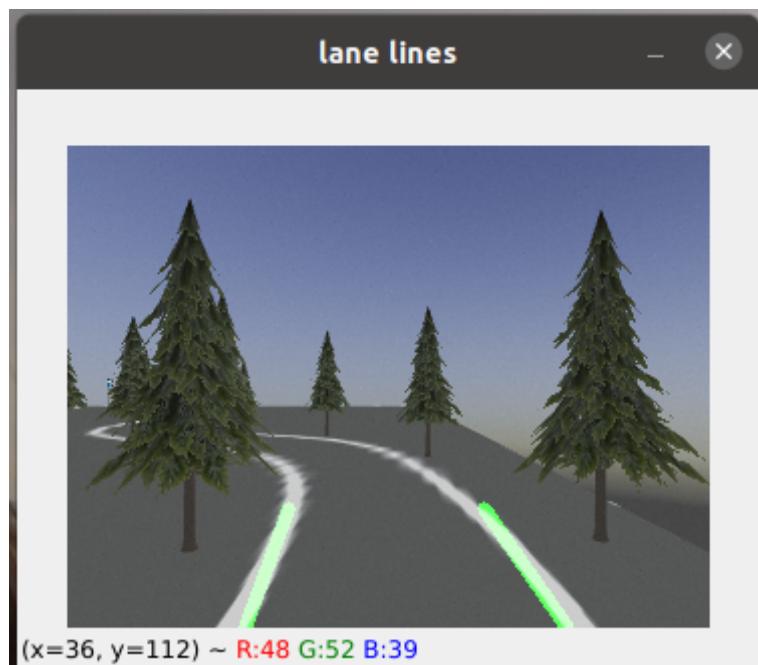
```
21     angle_to_mid_deg = int(angle_to_mid_radian * 180.0 / math.pi*0.6)
22     # this is the steering angle needed by picar front wheel
23     steering_angle = angle_to_mid_deg + 90
24
25     return steering_angle
```

Code 6: Tính toán góc lệch

Để dễ dàng quan sát, nhóm đã vẽ thêm 2 làn đường và đường thẳng chính giữa màu đỏ biển thị góc lái. Code được thể hiện bên dưới:

```
1 def display_lines(frame, lines, line_color=(0, 255, 0), line_width=5,
2     line_hight=4):
3     line_image = np.zeros_like(frame)
4     if lines is not None:
5         for line in lines:
6             for x1, y1, x2, y2 in line:
7                 cv2.line(line_image, (x1, y1), (x2, y2),
8                         line_color, line_width, line_hight)
9     line_image = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
10    return line_image
```

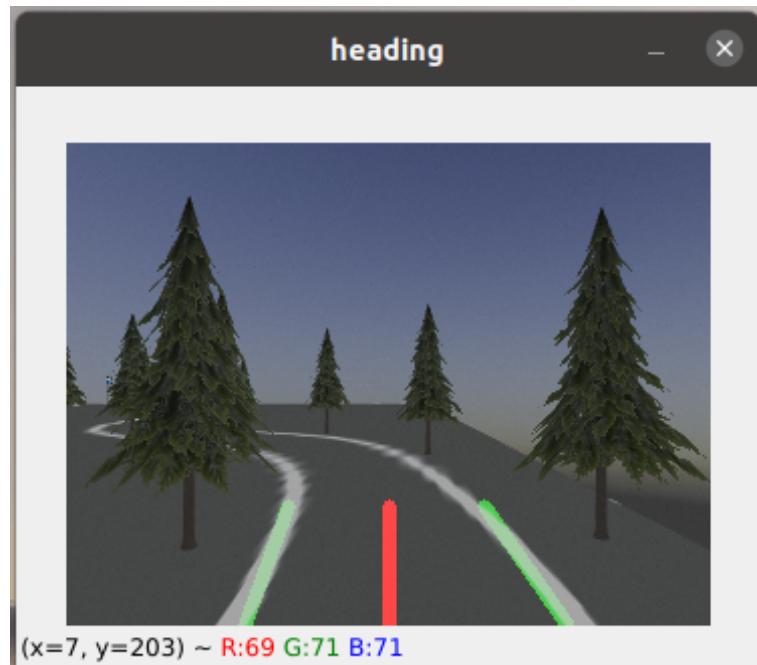
Code 7: Thêm hai làn đường



Hình 18: Vẽ 2 làn đường

```
1 def display_heading_line(frame, steering_angle, line_color=(0, 0, 255),  
2     line_width=5, line_hight=4):  
3     heading_image = np.zeros_like(frame)  
4     height, width, _ = frame.shape  
5  
5     steering_angle_radian = steering_angle / 180.0 * math.pi  
6     x1 = int(width / 2)  
7     y1 = height  
8     x2 = int(x1 - 2*height/3/math.tan(steering_angle_radian))  
9     y2 = int(3*height/4)  
10  
11    cv2.line(heading_image, (x1, y1), (x2, y2),  
12              line_color, line_width, line_hight)  
13    heading_image = cv2.addWeighted(frame, 0.8, heading_image, 1, 1)  
14  
15    return heading_image
```

Code 8: Thêm đường định hướng



Hình 19: Vẽ thêm đường xác định góc



4.6 Hiện thực điều khiển xe

Việc đầu tiên là kết nối với topic image_raw thông qua CvBridge để lấy được hình ảnh từ camera và tạo các Publisher để gửi lệnh điều khiển đến trình mô phỏng:

```
1 cv_bridge = CvBridge()
2 velocity_pub = rospy.Publisher('/autoware_gazebo/velocity',
3                                Float64, queue_size=5)
4 steeting_angle_pub = rospy.Publisher('/autoware_gazebo/steering_angle',
5                                Float64, queue_size=5)
6 def main():
7     global rate, velocity_pub
8     rospy.init_node('control', anonymous=True)
9     rate = rospy.Rate(50)
10
11    lane_sub = rospy.Subscriber(
12        "/image_raw", Image, lane_callback, queue_size=1)
13    sign_sub = rospy.Subscriber(
14        "/image_raw", Image, sign_callback, queue_size=1)
15    rospy.Subscriber("/command_control", String, command_sign_callback)
16    rospy.spin()
```

Xây dựng các hàm callback để xử lý dữ liệu nhận được và trả lại yêu cầu xuống cho trình mô phỏng:

```
1 def lane_callback(data):
2     global velocity_pub, steeting_angle_pub, angle_rad, command_sign, speed
3     try:
4         land_follower = LaneDetector()
5         frame = cv_bridge.imgmsg_to_cv2(data, "bgr8")
6         combo_image = land_follower.follow_lane(frame)
7         if command_sign == 0:
8             steeting_angle.append(land_follower.curr_steering_angle)
9             if(land_follower.line == 2):
10                 speed = 10
11             elif(land_follower.line == 1):
12                 speed = 5
13             elif(land_follower.line == 0):
14                 speed = 0
```



```
15         else:  
16             pass  
17         elif command_sign == 1:  
18             speed = 0  
19         else:  
20             pass  
21  
22     if(len(steeting_angle) == 10):  
23         data = np.average(steeting_angle, axis=None)  
24         angle_rad = round(((90 - data)/180 * math.pi * 1.2), 3)  
25         if(land_follower.line == 1):  
26             angle_rad = round(angle_rad + 0.5*angle_rad, 3)  
27         steeting_angle.clear()  
28  
29         velocity_pub.publish(Float64(speed))  
30         steeting_angle_pub.publish(Float64(angle_rad))  
31         print("Info: ", speed, angle_rad)  
32  
33     cv2.imshow('Lane Detector', combo_image)  
34     cv2.waitKey(2)  
35  
36 except CvBridgeError as e:  
37     print(e)
```

```
1 def sign_callback(data):  
2     global image_global  
3     try:  
4         image_global = cv_bridge.imgmsg_to_cv2(data, "bgr8")  
5         image_np = image_global.copy()  
6         image_np = cv2.cvtColor(image_np, cv2.COLOR_BGR2GRAY)  
7  
8         #Sign detector  
9         callback_processing_thread(image_np)  
10  
11    except CvBridgeError as e:  
12        print(e)
```



```
1 def command_sign_callback(data):
2     global command_sign, velocity_pub, steeting_angle_pub
3
4     print(data.data)
5     if data.data == "STOP":
6         command_sign = 1
7         velocity_pub.publish(Float64(0))
8         steeting_angle_pub.publish(Float64(0))
9
10    elif data.data == "TURN_RIGHT":
11        command_sign = 2
12        velocity_pub.publish(Float64(13))
13        start = rospy.get_time()
14        duration = rospy.Duration(0.175)
15        while((rospy.get_time() - start) < duration.to_sec()):
16            steeting_angle_pub.publish(Float64(-0.55))
17        velocity_pub.publish(Float64(0))
18        command_sign = 0
19
20    elif data.data == "TURN_LEFT":
21        command_sign = 3
22        velocity_pub.publish(Float64(13))
23        start = rospy.get_time()
24        duration = rospy.Duration(0.175)
25        while((rospy.get_time() - start) < duration.to_sec()):
26            steeting_angle_pub.publish(Float64(0.55))
27        velocity_pub.publish(Float64(0))
28        command_sign = 0
29
30    elif data.data == "GO_STRAIGH":
31        command_sign = 4
32        velocity_pub.publish(Float64(10))
33    else:
34        command_sign = 0
```

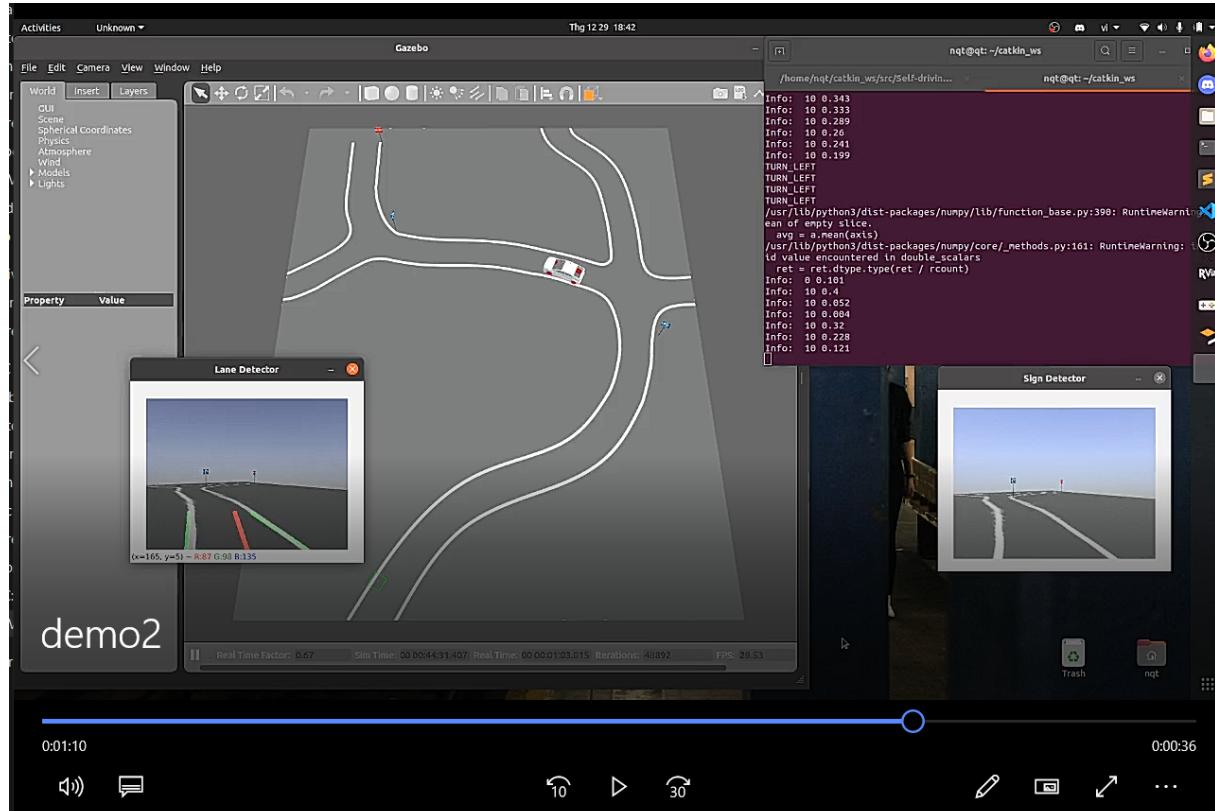
5 Kết quả và đánh giá

5.1 Kết quả

Video mô phỏng: [Link](#)

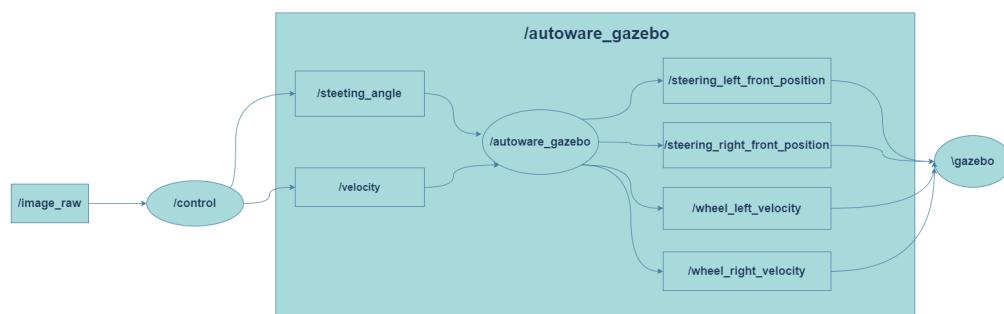
Link Github: [Self-driving-car](#)

Kết quả hiển thị trong mô phỏng:



Hình 20: Kết quả mô phỏng

Sơ đồ kết nối các node



Hình 21: Sơ đồ kết nối các node



5.2 Ưu điểm

- Hiện thực cơ bản các chức năng hệ thống
- Hệ thống có thể kết nối với các phần cứng để phát triển sau này

5.3 Hạn chế

- Chưa hiện thực được hết các chức năng của hệ thống
- Hệ thống nhận diện hoạt động chưa được ổn định

5.4 Phương hướng phát triển

Nhóm đã hoàn thành được các tính năng cơ bản của hệ thống. Tuy nhiên bên cạnh đó vẫn có một số chức năng mà nhóm chưa thực hiện được và nhóm cũng có thể hiểu để thực hiện được một hệ thống để sử dụng tốt trong thế giới thực cần thêm nhiều giai đoạn khác cũng như kiến thức và kinh nghiệm. Hiện tại nhóm chỉ có thể đề ra các phương hướng phát triển mà nhóm cho là đúng đắn sau này cho hệ thống.

- Hiện thực các chức năng còn thiếu của hệ thống như : nhận diện vật thể, né tránh người đi đường, nhận diện tín hiệu đèn giao thông.
- Hiện thực cải thiện các chứng năng chính của hệ thống
- Hiện thực phần cứng cho hệ thống



Kết luận

Qua thời gian nghiên cứu và làm việc nhóm em đã phần nào nắm được cách hoạt động của thị giác máy tính và các giả lập được sử dụng để phục vụ cho đề tài xe tự hành. Đặc biệt đồ án nghiên cứu đã giới thiệu về xe tự hành cũng như chứng minh cho độc giả thấy được đây là một công nghệ hay, cần được tiếp tục và nghiên cứu rộng hơn nữa.

Tuy nhiên trong đồ án này vẫn có nhiều ưu điểm của công nghệ nhưng nhóm vẫn chưa thực hiện được, cũng như thiếu hụt cơ sở hạ tầng và thiết bị để thiết kế một phần cứng cho hệ thống. Bên cạnh đó là sự ảnh hưởng của dịch Covid-19 đã làm khó khăn trong quá trình thực hiện đồ án này. Vì vậy cần thêm nhiều thời gian để có thể tiếp tục và nghiên cứu hơn nữa trong lĩnh vực này cũng như là cải thiện sản phẩm hiện tại mà nhóm đang có.

Cuối cùng nhóm xin gửi lời cảm ơn chân thành đến Thầy Trần Thanh Bình giảng viên phụ trách hướng dẫn nhóm và cũng là người giúp nhóm hoàn thành đồ án này.

Tài liệu

- [1] Lentin Joseph. *Robot Operating System for Absolute Beginners*.
- [2] *Blob Detection*. <https://learnopencv.com/blob-detection-using-opencv-python-c/>.
- [3] *HoughLine Transform*. https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html.
- [4] *Canny Edge Detection*. https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html.
- [5] *Tensorflow Keras Model*. https://www.tensorflow.org/api_docs/python/tf/keras/Model.
- [6] *Traffic Sign Detection*. <https://www.pyimagesearch.com/2019/11/04/traffic-sign-classification-with-keras-and-deep-learning/>.