

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM  
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG  
KHOA ĐIỆN – ĐIỆN TỬ



LÊ NHẬT TÂN

**MÔ PHỎNG PHÁT HIỆN BIỂN BÁO  
TỐC ĐỘ ĐIỀU KHIỂN TỐC ĐỘ ROBOT  
TỰ HÀNH TURTLEBOT3**

**ĐỒ ÁN CHUYÊN NGÀNH  
KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM  
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG  
KHOA ĐIỆN – ĐIỆN TỬ



LÊ NHẬT TÂN \_42001078

**MÔ PHỎNG PHÁT HIỆN BIẾN BÁO  
TỐC ĐỘ ĐIỀU KHIỂN TỐC ĐỘ ROBOT  
TỰ HÀNH TURTLEBOT3**

**KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG**

Người hướng dẫn  
**TS. Lê Anh Vũ**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

## LỜI CẢM ƠN

Đầu tiên em xin gửi lời cảm ơn chân thành đến giảng viên hướng dẫn Đồ án chuyên ngành của em, TS. Lê Anh Vũ. Trong quá trình thực hiện Đồ án, em đã có những thắc mắc và được Thầy nhiệt tình giải thích. Thầy còn truyền đạt thêm cho em nhiều kiến thức liên quan đến đề tài và em cảm thấy rất biết ơn vì Thầy đã dành thời gian để giảng giải cho em. Và nhờ sự hướng dẫn nhiệt tình cũng như định hướng của Thầy đã giúp em có thể hoàn thành được trọn vẹn Đồ án chuyên ngành của em.

Tiếp đến em xin cảm ơn các Thầy Cô trong khoa Điện - Điện tử, nhờ những kiến thức em được Thầy Cô truyền đạt trước đây đã giúp em có đủ hiểu biết để có thể nghiên cứu xa hơn và thực hiện các dự án như thế này.

Ngoài ra em cũng cảm ơn bạn Lê Hải Đăng đã hỗ trợ em, bạn đã giúp em hiểu rõ hơn về cách thức thực hiện đề tài.

Mọi sự giúp đỡ đến từ Thầy Lê Anh Vũ và các Thầy Cô, bạn bè để em có thể thực hiện được Đồ án này, em xin ghi nhớ và trân trọng. Và một lần nữa, em xin cảm ơn Thầy Lê Anh Vũ đã nhận lời hướng dẫn và hướng dẫn Đồ án chuyên ngành cho em!

TP. Hồ Chí Minh, ngày 1 tháng 5 năm 2024

Tác giả

Lê Nhật Tân

# CÔNG TRÌNH ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của TS. Lê Anh Vũ. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Đồ án chuyên ngành còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Đồ án chuyên ngành của mình.** Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 1 tháng 5 năm 2024

Tác giả

Lê Nhật Tân

# MÔ PHỎNG PHÁT HIỆN BIỂN BÁO TỐC ĐỘ ĐIỀU KHIỂN TỐC ĐỘ ROBOT TỰ HÀNH TURTLEBOT3

## TÓM TẮT

Đồ án sử dụng ROS Noetic Ninjemys hoạt động trên hệ điều hành Ubuntu 20.04 để mô phỏng robot tự hành TurtleBot3 di chuyển điểm điểm trong bản đồ đã định trước, bản đồ này được xây dựng mới hoàn toàn với phần mềm giả lập Gazebo.

Khi robot tự động di chuyển đến đích đã đánh dấu, sẽ kết hợp phát hiện 3 biển báo tốc độ bằng mô hình nhận diện được đào tạo với YOLOv8 trên nền tảng Google Colab. Khi robot phát hiện được biển báo tốc độ sẽ tự động điều chỉnh lại tốc độ theo quy định. Các tốc độ được điều khiển tương ứng với tên của 3 biển báo tốc độ như sau:

- Biển báo tốc độ *speed\_1*: 0.15 m/s.
- Biển báo tốc độ *speed\_2*: 0.20 m/s.
- Biển báo tốc độ *speed\_3*: 0.25 m/s.

Để thể hiện sự minh bạch về khả năng nhận diện vật thể của chương trình mô phỏng, ở đây là phát hiện các biển báo tốc độ, thì 3 biển báo tốc độ sẽ được đặt xen lẩn với 8 biển báo thông tin và những vật thể khác trong môi trường mô phỏng được xây dựng. Các biển báo được thiết kế trên Canva và mẫu robot sử dụng trong đồ án là Robot TurtleBot3 Waffle với tốc độ di chuyển tối đa lên đến 0.26 m/s.

Trong bài báo cáo của đồ án sẽ trình bày lý thuyết về cấu trúc của ROS như: node, topic, message, publish, subscribe cũng như định nghĩa máy học và thị giác máy tính; giới thiệu về các thuật toán tìm đường đi và mô hình động lực học truyền động vi sai 2 bánh. Tất cả nội dung bài báo cáo được trình bày qua 10 chương lớn:

1. GIỚI THIỆU ĐỀ TÀI.
2. CÀI ĐẶT MÔI TRƯỜNG.
3. CẤU TRÚC CỦA ROS.
4. XÂY DỰNG MÔI TRƯỜNG MÔ PHỎNG TRONG GAZEBO.
5. XÂY DỰNG BẢN ĐỒ ĐỊNH TRƯỚC SLAM.

6. TỰ ĐỊNH VỊ BẢN THÂN ACML.
7. MÔ HÌNH PHÁT HIỆN BIẾN BÁO TỐC ĐỘ.
8. THỰC THI PHẦN MỀM.
9. KẾT QUẢ MÔ PHỎNG.
10. KẾT LUẬN.

# MỤC LỤC

<b>DANH MỤC HÌNH VẼ .....</b>	<b>X</b>
<b>DANH MỤC BẢNG BIỂU.....</b>	<b>XIV</b>
<b>DANH MỤC CÁC CHỮ VIẾT TẮT .....</b>	<b>XV</b>
<b>CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI.....</b>	<b>1</b>
1.1    MỤC ĐÍCH THỰC HIỆN ĐỀ TÀI .....	1
1.2    MỤC TIÊU TỔNG QUÁT CỦA ĐỀ TÀI.....	2
1.3    MỤC TIÊU CỤ THỂ CỦA ĐỀ TÀI.....	3
<b>CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG.....</b>	<b>4</b>
2.1    CÀI ĐẶT UBUNTU 20.04.....	4
2.1.1    Giới thiệu về hệ điều hành Ubuntu .....	4
2.1.2    Tải file ISO .....	4
2.1.3    Tạo USB boot .....	5
2.1.4    Thực hiện cài đặt hệ điều hành .....	5
2.2    CÀI ĐẶT PHẦN MỀM .....	10
2.2.1    Cài đặt ROS.....	10
2.2.2    Cài đặt Gazebo.....	12
2.2.3    Cài đặt RVIZ .....	12
2.2.4    Cài đặt VS Code .....	13
2.3    THIẾT LẬP MÔI TRƯỜNG LÀM VIỆC CHO ROS .....	14
2.4    CÀI ĐẶT CÁC PHỤ THUỘC ĐỀ XÂY DỰNG CÁC GÓI TRONG ROS .....	14
2.5    GÓI TURTLEBOT .....	14
2.5.1    Giới thiệu về Turtlebot .....	14
2.5.2    Đặc điểm nổi bật của Turtlebot .....	15
2.5.3    Công nghệ lõi của Turtlebot .....	16
2.5.4    Quá trình phát triển của Turtlebot.....	17
2.5.5    Gói Turtlebot3.....	18
2.5.5.1    Giới thiệu về gói Turtlebot3 .....	18

2.5.5.2	Các mẫu robot trong gói Turtlebot3 .....	19
2.5.5.3	Tạo không gian làm việc giữa Turtlebot3 và ROS .....	20
2.5.5.4	Tạo không gian làm việc để phát hiện biến báo tốc độ .....	21
<b>CHƯƠNG 3. CẤU TRÚC CỦA ROS.....</b>	<b>22</b>	
3.1	NODE .....	22
3.2	TOPIC.....	23
3.3	MESSAGE .....	23
3.4	PUBLISH.....	24
3.5	SUBSCRIBE.....	24
3.6	CÂY TF TRONG ROS .....	24
3.7	CẤU TRÚC HOÀN CHỈNH CỦA ROS .....	26
<b>CHƯƠNG 4. XÂY DỰNG MÔI TRƯỜNG MÔ PHỎNG TRONG GAZEBO .....</b>	<b>28</b>	
4.1	XÂY DỰNG TƯỜNG .....	28
4.2	XÂY DỰNG CÁC BIÊN BÁO TỐC ĐỘ .....	28
4.3	XÂY DỰNG CÁC BIÊN BÁO THÔNG TIN .....	30
4.4	XÂY DỰNG MÔI TRƯỜNG MÔ PHỎNG .....	30
<b>CHƯƠNG 5. XÂY DỰNG BẢN ĐỒ ĐỊNH TRƯỚC SLAM.....</b>	<b>34</b>	
5.1	ĐỊNH NGHĨA SLAM .....	34
5.2	THỰC HIỆN SLAM .....	36
<b>CHƯƠNG 6. TỰ ĐỊNH VỊ BẢN THÂN ACML.....</b>	<b>38</b>	
6.1	ĐỊNH NGHĨA ACML.....	38
6.2	CÁC THUẬT TOÁN TÌM ĐƯỜNG ĐI CỦA TURTLEBOT3 .....	39
6.2.1	<i>Phân loại các thuật toán tìm đường đi.</i> .....	39
6.2.2	<i>Thuật toán tìm kiếm A* (A Star)</i> .....	41
6.2.3	<i>Thuật toán Dijkstra</i> .....	42
6.2.4	<i>Thuật toán DWA</i> .....	42
6.2.5	<i>So sánh các thuật toán</i> .....	43
6.3	MÔ HÌNH ĐỘNG LỰC HỌC TRUYỀN ĐỘNG VI SAI 2 BÁNH CỦA TURTLEBOT3 .....	44
6.4	THỰC HIỆN ACML .....	45
<b>CHƯƠNG 7. MÔ HÌNH PHÁT HIỆN BIÊN BÁO TỐC ĐỘ.....</b>	<b>48</b>	

7.1	ĐỊNH NGHĨA MÁY HỌC, THỊ GIÁC MÁY TÍNH VÀ MÔ HÌNH YOLOv8 .....	48
7.1.1	<i>Máy học .....</i>	48
7.1.2	<i>Thị giác máy tính.....</i>	50
7.1.3	<i>Mô hình YOLOv8 .....</i>	51
7.2	QUÁ TRÌNH HỌC MÁY .....	54
7.3	ĐÀO TẠO MÔ HÌNH PHÁT HIỆN BIÊN BÁO .....	55
7.3.1	<i>Chuẩn bị hình ảnh.....</i>	55
7.3.2	<i>Dán nhãn hình ảnh.....</i>	56
7.3.3	<i>Tiền xử lý ảnh.....</i>	58
7.3.4	<i>Tăng cường hình ảnh .....</i>	60
7.3.5	<i>Thực hiện đào tạo hình ảnh trên nền tảng Google Colab.....</i>	61
<b>CHƯƠNG 8. THỰC THI PHẦN MỀM .....</b>		<b>65</b>
8.1	CHƯƠNG TRÌNH PHÁT HIỆN BIÊN BÁO TỐC ĐỘ .....	65
8.1.1	<i>Sơ đồ nguyên lý chương trình phát hiện biến báo tốc độ.....</i>	65
8.1.2	<i>Mô tả hoạt động chương trình phát hiện biến báo tốc độ.....</i>	65
8.1.3	<i>Lưu đồ thuật toán chương trình phát hiện biến báo tốc độ .....</i>	66
8.2	CHƯƠNG TRÌNH PHÁT HIỆN BIÊN BÁO TỐC ĐỘ ĐIỀU KHIỂN TỐC ĐỘ ROBOT TỰ HÀNH TURTLEBOT3 .....	67
8.2.1	<i>Sơ đồ nguyên lý chương trình phát hiện biến báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3 .....</i>	67
8.2.2	<i>Mô tả hoạt động chương trình phát hiện biến báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3 .....</i>	68
8.2.3	<i>Lưu đồ thuật toán chương trình phát hiện biến báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3 .....</i>	69
<b>CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG .....</b>		<b>70</b>
9.1	MÔ HÌNH PHÁT HIỆN BIÊN BÁO SAU KHI ĐÀO TẠO .....	70
9.2	CÂY TF HOÀN CHỈNH CỦA CHƯƠNG TRÌNH MÔ PHỎNG .....	74
9.3	CÁC TOPIC ĐƯỢC SỬ DỤNG.....	74
9.4	KẾT QUẢ MÔ PHỎNG PHÁT HIỆN BIÊN BÁO TỐC ĐỘ .....	75
9.5	KẾT QUẢ MÔ PHỎNG PHÁT HIỆN BIÊN BÁO TỐC ĐỘ ĐIỀU KHIỂN TỐC ĐỘ ROBOT TỰ HÀNH TURTLEBOT3 .....	76

<b>CHƯƠNG 10. KẾT LUẬN.....</b>	<b>78</b>
10.1    KẾT LUẬN.....	78
10.2    ƯU ĐIỂM .....	78
10.3    NHƯỢC ĐIỂM .....	79
10.4    ỨNG DỤNG.....	79
10.5    HƯỚNG PHÁT TRIỂN.....	80
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>81</b>
<b>PHỤ LỤC A MÃ NGUỒN PYTHON CHƯƠNG TRÌNH PHÁT HIỆN BIỂN BÁO TỐC ĐỘ .....</b>	<b>A-1</b>
<b>PHỤ LỤC B MÃ NGUỒN PYTHON CHƯƠNG TRÌNH PHÁT HIỆN BIỂN BÁO TỐC ĐỘ ĐIỀU KHIỂN TỐC ĐỘ ROBOT TỰ HÀNH TURTLEBOT3.....</b>	<b>B-1</b>

## DANH MỤC HÌNH VẼ

Hình 1 - 1. Robot tự hành trong kho hàng của Amazon .....	2
Hình 2 - 1. Tải file ISO từ trang web của công ty.....	5
Hình 2 - 2. Boot Menu .....	6
Hình 2 - 3. Tắt Secure Boot trong BIOS .....	6
Hình 2 - 4. Máy đang nhận USB boot.....	7
Hình 2 - 5. Lựa chọn ngôn ngữ và cài đặt Ubuntu.....	7
Hình 2 - 6. Tạo tài khoản đăng nhập vào máy tính Ubuntu .....	8
Hình 2 - 7. Hoàn thành bước tạo tài khoản Ubuntu .....	8
Hình 2 - 8. Màn hình sau khi khởi động lại máy .....	9
Hình 2 - 9. Tóm lược kiến trúc ROS.....	10
Hình 2 - 10. Cài đặt ROS .....	11
Hình 2 - 11. Cài đặt file .deb của VS Code.....	13
Hình 2 - 12. Cài đặt phần mở rộng cho ngôn ngữ Python trên VS Code .....	14
Hình 2 - 13. Các thế hệ TurtleBots .....	17
Hình 2 - 14. Các mẫu robot trong gói TurtleBot3 .....	18
Hình 2 - 15. Các thành phần của Turtlebot3 Burger.....	19
Hình 2 - 16. So sánh thông số Turtlebot3 Burger và Waffle .....	20
Hình 3 - 1. Hai node trong ROS đang giao tiếp với nhau .....	22
Hình 3 - 2. Ví dụ về topic trong ROS .....	23
Hình 3 - 3. Ví dụ về message trong ROS .....	23
Hình 3 - 4. Chi tiết cấu trúc của ROS .....	26
Hình 4 - 1. Xây dựng tường trong ROS .....	28
Hình 4 - 2. Bản thiết kế của biển báo speed_1 .....	29
Hình 4 - 3. Bản thiết kế của biển báo speed_2 .....	29

Hình 4 - 4. Bản thiết kế của biển báo speed_3 .....	29
Hình 4 - 5. Các biển báo trong môi trường mô phỏng .....	30
Hình 4 - 6. Môi trường mô phỏng với góc nhìn từ trên xuống .....	31
Hình 4 - 7. Môi trường mô phỏng với góc nghiêng nhìn từ điểm cuối đến điểm bắt đầu .....	31
Hình 4 - 8. Môi trường mô phỏng với góc nghiêng nhìn từ điểm bắt đầu đến điểm cuối .....	32
Hình 4 - 9. Môi trường mô phỏng với góc nhìn gần các biển báo tốc độ .....	32
Hình 4 - 10. Môi trường mô phỏng với góc nhìn gần các biển báo thông tin .....	33
Hình 4 - 11. Môi trường mô phỏng với góc nhìn gần biển báo hình ảnh lá cờ Việt Nam và logo TDTU .....	33
Hình 5 - 1. Cách tiếp cận lý thuyết của một tác nhân/robot cố gắng tính toán vị trí thực của nó trong khi di chuyển .....	34
Hình 5 - 2. Thực hiện SLAM .....	36
Hình 5 - 3. Các bản đồ đã lưu sau khi SLAM .....	37
Hình 6 - 1. Ví dụ về một stack điều hướng ROS .....	39
Hình 6 - 2. Ví dụ về thuật toán lập kế hoạch đường đi cục bộ bằng phương pháp dựa trên lưới .....	40
Hình 6 - 3. Mô hình truyền động vi sai của TurtleBot3 .....	44
Hình 6 - 4. Công thức truyền động vi sai của TurtleBot3 .....	45
Hình 6 - 5. Hình ảnh lần đầu khi khởi chạy ACML .....	45
Hình 6 - 6. Đánh dấu lại vị trí robot .....	46
Hình 6 - 7. Chọn đích đến cho robot tự di chuyển .....	47
Hình 7 - 1. Phân loại máy học .....	48
Hình 7 - 2. Cấu trúc mô hình YOLO .....	52
Hình 7 - 3. So sánh YOLOv8 với các phiên bản khác .....	52

Hình 7 - 4. Các bước trong quá trình học máy.....	55
Hình 7 - 5. Tạo số lượng hình ảnh ban đầu.....	55
Hình 7 - 6. Tải hình ảnh lên Roboflow .....	56
Hình 7 - 7. Chọn phương thức dán nhãn dữ liệu trên Roboflow .....	57
Hình 7 - 8. Thực hiện dán nhãn dữ liệu trên Roboflow .....	57
Hình 7 - 9. Phân loại tệp hình ảnh sau khi dán nhãn.....	58
Hình 7 - 10. Kiểm tra số lượng và phân loại hình ảnh.....	58
Hình 7 - 11. Các bước tiền xử lý ảnh trên Roboflow.....	59
Hình 7 - 12. Tăng cường số lượng hình ảnh trên Roboflow .....	60
Hình 7 - 13. Chọn sử dụng GPU trên Google Colab .....	61
Hình 7 - 14. Tải về YOLOv8 trên Google Colab.....	62
Hình 7 - 15. Sao chép API để tải về tệp hình ảnh đã dán nhãn từ Roboflow .....	62
Hình 7 - 16. Tải về tệp hình ảnh đã dán nhãn trên Google Colab .....	62
Hình 7 - 17. Thực hiện đào tạo hình ảnh trên Google Colab .....	63
Hình 7 - 18. Kiểm tra các tệp mới được tạo sau khi thực hiện đào tạo hình ảnh trên Google Colab .....	63
Hình 7 - 19. Thực hiện thẩm định sau khi đào tạo .....	63
Hình 7 - 20. Tải về toàn bộ dữ liệu trên Google Colab.....	64
Hình 7 - 21. Bên trong thư mục chứa toàn bộ dữ liệu trên Google Colab đã tải về ..	64
 Hình 8 - 1. Sơ đồ nguyên lý chương trình phát hiện biển báo tốc độ .....	65
Hình 8 - 2. Topic /camera/rgb/image_raw và /cmd_vel trước khi khởi chạy detect.py .....	66
Hình 8 - 3. Topic /camera/rgb/image_raw và /cmd_vel sau khi khởi chạy detect.py .....	66
Hình 8 - 4. Lưu đồ thuật toán chương trình phát hiện biển báo tốc độ.....	67
Hình 8 - 5. Sơ đồ nguyên lý chương trình phát hiện biển báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3 .....	68

Hình 8 - 6. Topic /camera/rgb/image_raw và /cmd_vel sau khi khởi chạy detect.py và 8control.py.....	68
Hình 8 - 7. Lưu đồ thuật toán chương trình phát hiện biển báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3 .....	69
Hình 9 - 1. Kết quả tổng quan quá trình đào tạo .....	70
Hình 9 - 2. Biểu đồ thể hiện giá trị F1 .....	71
Hình 9 - 3. Biểu đồ thể hiện giá trị Precision.....	71
Hình 9 - 4. Biểu đồ thể hiện giá trị Recall .....	71
Hình 9 - 5. Biểu đồ thể hiện mối quan hệ giữa Precision và Recall .....	72
Hình 9 - 6. Ma trận nhầm lẫn .....	72
Hình 9 - 7. Phiên bản chuẩn hóa của ma trận nhầm lẫn.....	73
Hình 9 - 8. Cây TF hoàn chỉnh của đồ án .....	74
Hình 9 - 9. Danh sách các topic đang sử dụng.....	75
Hình 9 - 10. Kết quả thực thi chương trình phát hiện biển báo tốc độ .....	76
Hình 9 - 11. Chương trình phát hiện được biến báo tốc độ speed_1 và thay đổi tốc độ robot thành 0.15 m/s.....	76
Hình 9 - 12. Chương trình phát hiện được biến báo tốc độ speed_2 và thay đổi tốc độ robot thành 0.20 m/s.....	77
Hình 9 - 13. Chương trình phát hiện được biến báo tốc độ speed_3 và thay đổi tốc độ robot thành 0.25 m/s.....	77

## **DANH MỤC BẢNG BIỂU**

Bảng 5 - 1. So sánh LiDAR và Radar .....	35
Bảng 6 - 1. So sánh 2 loại thuật toán .....	41
Bảng 6 - 2. So sánh về độ phức tạp của 3 thuật toán .....	43
Bảng 7 - 1. So sánh các phiên bản của YOLOv8.....	53

## **DANH MỤC CÁC CHỮ VIẾT TẮT**

ACML	Adaptable Cognitive-Motivated Meta-Learning
AI	Artificial intelligence
API	Application Programming Interface
cmd_vel	command velocity
CNN	Convolutional Neural Network
DL	Deep Learning
DWA	Dynamic Window Approach
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
LINUX	Lovable Intellect Not Using XP
m	Meter
mAP	Mean Average Precision
ML	Machine Learning
NMS	Non-maximum suppression
OpenCV	Open Source Computer Vision Library
RADAR	Radio Detection and Ranging
RGB	Red, Green, Blue
ROS	Robot Operating System
RVIZ	ROS visualization
s	Second
SDF	Simulation Description Format
SLAM	Simultaneous Localization and Mapping
sudo	substitute user do / super user do
VS Code	Visual Studio Code
XML	Extensible Markup Language
YAML	Yet Another Markup Language

## CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

### 1.1 Mục đích thực hiện đề tài

Robot tự hành hiện nay đã đạt được những tiến bộ đáng kể và đang ngày càng được tích hợp nhiều hơn vào cuộc sống hàng ngày như ví dụ ở Hình 1 - 1. Mục đích của đề tài này là để phát triển một hệ thống mô phỏng có khả năng phát hiện biển báo tốc độ và điều chỉnh tốc độ của robot tự hành Turtlebot3 tương ứng. Điều này không chỉ giúp cải thiện khả năng tự động hóa của robot mà còn đóng góp vào nghiên cứu và phát triển các hệ thống tự lái thông minh, cũng như xây dựng cộng đồng phát triển robot nói chung và Turtlebot nói riêng thêm lớn mạnh.

Khi áp dụng vào thực tế, đề tài mô phỏng phát hiện biển báo tốc độ và điều khiển tốc độ robot tự hành Turtlebot3 có thể mang lại nhiều lợi ích và ứng dụng tiềm năng như:

- An toàn giao thông: Cải thiện an toàn giao thông bằng cách giúp các phương tiện tự hành tuân thủ các quy định về tốc độ, giảm nguy cơ tai nạn.
- Nghiên cứu và phát triển: Phục vụ làm mô hình thử nghiệm cho các nghiên cứu về hệ thống lái tự động và học máy, đặc biệt trong việc phát triển thuật toán nhận diện và phản ứng với biển báo giao thông.
- Giáo dục: Hỗ trợ giảng dạy và học tập trong lĩnh vực robot học và tự động hóa, cung cấp một nền tảng thực hành cho sinh viên và nhà nghiên cứu.
- Ứng dụng công nghiệp: Có thể được sử dụng trong các ngành công nghiệp như sản xuất, logistics, nơi mà việc di chuyển tự động và an toàn là quan trọng.
- Phát triển đô thị thông minh: Góp phần vào việc xây dựng các đô thị thông minh, nơi mà các phương tiện tự hành có thể giao tiếp và tương tác một cách thông minh với cơ sở hạ tầng giao thông.

Những ứng dụng này chỉ là một phần của tiềm năng mà đề tài có thể mang lại khi được triển khai và tích hợp vào các hệ thống thực tế. Khi tiếp tục nghiên cứu và phát

triển đề tài này sẽ đảm bảo rằng các mô phỏng có thể đáp ứng một cách hiệu quả và an toàn vào môi trường thực tế trong thời gian lâu dài.



**Hình 1 - 1. Robot tự hành trong kho hàng của Amazon**

(Nguồn: VnExpress)

## **1.2 Mục tiêu tổng quát của đề tài**

Đề tài mô phỏng phát hiện biển báo tốc độ và điều khiển tốc độ robot tự hành Turtlebot3 có những yêu cầu sau:

- Nắm được cấu trúc của ROS, bao gồm: node, topic, message, publish, subscribe và Gazebo, RVIZ.
- Xây dựng bản đồ định trước SLAM (gmapping) tự định vị bản thân trong bản đồ (ACML).
- Mô phỏng robot tự hành Turtlebot3 di chuyển trên nền ROS1 Neotic Ubuntu 20.04, mô phỏng Gazebo, quan sát RVIZ, phát hiện biển báo tốc độ. Điều khiển 3 tốc độ robot khác nhau.
- Xây dựng sơ đồ nguyên lý, lưu đồ thuật toán.
- Viết code, chạy code, kiểm tra sửa lỗi và mô phỏng.

- Hoàn thiện sản phẩm thực tế và viết báo cáo.

### **1.3 Mục tiêu cụ thể của đề tài**

Đề tài được thực hiện với các bước như sau:

- Cài đặt Ubuntu 20.04, ROS1 Neotic cùng với Gazebo và RVIZ.
- Tải và sử dụng các gói mô phỏng của Turtlebot3.
- Tạo môi trường làm việc để các thành phần (robot và môi trường giả lập) giao tiếp với nhau.
- Xây dựng bản đồ mô phỏng trong Gazebo để gắn các biển báo và mô phỏng robot tự hành Turtlebot3.
- Xây dựng bản đồ định trước SLAM bằng phương pháp gmapping, sau đó thực hiện tự định vị bản thân robot trong bản đồ này.
- Thiết kế 3 biển báo tốc độ bằng công cụ thiết kế đồ họa trực tuyến Canva và tạo các tệp dữ liệu như:
  - Training Set: để đào tạo dữ liệu.
  - Validation Set: để kiểm tra độ chính xác của mô hình máy học trong quá trình huấn luyện, tránh Overfitting.
  - Testing Set: tập dữ liệu dùng để kiểm tra sau khi mô hình đã học xong.
- Thực hiện dán nhãn dữ liệu (Data Labeling) với Roboflow. Sau đó thiết lập các bước tiền xử lý ảnh và tăng cường dữ liệu đầu vào để tăng độ chính xác.
- Thực hiện đào tạo dữ liệu (train data) với mô hình YOLOv8 trên nền tảng Google Colab để đạt được độ chính xác 97%.
- Thiết kế thêm 8 biển báo thông tin khác.
- Tìm hiểu ngôn ngữ XML, đưa tất cả các biển báo vào môi trường mô phỏng Gazebo.
- Tìm hiểu ngôn ngữ Python, viết chương trình phát hiện biển báo tốc độ.
- Viết chương trình phát hiện biển báo tốc độ sau đó điều khiển tốc độ robot tự hành Turtlebot3 di chuyển điểm đến với 3 tốc độ khác nhau (0.25 m/s, 0.20 m/s và 0.15 m/s) trong môi trường mô phỏng đã xây dựng trước đó.

## CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

### 2.1 Cài đặt Ubuntu 20.04

#### 2.1.1 Giới thiệu về hệ điều hành Ubuntu

Ubuntu là một hệ điều hành mở, miễn phí bản quyền và phù hợp với công việc lập trình. Bên cạnh đó hệ điều hành Ubuntu là một trong những hệ điều hành mã nguồn mở phổ biến nhất hiện nay. Ubuntu được phát triển bởi công ty Canonical và cộng đồng người dùng trên toàn thế giới. Những lý do để cài đặt Ubuntu như:

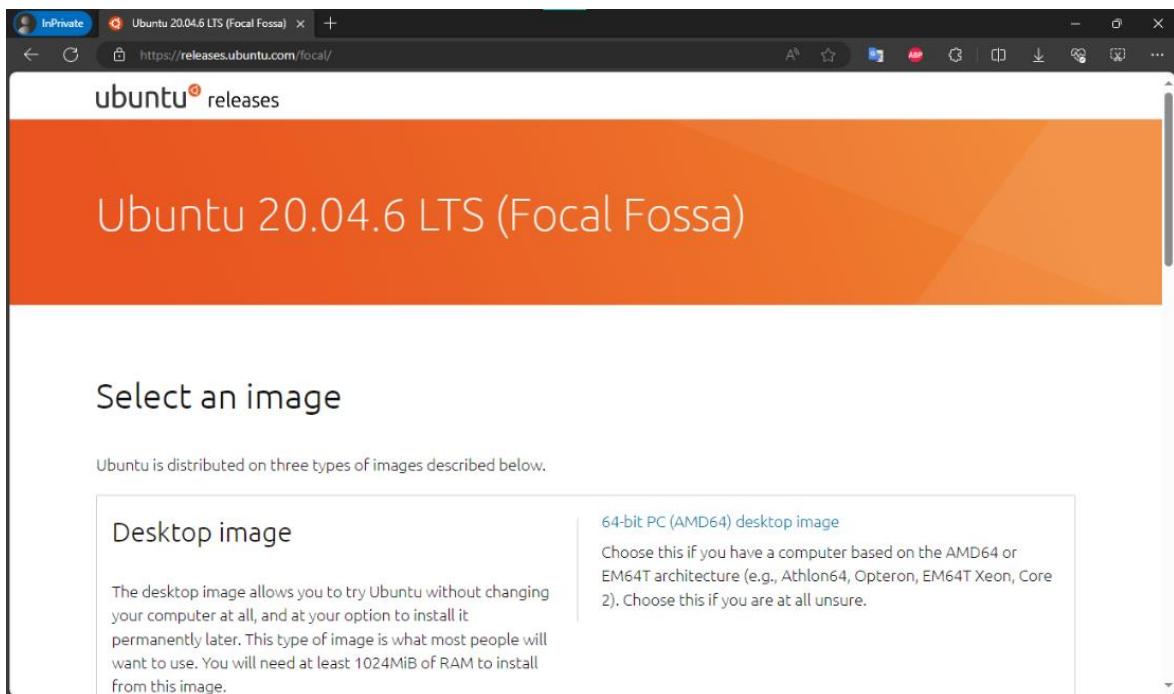
- Có nhiều phiên bản khác nhau, phù hợp với nhiều nhu cầu khác nhau.
- Ubuntu cung cấp một giao diện đồ họa dễ sử dụng, và có thể tùy biến.
- Ubuntu cũng có nhiều ứng dụng hữu ích nhưng miễn phí.
- Ubuntu được xây dựng trên nền tảng của hệ điều hành LINUX, nên có tính bảo mật cao, ổn định, và tương thích với nhiều loại phần cứng.

Hiện tại em cần sử dụng ROS Noetic hay còn gọi là ROS1 nên em sẽ chọn bản Ubuntu 20.04 để phù hợp nhất.

#### 2.1.2 Tải file ISO

Mỗi hệ điều hành đều được cài đặt từ file ISO. File ISO là một định dạng file dùng để lưu trữ bản sao của nội dung trên đĩa quang, như CD, DVD hay Blu-ray. File ISO chứa tất cả các tệp, thư mục và thông tin hệ thống tệp của đĩa gốc.

File ISO cũng có thể được sử dụng để tạo đĩa sao lưu, phân phối phần mềm hay cài đặt hệ điều hành. File ISO còn có thể được gắn vào ổ đĩa ảo hoặc được giải nén bằng các phần mềm như WinRAR. Để tải file ISO của Ubuntu 20.04 cần vào trang web của công ty Canonical như Hình 2 - 1.



**Hình 2 - 1. Tải file ISO từ trang web của công ty**

### **2.1.3 Tạo USB boot**

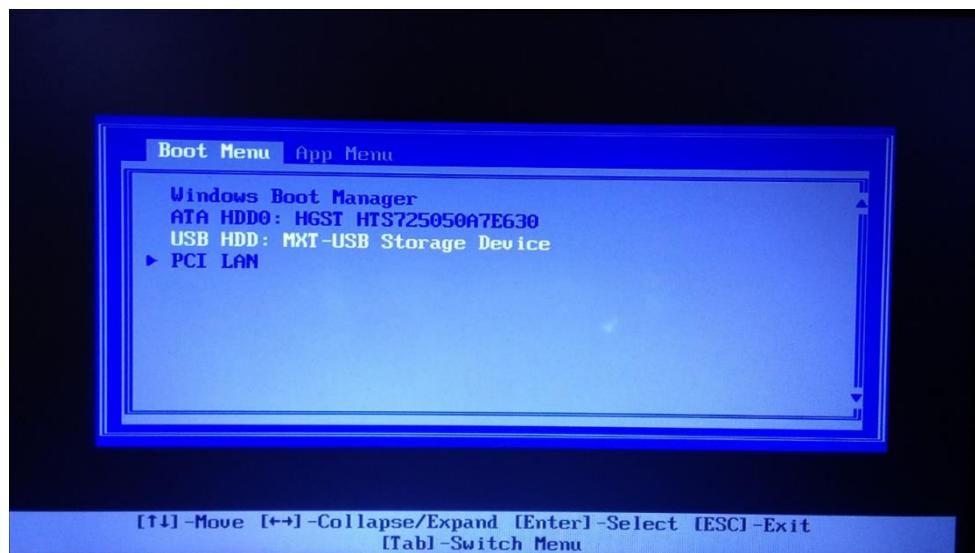
Sau khi đã có file ISO, cần chuyển vào 1 USB bằng các công cụ chuyên dụng để tạo USB boot. Có các phần mềm để thực hiện điều này, em chọn balenaEtcher vì đơn giản nhưng hiệu quả. Lưu ý, cần sao lưu dữ liệu trong USB.

Phần mềm balenaEtcher không chỉ giúp flash file ISO vào USB nhanh chóng chỉ với 1 bước mà còn kiểm tra lại, đảm bảo USB boot hoạt động được. Giao diện balenaEtcher gồm 3 phần:

- Flash from file: chọn file ISO.
- Select target: chọn USB tạo USB boot.
- Flash: balenaEtcher thực hiện việc flash file ISO vào USB đã chọn.

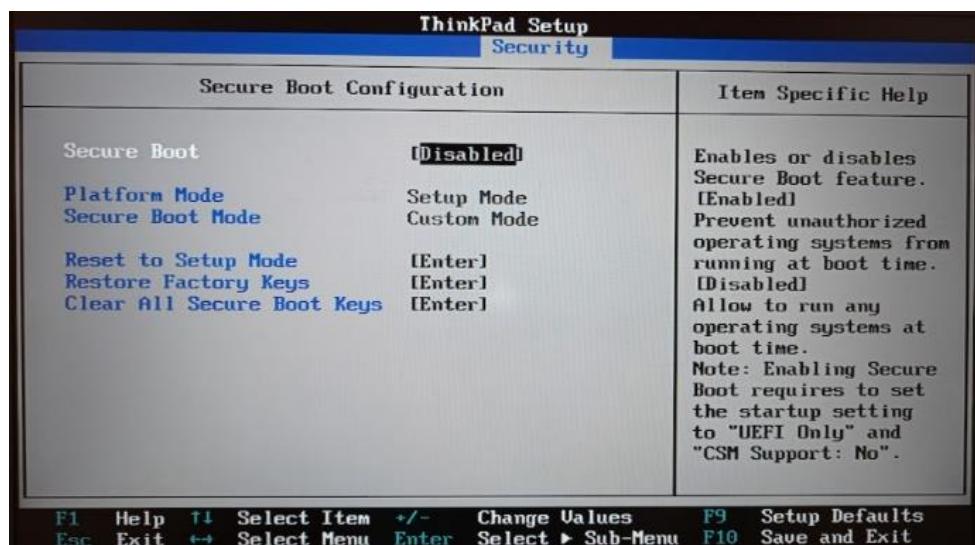
### **2.1.4 Thực hiện cài đặt hệ điều hành**

Khi đã chuẩn bị USB boot xong, cắm USB vào máy và bấm nút nguồn khởi động.



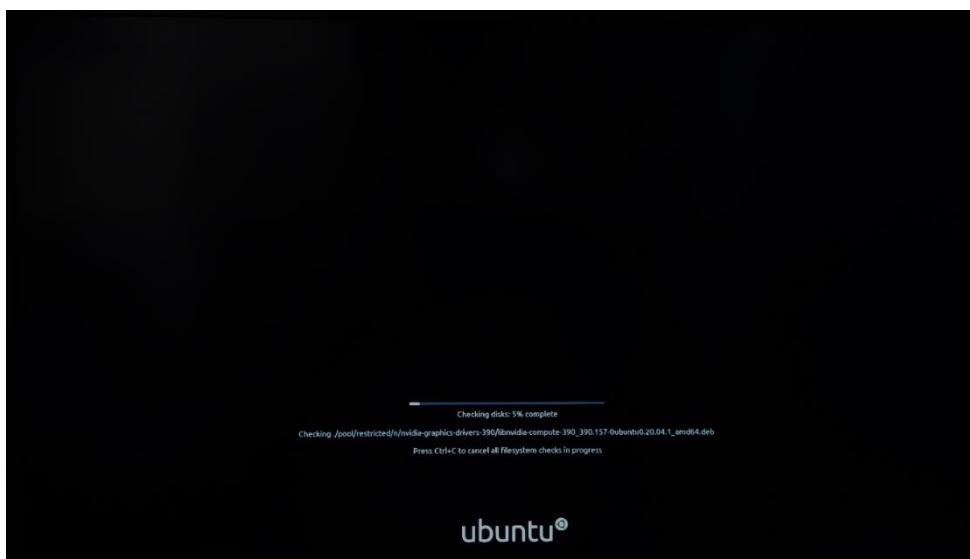
Hình 2 - 2. Boot Menu

Khi khởi động chú ý chọn phím tắt để truy cập vào BIOS hoặc Boot Menu như Hình 2 – 2. Tùy mỗi hãng sẽ có phím tắt khác nhau. Sau khi vào được BIOS thì chọn boot từ USB vừa cắm vào thay vì ổ đĩa, như vậy hệ điều hành mới sẽ bắt đầu được cài đặt vào. Trước khi cài đặt, đảm bảo đã sao lưu dữ liệu nếu trước đó đã cài đặt hệ điều hành nào đó hoặc ổ đĩa đang có dữ liệu. Nếu không vào được Boot Menu thì vào BIOS, tắt Secure Boot như Hình 2 – 3.

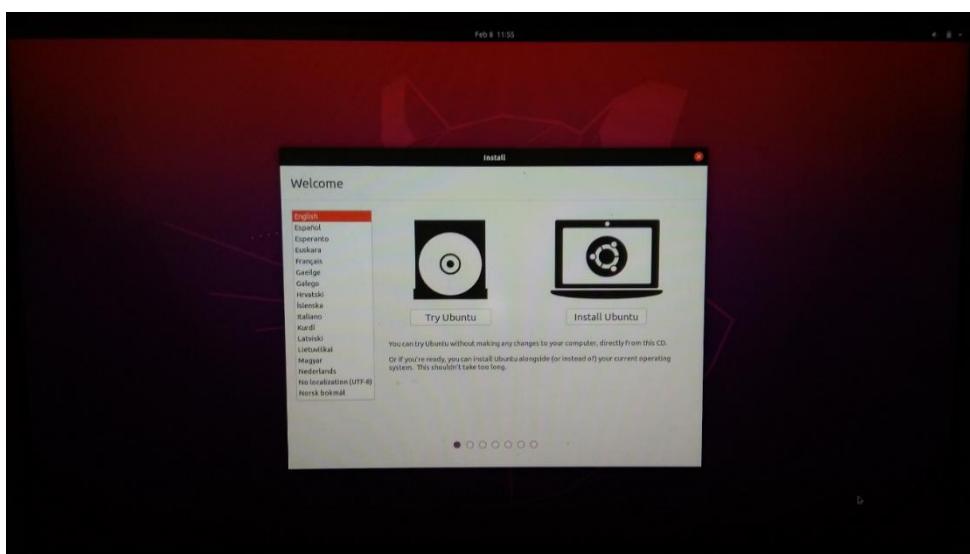


Hình 2 - 3. Tắt Secure Boot trong BIOS

Nếu trước đó đã có cài hệ điều hành Windows, thì vào Control Panel ➤ All Control Panel Items (View by: Large/Small icons) ➤ Power Options ➤ System Settings ➤ *Change settings that are currently unavailable* để tắt Turn on fast startup giúp dễ dàng vào BIOS. Hình 2 – 4 thể hiện máy đang nhận USB boot.



**Hình 2 - 4. Máy đang nhận USB boot**

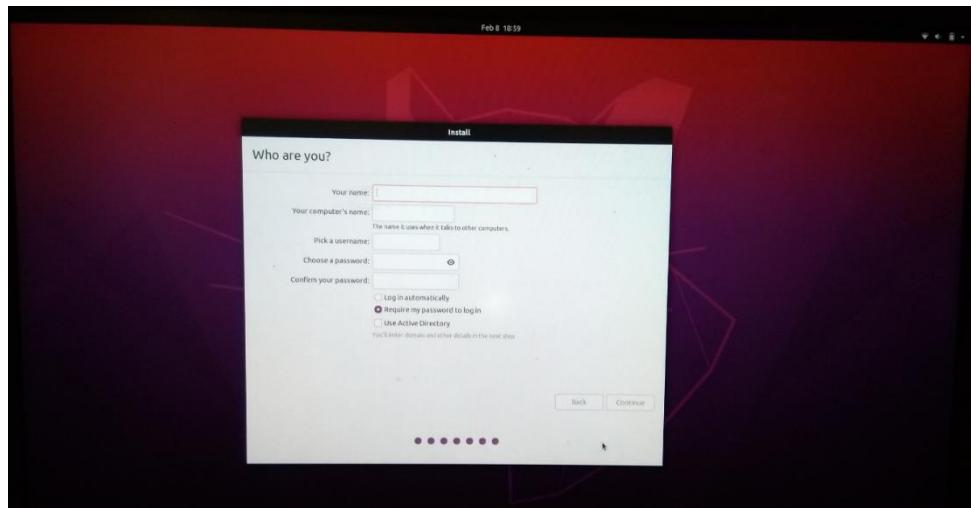


**Hình 2 - 5. Lựa chọn ngôn ngữ và cài đặt Ubuntu**

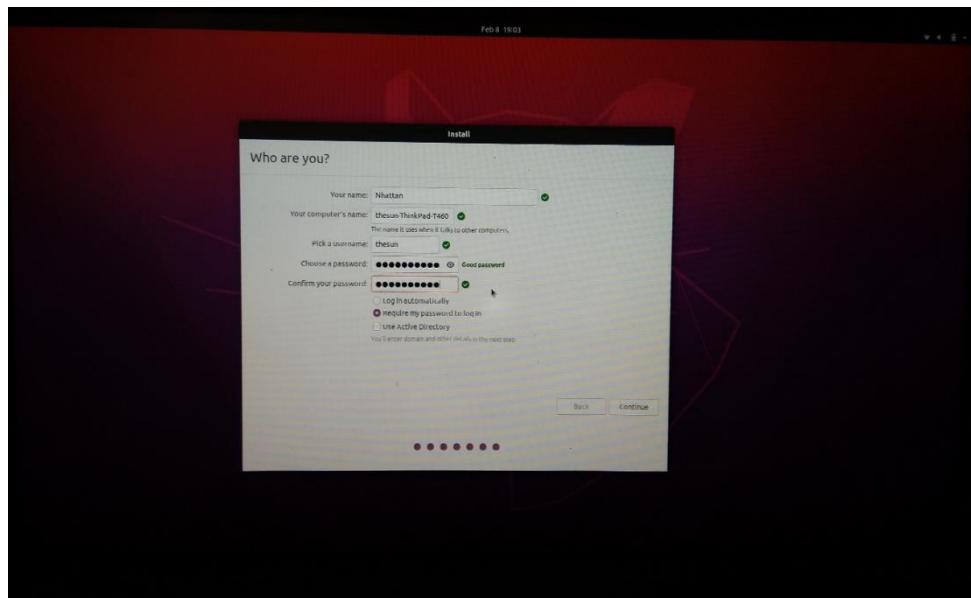
Có thể chọn sử dụng thử Ubuntu (Try Ubuntu) trước khi cài đặt chính thức (Install Ubuntu) như Hình 2 – 5.

Có các kiểu cài đặt Ubuntu vào ổ đĩa:

- Cài đặt song song: Sử dụng trình cài đặt Ubuntu để phân chia ổ cứng và tạo phân vùng riêng cho Ubuntu cùng ổ đĩa với hệ điều hành trước đó.
- Cài đặt thay thế: Cài đặt Ubuntu đè lên hệ điều hành hiện có và xóa toàn bộ dữ liệu trên ổ cứng trước khi cài đặt Ubuntu.
- Cài đặt ở phân vùng khác hệ điều hành trước đó.



**Hình 2 - 6. Tạo tài khoản đăng nhập vào máy tính Ubuntu**



**Hình 2 - 7. Hoàn thành bước tạo tài khoản Ubuntu**

Sau khi cài đặt hoàn tất, cần tạo tài khoản đăng nhập (Hình 2 – 6 và Hình 2 – 7) Ubuntu sẽ yêu cầu khởi động lại máy. Khi ấy cần bấm vào ‘Restart Now’. Sau đó trên màn hình sẽ có thông báo của Ubuntu nhắc tháo USB boot ra.



**Hình 2 - 8. Màn hình sau khi khởi động lại máy**

Sau khi cài đặt Ubuntu xong như Hình 2 - 8, cần cập nhật các gói phần mềm, đây là một cách để đảm bảo rằng máy tính có các phiên bản mới nhất của các ứng dụng, thư viện và bảo mật trên hệ thống. Cập nhật gói cũng có thể giúp khắc phục các lỗi, cải thiện hiệu suất và tương thích với các thiết bị khác.

Có thể cập nhật gói trong Ubuntu bằng cách sử dụng các lệnh trong terminal. Terminal là một ứng dụng cho phép chạy các lệnh dòng lệnh trên LINUX, bao gồm Ubuntu. Để mở terminal trên Ubuntu, có thể làm theo một trong các cách sau:

- Nhấn tổ hợp phím Ctrl + Alt + T2.
- Nhấn phím Super (hay còn gọi là phím Windows), sau đó gõ terminal và chọn biểu tượng terminal.
- Nhấn Alt + F2, trong hộp thoại Run a Command, gõ gnome-terminal.

Lệnh *sudo get update* và *sudo get upgrade* là 2 lệnh dùng để cập nhật các gói phần mềm trên Ubuntu. Lệnh *sudo get update* sẽ tải về thông tin về các phiên bản mới nhất của các gói từ các nguồn được cấu hình trong tệp /etc/apt/sources.list. Lệnh *sudo get upgrade* sẽ cài đặt các phiên bản mới nhất của tất cả các gói đã được cài đặt trên hệ thống từ các nguồn đó. Lệnh *sudo get upgrade* cũng có thể cài đặt các gói mới nếu cần thiết để đáp ứng các phụ thuộc, nhưng sẽ không xóa các gói hiện có.

Một số điểm cần lưu ý khi sử dụng hai lệnh này là:

- Nên chạy lệnh `sudo get update` trước khi chạy lệnh `sudo get upgrade` để đảm bảo rằng có danh sách cập nhật các gói từ internet.
- Cần có quyền sudo (superuser do) để chạy hai lệnh này, nghĩa là phải nhập mật khẩu của người dùng có quyền quản trị hệ thống.
- Nên sao lưu dữ liệu quan trọng trước khi cập nhật các gói, vì có thể có một số lỗi hoặc xung đột xảy ra trong quá trình cập nhật.

## 2.2 Cài đặt phần mềm

### 2.2.1 Cài đặt ROS

ROS dựa trên những nỗ lực của các nhà nghiên cứu từ Phòng thí nghiệm Trí tuệ nhân tạo Stanford và được phát triển thêm tại Willow Garage. Từ đó ROS tiếp tục lan rộng khắp thế giới robot và trở thành một phần mềm trung gian robot mã nguồn mở quan trọng. Mục tiêu chính của Hệ điều hành Robot là đơn giản hóa việc tái sử dụng mã khi phát triển các ứng dụng cho robot dịch vụ. Do đó, nó cung cấp các công cụ và thư viện để thao tác, mô phỏng và trực quan hóa phần cứng robot, giao tiếp giữa các tiến trình, quản lý gói, hơn mươi một nghìn gói duy nhất và nhiều hơn nữa. Vì các gói cốt lõi ROS được cấp phép theo tiêu chuẩn BSD và các gói khác có giấy phép cho phép tương tự, đó là những lựa chọn phù hợp cho quá trình phát triển sản phẩm thương mại.



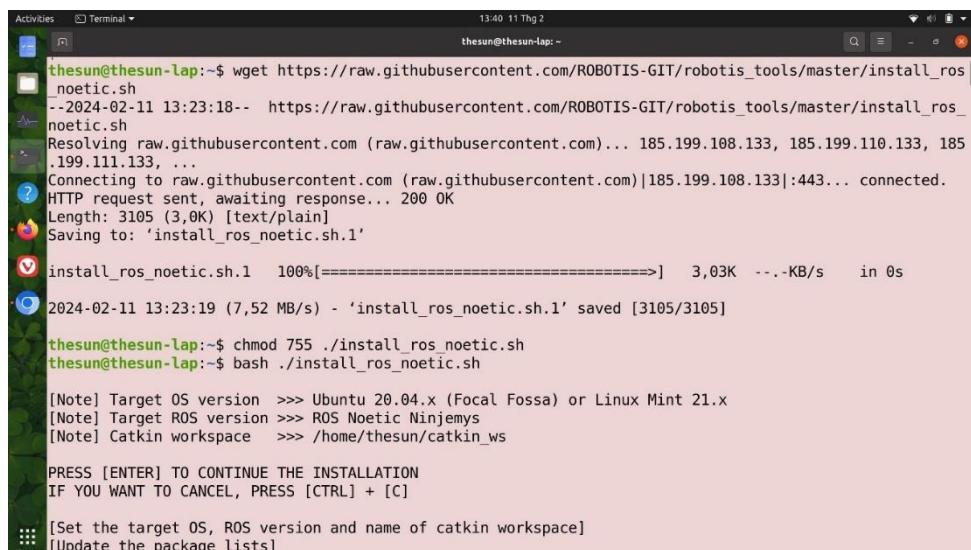
Hình 2 - 9. Tóm lược kiến trúc ROS

(Nguồn: ROS)

ROS được thiết kế theo mô-đun vì hầu hết các hệ thống robot được chia thành nhiều thành phần khác nhau. Do đó, kiến trúc ROS chung có cấu trúc đồ thị bao gồm

nhiều node ROS như thể hiện trong Hình 2 - 9, được kết nối thông qua các node được gọi là topic. Các node đại diện cho một quá trình duy nhất đang chạy và các chủ đề có thể được hiểu là một message. ROS không phải là một hệ điều hành theo nghĩa truyền thống như Windows hay LINUX, mà là một khung phần mềm giúp đơn giản hóa việc xây dựng các ứng dụng robot phức tạp và mạnh mẽ.

Theo truyền thống của ROS, các phiên bản này được đặt tên theo các loại rùa và ở đồ án này em sử dụng phiên bản ROS Noetic Ninjemys. ROS Noetic Ninjemys là bản phân phối ROS thứ mười ba. Nó được phát hành vào ngày 23 tháng 5 năm 2020 và được tải về bằng các lệnh trong Hình 2 – 10.



```
Activities Terminal 13:40 11 Thg 2
thesun@thesun-lap:~$ wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis_tools/master/install_ros_noetic.sh
--2024-02-11 13:23:18-- https://raw.githubusercontent.com/ROBOTIS-GIT/robotis_tools/master/install_ros_noetic.sh
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3105 (3.0K) [text/plain]
Saving to: 'install_ros_noetic.sh.1'

install_ros_noetic.sh.1 100%[=====] 3,03K --.-KB/s in 0s
2024-02-11 13:23:19 (7.52 MB/s) - 'install_ros_noetic.sh.1' saved [3105/3105]

thesun@thesun-lap:~$ chmod 755 ./install_ros_noetic.sh
thesun@thesun-lap:~$ bash ./install_ros_noetic.sh

[Note] Target OS version >>> Ubuntu 20.04.x (Focal Fossa) or Linux Mint 21.x
[Note] Target ROS version >>> ROS Noetic Ninjemys
[Note] Catkin workspace >>> /home/thesun/catkin_ws

PRESS [ENTER] TO CONTINUE THE INSTALLATION
IF YOU WANT TO CANCEL, PRESS [CTRL] + [C]

[Set the target OS, ROS version and name of catkin workspace]
[Update the package lists]
```

**Hình 2 - 10. Cài đặt ROS**

ROS cung cấp một tập hợp các công cụ, thư viện và quy ước cho phép các nhà phát triển robot:

- Chia nhỏ các tác vụ thành các thành phần độc lập: Mỗi thành phần trong ROS có thể được phát triển và thử nghiệm riêng biệt, sau đó tích hợp lại với nhau để tạo thành một hệ thống hoàn chỉnh.
- Tái sử dụng mã: Nhiều thành phần của ROS là mã nguồn mở, có nghĩa là các nhà phát triển có thể tái sử dụng mã hiện có thay vì phải viết lại từ đầu.

- Giao tiếp giữa các robot và các thành phần khác nhau: ROS cung cấp các công cụ cho phép các robot giao tiếp với nhau, cũng như với các cảm biến, bộ truyền động và các hệ thống khác.

### 2.2.2 Cài đặt Gazebo

Gazebo là một phần mềm mô phỏng robot trong thế giới mở, cho phép người dùng thiết kế, xây dựng, kiểm tra và cải thiện các hệ thống robot phức tạp. Gazebo có những tính năng chính sau:

- Mô phỏng động: Gazebo sử dụng nhiều công cụ vật lý hiệu suất cao để mô phỏng chính xác các đối tượng, lực, va chạm, ma sát và động lực học của robot và môi trường.
- Đồ họa 3D nâng cao: Gazebo sử dụng OGRE để tạo ra các môi trường và hiển thị các hình dạng, kết cấu, ánh sáng, bóng tối thực tế.
- Cảm biến và tiếng ồn: Gazebo có thể mô phỏng các cảm biến phổ biến như camera, lidar, IMU, GPS, v.v. và thêm tiếng ồn.
- Nhiều mô hình robot: Gazebo có sẵn nhiều mô hình robot sẵn có, bao gồm PR2, Pioneer2 DX, iRobot Create, TurtleBot,...
- API và plugin: Gazebo cung cấp một API mạnh mẽ cho người dùng để tùy chỉnh và điều khiển robot, cảm biến và môi trường. Gazebo cũng hỗ trợ việc viết các plugin để mở rộng các chức năng của nó.
- Mạng, đám mây và dòng lệnh: Gazebo có thể chạy trên nhiều máy tính thông qua mạng, cho phép người dùng chia sẻ và cộng tác trên các mô phỏng.

Khi cài đặt ROS, Gazebo được mặc định cài đặt kèm theo, nhưng vẫn có thể cài đặt riêng biệt với lệnh: `sudo apt-get install gazebo11`, với phiên bản mới nhất là Gazebo 11.

### 2.2.3 Cài đặt RVIZ

RVIZ là một phần mềm trực quan hóa 3D cho robot, cảm biến và thuật toán. Nó cho phép xem dữ liệu từ Gazebo (nếu đang mô phỏng) hoặc dữ liệu thế giới thực (nếu

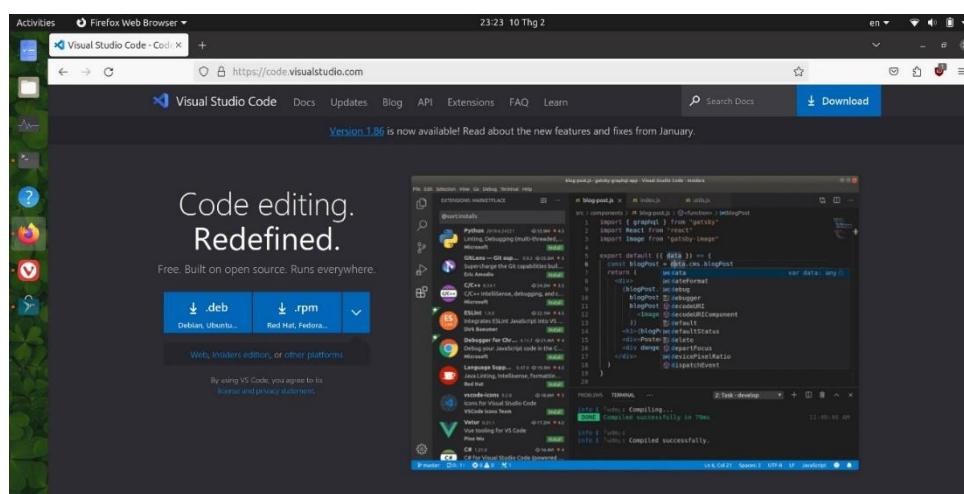
không sử dụng Gazebo mà là một robot thực). RVIZ là một công cụ hữu ích trong hệ thống ROS, giúp giám sát trực tiếp và trực quan 3 chiều của các loại cảm biến và hoạt động của robot. Ta có thể truy xuất các loại tín hiệu cung cấp trong môi trường ROS, ví dụ như cảm biến Lidar, IMU, GPS và nhiều hơn nữa. Cũng như Gazebo. RVIZ cũng được cài đặt kèm theo ROS.

Để cài đặt RVIZ trong ROS Noetic, cần sử dụng lệnh sau: *sudo apt-get install ros-noetic-rviz*.

#### 2.2.4 Cài đặt VS Code

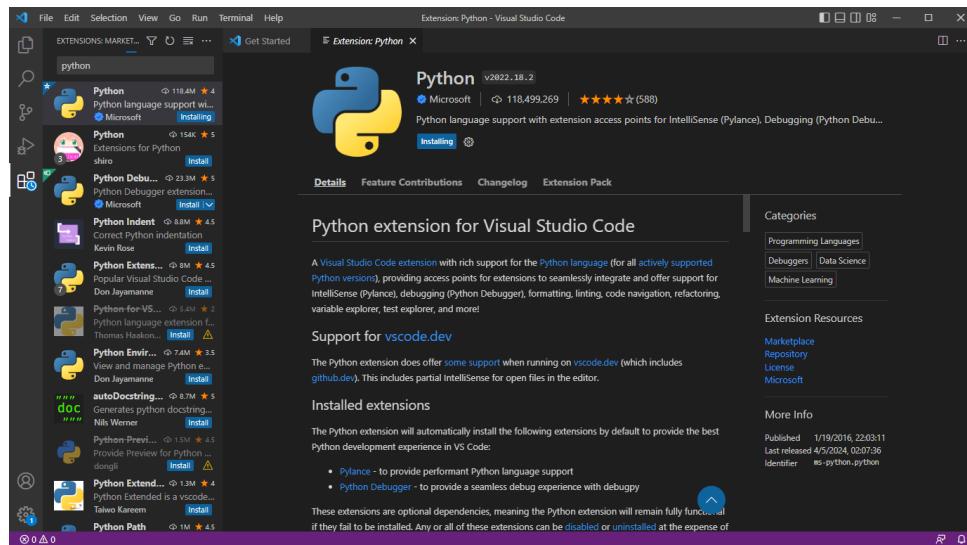
Visual Studio Code (VS Code) là một trình soạn thảo mã nguồn miễn phí, mã nguồn mở được phát triển bởi Microsoft. Nó có sẵn cho Windows, macOS và LINUX. VS Code là một công cụ mạnh mẽ và linh hoạt được sử dụng bởi các nhà phát triển trên toàn thế giới để viết mã cho nhiều ngôn ngữ lập trình khác nhau. Các tính năng nổi bật của VS Code:

- Hỗ trợ nhiều ngôn ngữ.
- Tự động hoàn thành mã.
- Kiểm tra cú pháp.
- Gỡ lỗi.
- Phát triển web.
- Nhiều mở rộng (EXTENSIONS).



Hình 2 - 11. Cài đặt file .deb của VS Code

Để cài đặt VS Code, cần tải về file .deb như Hình 2 - 11, sau đó thì ấn vào để cài đặt phần mềm vào máy như Hình 2 - 12.



Hình 2 - 12. Cài đặt phần mở rộng cho ngôn ngữ Python trên VS Code

### 2.3 Thiết lập môi trường làm việc cho ROS

Phải lấy nguồn tập lệnh này trong mọi thiết bị đầu cuối bash mà ta sử dụng ROS: `source /opt/ros/noetic/setup.bash`.

### 2.4 Cài đặt các phụ thuộc để xây dựng các gói trong ROS

Để tạo và quản lý không gian làm việc ROS riêng, có nhiều công cụ và yêu cầu khác nhau được phân phối riêng. Ví dụ: `rosinstall` là một công cụ dòng lệnh được sử dụng thường xuyên cho phép dễ dàng tải xuống nhiều cây nguồn cho các gói ROS bằng một lệnh. Để cài đặt công cụ này và các phụ thuộc khác để xây dựng các gói ROS, cần chạy: `sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential`.

### 2.5 Gói Turtlebot

#### 2.5.1 Giới thiệu về Turtlebot

TurtleBot là một bộ dụng cụ robot cá nhân, giá rẻ với phần mềm mã nguồn mở. Được tạo ra bởi Melonee Wise và Tully Foote tại Willow Garage vào tháng 11 năm

2010. Nó được thiết kế chủ yếu cho mục đích nghiên cứu và giảng dạy trong lĩnh vực robot di động và trí tuệ nhân tạo (AI). Turtlebot đã trở thành một nền tảng phổ biến trong cộng đồng nghiên cứu robotics và được sử dụng rộng rãi tại các trường đại học, viện nghiên cứu và công ty công nghệ trên toàn thế giới.

Turtlebot có thiết kế đơn giản nhưng linh hoạt, bao gồm một khung gầm di động được trang bị bánh xe, bộ xử lý, cảm biến và phần mềm mã nguồn mở. Phần cứng của Turtlebot thường được tích hợp với một máy tính nhúng (như Raspberry Pi) hoặc một máy tính xách tay để cung cấp năng lực xử lý. Phần mềm điều khiển robot được xây dựng trên nền tảng Robot Operating System (ROS), cho phép lập trình và tích hợp các chức năng phức tạp một cách dễ dàng. Ngoài bộ dụng cụ, người dùng có thể tải xuống Bộ công cụ phát triển phần mềm (SDK) của TurtleBot từ wiki ROS.

Là một nền tảng robot di động dành cho người mới bắt đầu, TurtleBot sở hữu nhiều khả năng tương tự như các nền tảng robot lớn hơn của công ty, chẳng hạn như PR2.

### **2.5.2 Đặc điểm nổi bật của Turtlebot**

Một trong những điểm nổi bật của Turtlebot là khả năng tùy chỉnh cao. Người dùng có thể lắp đặt thêm các cảm biến như camera, laser scanner, cảm biến siêu âm, v.v. để mở rộng khả năng nhận thức môi trường của robot. Ngoài ra, Turtlebot cũng hỗ trợ việc gắn các công cụ và phụ kiện khác nhau, tùy thuộc vào mục đích sử dụng cụ thể. Các đặc điểm nổi bật của Turtlebot có thể liệt kê chi tiết như sau:

#### **1. Dễ sử dụng:**

- Turtlebot được thiết kế với phần cứng và phần mềm đơn giản, dễ dàng cài đặt và sử dụng. Phù hợp cho người mới bắt đầu học về robot di động.
- Có nhiều tài liệu hướng dẫn, ví dụ mã và cộng đồng hỗ trợ trực tuyến.

#### **2. Khả năng mở rộng:**

- Turtlebot có nhiều khe cắm mở rộng cho phép tích hợp thêm các cảm biến, bộ truyền động và các thiết bị khác. Phù hợp cho các dự án nghiên cứu và phát triển robot.

- Nền tảng ROS hỗ trợ nhiều thư viện và công cụ cho phép phát triển các ứng dụng robot phức tạp.

**3. Giá cả phải chăng:**

- Turtlebot có giá thành tương đối rẻ so với các robot di động khác trên thị trường. Phù hợp cho các mục đích giáo dục và đào tạo.
- Có nhiều phiên bản Turtlebot với mức giá khác nhau phù hợp với nhu cầu và ngân sách của người sử dụng.

**4. Kích thước nhỏ gọn:**

- Turtlebot có kích thước nhỏ gọn, dễ dàng di chuyển và cất giữ.
- Phù hợp cho sử dụng trong môi trường hạn chế.

**5. Tính linh hoạt:**

- Turtlebot có thể di chuyển theo nhiều hướng khác nhau, bao gồm tiến, lùi, trái, phải và xoay. Phù hợp cho nhiều ứng dụng robot khác nhau.
- Có thể điều khiển bằng nhiều cách khác nhau, bao gồm điều khiển từ xa, joystick và giao diện lập trình.

Ngoài ra, Turtlebot còn có một số ưu điểm khác như:

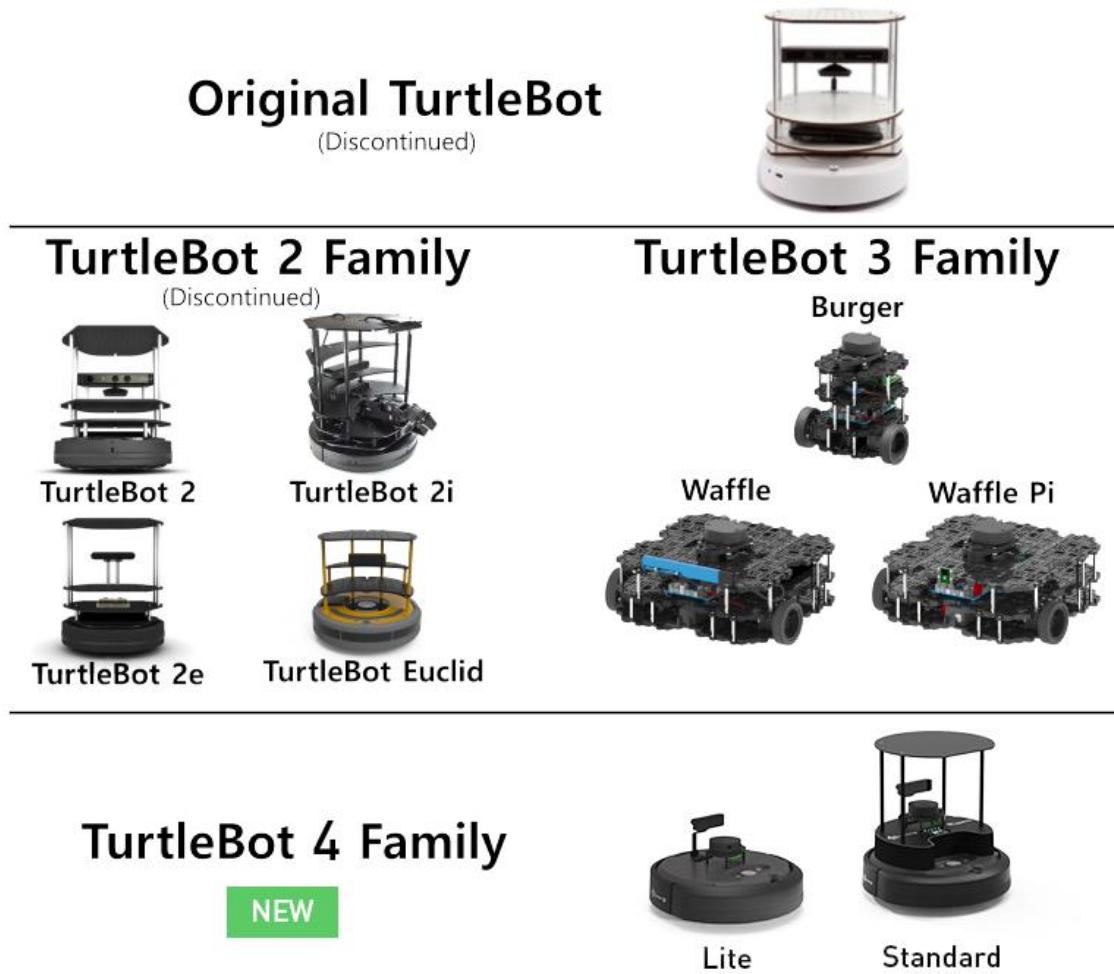
- Có nhiều phiên bản với các tính năng khác nhau, phù hợp với nhu cầu sử dụng đa dạng.
- Cộng đồng người dùng lớn và hoạt động tích cực. Được hỗ trợ bởi nhiều nhà cung cấp dịch vụ và sản phẩm robot.

### **2.5.3 Công nghệ lõi của Turtlebot**

Công nghệ cốt lõi của TurtleBot là SLAM và ACML, giúp nó phù hợp với các robot dịch vụ gia đình. TurtleBot có thể chạy các thuật toán SLAM để xây dựng bản đồ và di chuyển xung quanh phòng. Ngoài ra, nó có thể được điều khiển từ xa bằng máy tính xách tay, tay cầm chơi game hoặc điện thoại thông minh Android. TurtleBot cũng có thể theo chân người khi họ đi bộ trong phòng. Bên cạnh đó, nó còn đi kèm với các khả năng điều hướng bằng tay ROS, một phụ kiện tay robot được phát triển để thao tác với các vật thể.

#### 2.5.4 Quá trình phát triển của Turtlebot

Hiện nay có 4 thế hệ TurtleBots như hình 2 - 13, lịch sử phát triển như sau:



Hình 2 - 13. Các thế hệ TurtleBots

(Nguồn: UCR-Robotics)

- TurtleBot1 bao gồm đế iRobot Create, bộ pin 3.000mAh, bảng nguồn TurtleBot với con quay hồi chuyển, cảm biến Kinect, máy tính xách tay Asus 1215N với bộ xử lý lõi kép và bộ lắp phần cứng gắn mọi thứ lại với nhau và bổ sung thêm các cảm biến trong tương lai. TurtleBot đầu tiên được tạo ra tại Willow Garage bởi Melonee Wise và Tully Foote vào tháng 11 năm 2010.
- TurtleBot2 bao gồm đế Yujin Kobuki, bộ pin 2.200mAh, cảm biến Kinect, máy tính xách tay Asus 1215N với bộ xử lý lõi kép, bộ sạc nhanh, đế sạc và

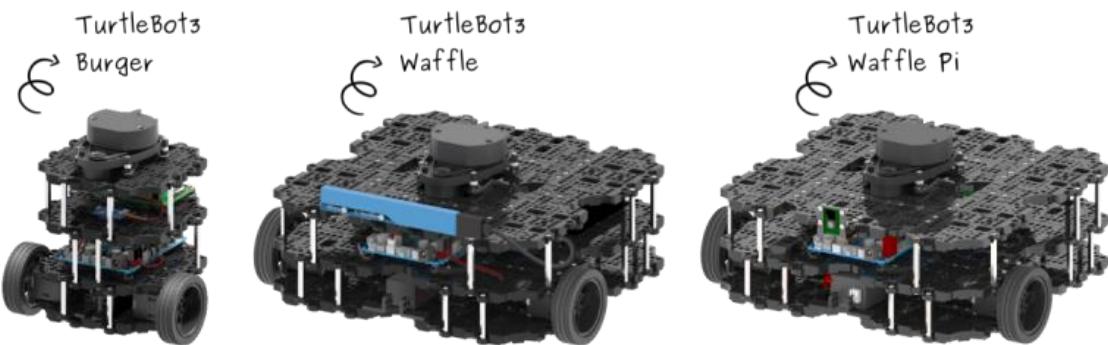
bộ gắn phần cứng gắn mọi thứ lại với nhau và bổ sung thêm các cảm biến trong tương lai. Turtlebot2 được phát hành vào tháng 10 năm 2012.

- TurtleBot3 được tạo thành từ các tấm mô-đun mà người dùng có thể tùy chỉnh hình dạng. Có sẵn ba loại: Waffle cỡ nhỏ và Waffle cỡ trung bình, Waffle Pi. TurtleBot3 bao gồm một đế, hai động cơ Dynamixel, bộ pin 1.800mAh, LIDAR 360 độ, một máy ảnh (+ camera RealSense cho bộ bánh quế, + Camera Raspberry Pi cho bộ bánh quế Pi), một SBC (máy tính bo mạch đơn: Raspberry PI 3 và Intel Joule 570x) và bộ lắp phần cứng gắn mọi thứ lại với nhau và bổ sung thêm các cảm biến trong tương lai. Turtlebot3 được phát hành vào tháng 5 năm 2017.

## 2.5.5 Gói Turtlebot3

### 2.5.5.1 Giới thiệu về gói Turtlebot3

TurtleBot 3 được tạo ra với sự hợp tác giữa Open Robotics và ROBOTIS. TurtleBot3 là một phần của loạt TurtleBot (Hình 2 - 14) và được biết đến như một robot nền tảng tiêu chuẩn ROS.

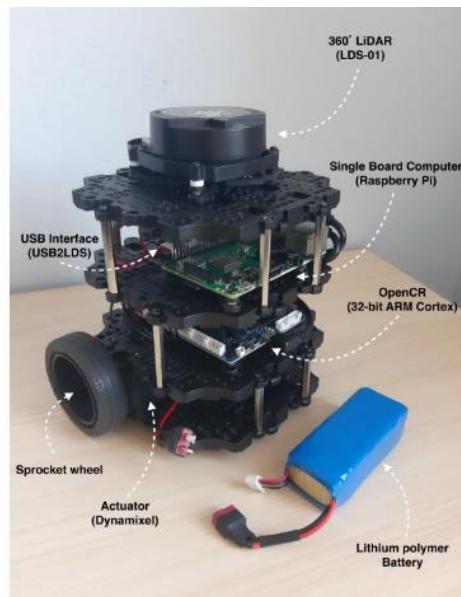


Hình 2 - 14. Các mẫu robot trong gói TurtleBot3

(Nguồn: ROS)

Nó được thiết kế để dễ lắp ráp và sử dụng, có thể lập trình và khá rẻ so với các nền tảng phần cứng robot khác. TurtleBot3 bao gồm một kiến trúc giống như bánh

mì kẹp thịt bốn lớp, trong đó mỗi lớp làm bằng nhựa chịu trách nhiệm cho một chức năng cụ thể như hình 2 - 15.



**Hình 2 - 15. Các thành phần của Turtlebot3 Burger**

(Nguồn: ROS)

Robot có thể mở rộng cho các mô-đun tiếp theo không được cung cấp trong gói tiêu chuẩn. Lớp đầu tiên chứa LiDAR 360 độ, lớp thứ hai là máy tính bảng đơn (Raspberry Pi 3 Model B) và giao diện USB (USB2LDS), lớp thứ ba là mô-đun điều khiển mã nguồn mở cho ROS (ARM Cortex-M7 32 bit) và trên lớp cuối cùng có hai bộ truyền động robot (Dynamixel), pin Li-Po 11V, hai bánh xe và lốp xe. Việc lắp ráp robot mất khoảng năm giờ làm việc vì tài liệu được cung cấp khá chính xác. Các lựa chọn thay thế có khả năng và giá cả tương tự như robot di động E-puck2, robot Sparki, iCub hoặc ArduPilot.

#### **2.5.5.2 Các mẫu robot trong gói Turtlebot3**

Gói Turtlebot3 phô biến nhất với 2 mẫu robot đó là Burger và Waffle; chúng không chỉ khác biệt về hình dáng mà còn chênh lệch nhau các thông số được thể hiện trong Hình 2 - 16 dưới đây.

Items	Burger	Waffle Pi
Maximum translational velocity	0.22 m/s	0.26 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)	1.82 rad/s (104.27 deg/s)
Maximum payload	15kg	30kg
Size (L x W x H)	138mm x 178mm x 192mm	281mm x 306mm x 141mm
Weight (+ SBC + Battery + Sensors)	1kg	1.8kg
Threshold of climbing	10 mm or lower	10 mm or lower
Expected operating time	2h 30m	2h
Expected charging time	2h 30m	2h 30m
SBC (Single Board Computers)	Raspberry Pi	Raspberry Pi
MCU	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Remote Controller	-	RC-100B + BT-410 Set (Bluetooth 4, BLE)
Actuator	XL430-W250	XM430-W210
LDS(Laser Distance Sensor)	360 Laser Distance Sensor <a href="#">LDS-01</a> or <a href="#">LDS-02</a>	360 Laser Distance Sensor <a href="#">LDS-01</a> or <a href="#">LDS-02</a>
Camera	-	Raspberry Pi Camera Module v2.1
IMU	Gyroscope 3 Axis Accelerometer 3 Axis	Gyroscope 3 Axis Accelerometer 3 Axis
Power connectors	3.3V / 800mA 5V / 4A 12V / 1A	3.3V / 800mA 5V / 4A 12V / 1A
Expansion pins	GPIO 18 pins Arduino 32 pin	GPIO 18 pins Arduino 32 pin
Peripheral	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4
DYNAMIXEL ports	RS485 x 3, TTL x 3	RS485 x 3, TTL x 3
Audio	Several programmable beep sequences	Several programmable beep sequences
Programmable LEDs	User LED x 4	User LED x 4
Status LEDs	Board status LED x 1 Arduino LED x 1 Power LED x 1	Board status LED x 1 Arduino LED x 1 Power LED x 1

**Hình 2 - 16. So sánh thông số Turtlebot3 Burger và Waffle**

(Nguồn: ROBOTIS e-Manual)

Ở đê tài này, em sử dụng mẫu Turtlebot3 Waffle với tốc độ di chuyển tối đa là 0.26 m/s.

### 2.5.5.3 Tạo không gian làm việc giữa Turtlebot3 và ROS

Để đưa Turtlebot3 vào dự án ROS, cần xây dựng môi trường làm việc với các bước sau:

- Tạo không gian làm việc: `mkdir -p ~/dacn/src`. Câu lệnh trên sẽ tạo 1 thư mục tên ‘dacn’ và 1 thư mục nhỏ tên ‘src’.
- Vào thư mục ‘src’ và tải gói mô phỏng của Turtlebot3 trên GitHub: `git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git`.

- Trở ra thư mục ‘dacn’ và xây dựng các gói phần mềm trong Robot Operating System (ROS) bằng một hệ thống build dựa trên CMake, giúp đơn giản hóa quá trình xây dựng các chương trình ROS, thư viện và các gói khác: *catkin\_make*.
- Chạy lệnh: *source devel/setup.bash* để thiết lập môi trường làm việc với một không gian làm việc ROS cụ thể với *devel/setup.bash* là tập lệnh chịu trách nhiệm thiết lập các biến môi trường và đường dẫn cần thiết để không gian làm việc ROS.

Về sau, khi chạy mô phỏng cần truy cập vào không gian làm việc, thiết lập môi trường làm việc và chỉ định mô hình cụ thể với lệnh: *export TURTLEBOT3\_MODEL=model*, trong đó *model* bao gồm “waffle”, “waffle”, và “waffle\_pi”.

#### 2.5.5.4 Tạo không gian làm việc để phát hiện biển báo tốc độ

Truy cập thư mục *src* đã tạo trong *dacn*. Đầu tiên tạo một thư mục có tên *yolov8*: *catkin\_create\_pkg yolov8 rospy roscpp* và các gói thư viện từ Github: *git clone https://github.com/ultralytics/ultralytics*. Tiếp tục chạy lệnh để tải về: *pip install ultralytics*. Cuối cùng là xây dựng các gói trong thư mục *yolov8* vừa tạo: *catkin\_make*.

Tiếp đến tạo hai chương trình để thực hiện mô phỏng:

- *detect.py*: chương trình mô phỏng phát hiện biển báo tốc độ.
- *8control.py*: chương trình mô phỏng phát hiện biển báo tốc độ điều khiển tốc độ.

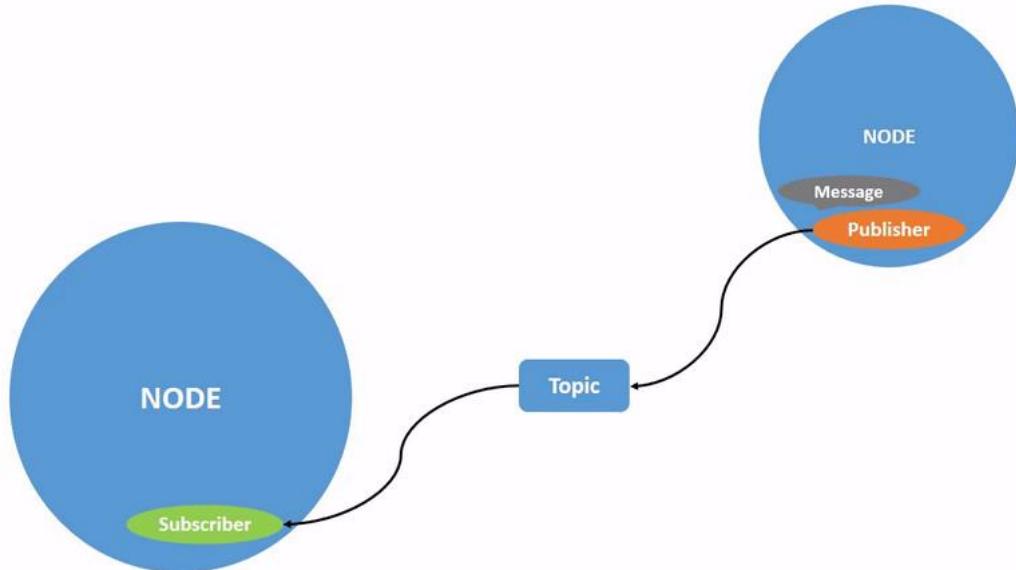
## CHƯƠNG 3. CẤU TRÚC CỦA ROS

### 3.1 Node

Trong ROS một Node là một quá trình thực hiện một tác vụ cụ thể. Các Node có thể giao tiếp với nhau bằng cách sử dụng publish/subscribe (truyền/nhận liên tục) hoặc Service (máy khách/máy chủ).

Ví dụ, một con robot thường có các node như Laser scanner, Camera, 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.

Khi thực thi, mỗi node sẽ phải đăng ký với Master. Trong ROS, Master cung cấp các dịch vụ đặt tên và đăng ký (naming and registration) cho các node còn lại trong hệ thống ROS. Hình 3 - 1 trình bày việc publish (truyền) và subscribe (nhận) của các node cũng như các topic và service. ROS Master được gọi bằng lệnh `roscore`.



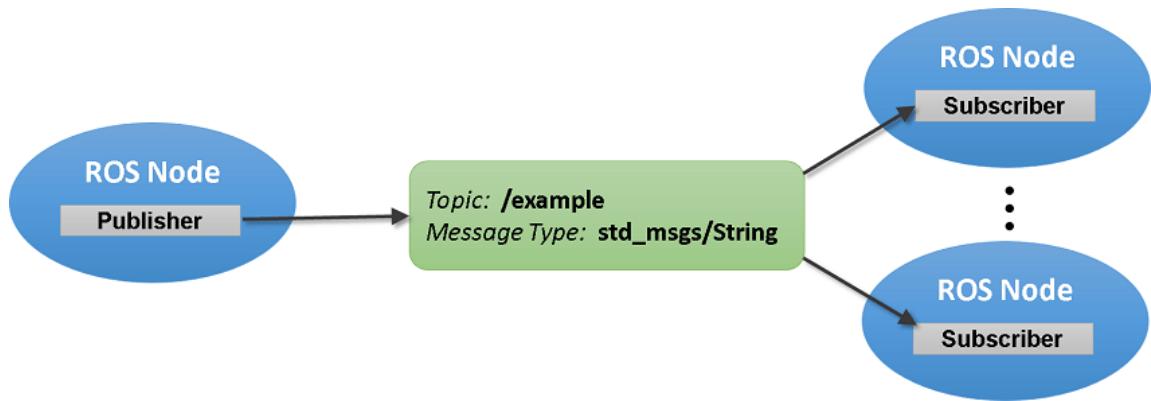
Hình 3 - 1. Hai node trong ROS đang giao tiếp với nhau

(Nguồn: Open Robotics)

### 3.2 Topic

Trong ROS, Topic là một hệ thống truyền tải dữ liệu dựa trên một hệ thống publish và subscribe. Một hoặc nhiều node có thể publish (truyền) dữ liệu cho một topic và một hoặc nhiều node khác có thể subscribe (nhận) để nhận dữ liệu từ topic đó.

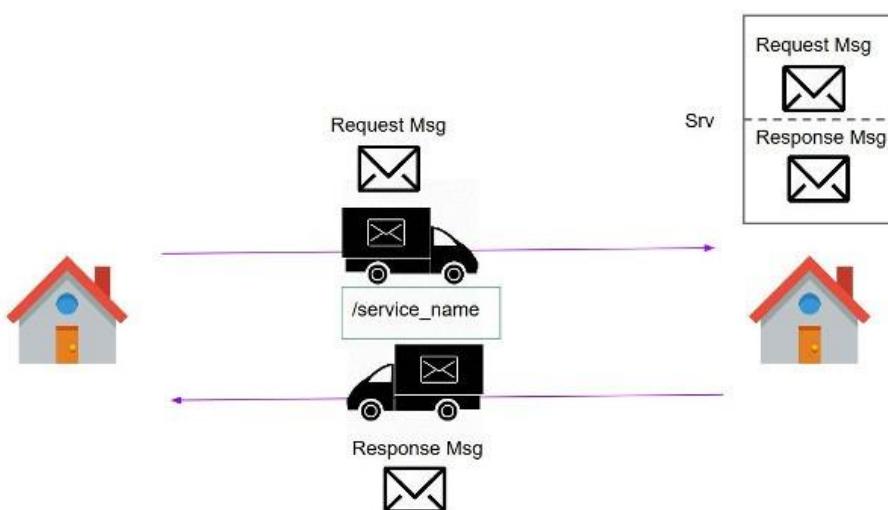
Ví dụ như Hình 3 - 2, một node có thể publish (truyền) dữ liệu từ cảm biến, và một node khác có thể subscribe (nhận) dữ liệu từ topic đó để xử lý hoặc thực hiện một hành động cụ thể. Topic giúp các node trong ROS có thể giao tiếp và tương tác với nhau một cách linh hoạt và hiệu quả.



Hình 3 - 2. Ví dụ về topic trong ROS

(Nguồn: Open Robotics)

### 3.3 Message



Hình 3 - 3. Ví dụ về message trong ROS

(Nguồn: Open Robotics)

Message là một cấu trúc dữ liệu mà các node sử dụng để trao đổi với nhau thông qua topic như Hình 3 - 3. Message bao gồm sự kết hợp của nhiều kiểu dữ liệu. Các node tương tác với nhau bằng cách send và receive ROS message.

Các gói thư viện của ROS đã bao gồm nhiều kiểu Message cơ bản như *std\_msgs*, *sensor\_msgs*. Nếu muốn xây dựng một loại message riêng, nó sẽ được đặt trong *package\_name/msg /myMessageType.msg*.

### 3.4 Publish

Publish là một hành động trong hệ thống giao tiếp dựa trên mô hình publish/subscribe. Một node khi muốn truyền dữ liệu ra ngoài sẽ thực hiện hành động publish. Dữ liệu này được truyền thông qua một Topic. Ví dụ, một node có thể publish (truyền) dữ liệu từ cảm biến, và một node khác có thể subscribe (nhận) dữ liệu từ topic đó để xử lý hoặc thực hiện một hành động cụ thể.

Như vậy, publish trong ROS là việc một node truyền dữ liệu ra ngoài thông qua một topic.

### 3.5 Subscribe

Cũng như publish, subscribe là một hành động trong hệ thống giao tiếp dựa trên mô hình publish/subscribe nhưng chức năng của subscribe ngược lại với publish.

Một node khi muốn nhận dữ liệu từ một node khác sẽ thực hiện hành động subscribe. Dữ liệu này được nhận thông qua một Topic. Ví dụ, một node có thể subscribe (nhận) dữ liệu từ một topic mà node khác đã publish (truyền), và sau đó xử lý hoặc thực hiện một hành động cụ thể dựa trên dữ liệu đó.

Như vậy, subscribe trong ROS là việc một node nhận dữ liệu từ một topic.

### 3.6 Cây TF trong ROS

Trong ROS, TF (Transform) là một gói phần mềm giúp theo dõi các chuyển đổi hệ tọa độ qua thời gian. TF cho phép xem về vị trí và hướng của một khung tọa độ (frame) tại một thời điểm cụ thể, và biến đổi các vector, hướng và các đối tượng khác

giữa các khung tọa độ. TF sử dụng một cây dữ liệu để theo dõi tất cả các khung tọa độ trong hệ thống. Cây này được gọi là cây TF, mỗi node trong cây đại diện cho một khung tọa độ và mỗi cạnh đại diện cho mối quan hệ hình học giữa hai khung. Để chạy công cụ này cần dùng lệnh: `rosrun rqt_tf_tree rqt_tf_tree`.

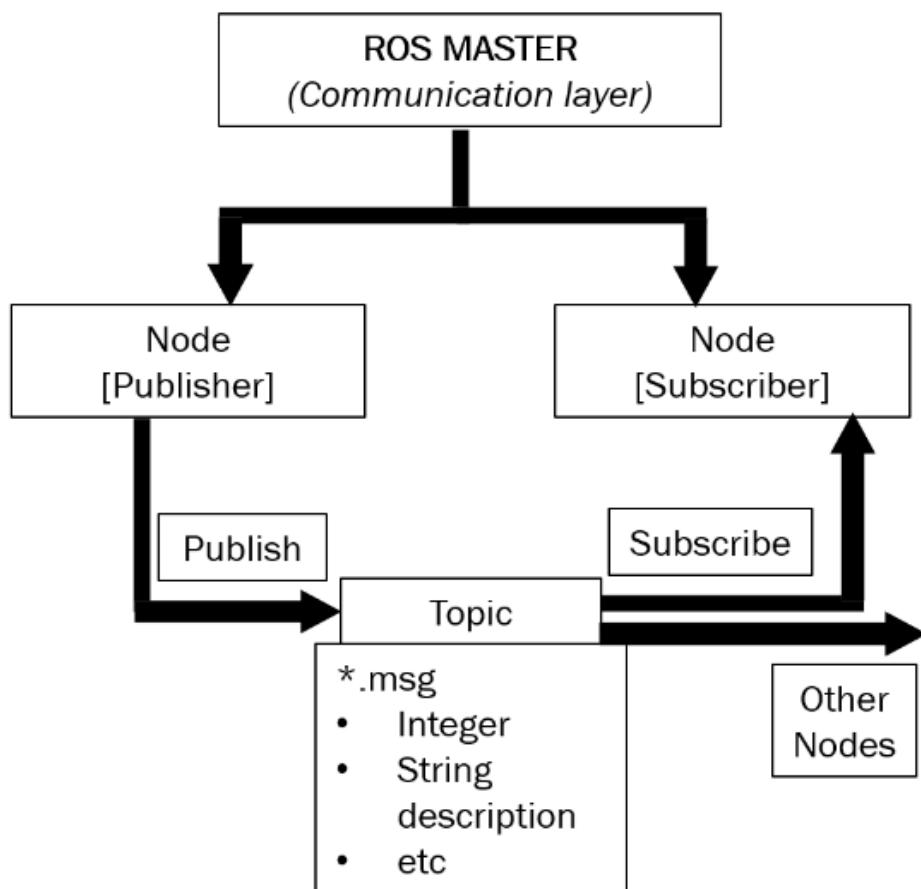
Một số khung tọa độ phổ biến và chức năng của chúng như sau:

- map: Đây thường là khung toàn cục thể hiện bản đồ môi trường mà robot đang hoạt động. Nó thường cố định và không thay đổi theo thời gian.
- odom: Viết tắt của odometry, khung này thường được gắn vào vị trí và hướng ban đầu của robot. Nó thể hiện tư thế (vị trí và hướng) của robot dựa trên dữ liệu đo hình của nó. Trong thời gian ngắn, nó chính xác hơn khung bản đồ nhưng có thể trôi dạt theo thời gian.
- base\_footprint: Khung này thường ở chân robot và tiếp xúc với mặt đất. Nó rất hữu ích cho các hoạt động cần tính đến sự tiếp xúc của robot với mặt đất, chẳng hạn như điều hướng hoặc lập kế hoạch đường đi.
- base\_link: Đây thường là khung tham chiếu chính của robot. Các khung khác như khung dành cho cảm biến hoặc bộ điều khiển được xác định liên quan đến khung này.
- camera\_link: Đây là khung chính cho cảm biến camera. Các khung khác liên quan đến máy ảnh được xác định tương ứng với khung này.
- camera\_deep\_frame & camera\_deep\_optical\_frame: Các khung này lần lượt thể hiện tâm quang học của camera độ sâu và khung tham chiếu quang học.
- camera\_rgb\_frame & camera\_rgb\_optical\_frame: Tương tự, các khung này lần lượt thể hiện tâm quang học của máy ảnh RGB và khung tham chiếu quang học.
- caster\_back\_left\_link, caster\_back\_right\_link: Những khung này thường được sử dụng cho bánh xe bên trái và bên phải của robot truyền động vi sai.
- imu\_link: Khung này được liên kết với cảm biến Đơn vị đo lường quán tính (IMU), cung cấp dữ liệu về tốc độ tuyến tính, vận tốc góc và từ trường (tùy chọn).

- base\_scan: Khung này thường được liên kết với máy quét laser hoặc cảm biến LIDAR gắn trên robot.
- Wheel\_left\_link, Wheel\_right\_link: Các khung này thể hiện bánh xe bên trái và bên phải của robot truyền động vi sai.

### 3.7 Cấu trúc hoàn chỉnh của ROS

Cấu trúc của ROS bao gồm các thành phần chính để đã kể trên, chúng hoạt động một cách chặt chẽ và có kết nối với nhau để hoàn thành một nhiệm vụ chung như Hình 3 - 4. Có thể tóm tắt ngắn gọn lại như sau:



Hình 3 - 4. Chi tiết cấu trúc của ROS

(Nguồn: Open Robotics)

- Nodes: Mỗi node là một quá trình thực hiện một tác vụ cụ thể.
- Topics: Là một hệ thống truyền tải dữ liệu dựa trên mô hình publish/subscribe.

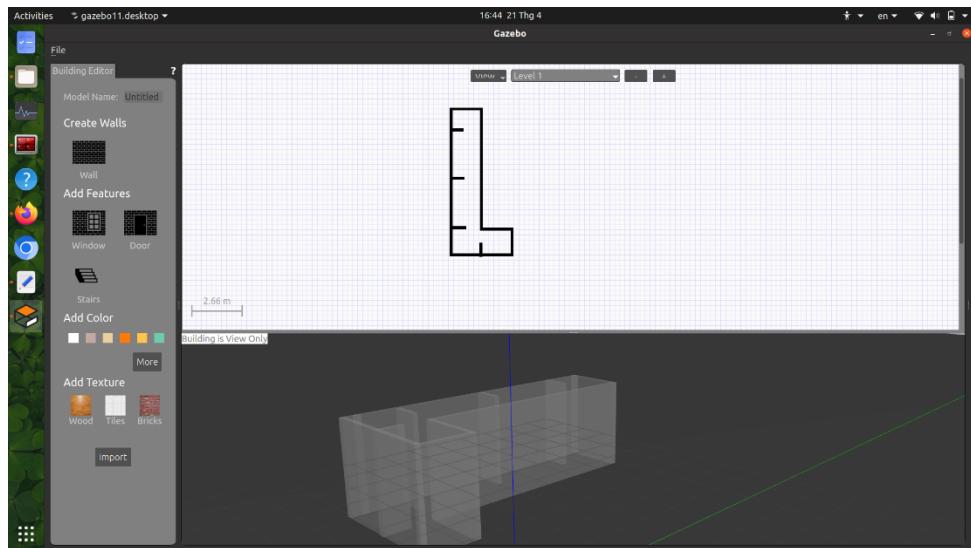
- **Messages:** Là một cấu trúc dữ liệu mà các node sử dụng để trao đổi với nhau, bao gồm sự kết hợp của nhiều kiểu dữ liệu
- **Master:** Cung cấp các dịch vụ đặt tên và đăng ký (naming and registration) cho các node còn lại trong hệ thống ROS. Nó theo dõi việc publish (truyền) và subscribe (nhận) của các node cũng như các topic và service.
- **Services:** Là một hệ thống giao tiếp dựa trên mô hình client/server. Một node (server) cung cấp một dịch vụ và một node khác (client) có thể yêu cầu dịch vụ đó.
- **Parameter Server:** Là một dịch vụ được cung cấp bởi ROS Master cho phép dữ liệu được lưu trữ và truy cập bởi bất kỳ node nào.
- **Publish:** Truyền message thông qua topic.
- **Subscribe:** Nhận message thông qua topic.

## CHƯƠNG 4. XÂY DỰNG MÔI TRƯỜNG MÔ PHỎNG TRONG GAZEBO

### 4.1 Xây dựng tường

Đầu tiên em sẽ xây dựng tường bao bọc xung quanh để tạo không gian di chuyển của TurtleBot như Hình 4 - 1, ở đây em sử dụng chế độ Building Editor trong thẻ Edit của phần mềm Gazebo. Sau đó lưu mô hình trên vào thư mục `dacn/src/turtlebot3_simulations/turtlebot3_gazebo/model` với 1 thư mục tên là `my_world_3` gồm 2 file:

- `model.config`: Đây là tệp cấu hình chứa thông tin meta về mô hình. Tệp này cung cấp cho Gazebo các thông tin cần thiết để nạp mô hình vào môi trường mô phỏng.
- `model.sdf`: Đây là tệp chứa mô tả chi tiết về mô hình theo định dạng SDF. Tệp này mô tả các thuộc tính vật lý, động học và hình ảnh của mô hình.



Hình 4 - 1. Xây dựng tường trong ROS

### 4.2 Xây dựng các biển báo tốc độ

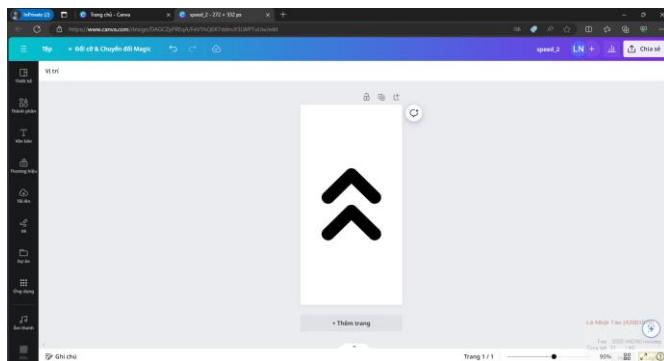
Biển báo tốc độ đóng vai trò quy định cho 3 tốc độ mà robot cần nhận diện và xử lý với quy ước đặt tên như các như Hình 4 - 2, Hình 4 - 3 và Hình 4 - 4:

- Biển báo speed\_1 đại diện cho tốc độ di chuyển chậm: 0.15 m/s.



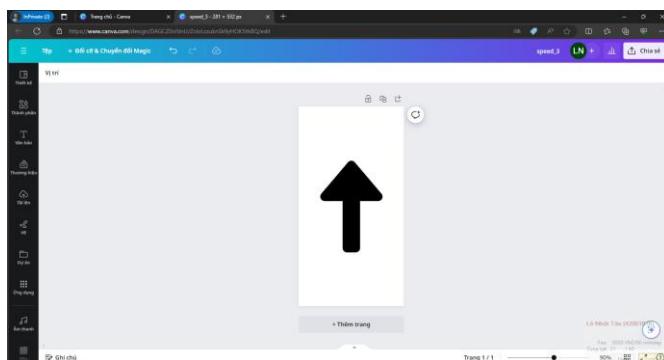
**Hình 4 - 2. Bản thiết kế của biển báo speed\_1**

- Biển báo speed\_2 đại diện cho tốc độ di chuyển trung bình: 0.20 m/s.



**Hình 4 - 3. Bản thiết kế của biển báo speed\_2**

- Biển báo speed\_3 đại diện cho tốc độ di chuyển nhanh: 0.25 m/s.



**Hình 4 - 4. Bản thiết kế của biển báo speed\_3**

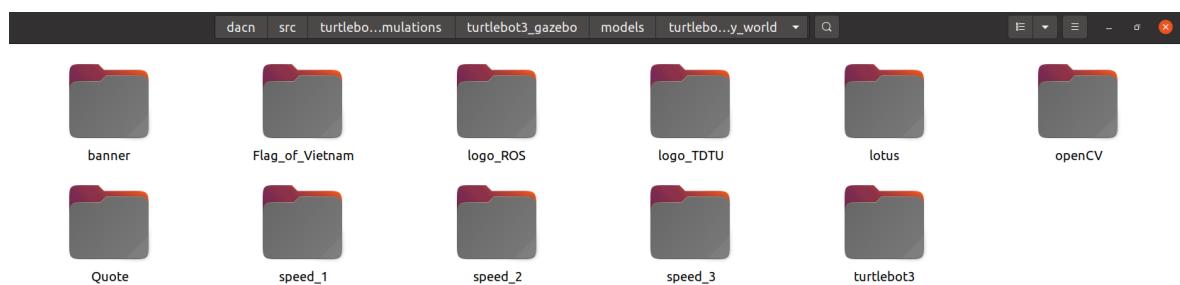
Tất cả các biển báo được thiết kế trên Canva và được đưa vào thư mục *model* cùng với *my\_world\_3*, đặt tên là *turtlebot3\_empty\_world* và phân thành 3 thư mục nhỏ: *speed\_1*, *speed\_2*, *speed\_3* và trong 1 thư mục nhỏ sẽ gồm các tệp và thư mục liên kết với nhau. Lấy ví dụ thư mục *speed\_1* sẽ gồm:

- *model.config*.

- model.sdf.
- Thư mục *materials* gồm thư mục *scripts* (chứa tệp *speed\_1.material*) và thư mục *textures* (chứa hình ảnh của biển báo *speed\_1*).

### 4.3 Xây dựng các biển báo thông tin

Tương tự như các biển báo tốc độ, các biển báo thông tin cũng được thiết kế trên Canva và đưa vào thư mục *model/turtlebot3\_empty\_world* như Hình 4 - 5. Vì vậy *turtlebot3\_empty\_world* sẽ bao gồm các thư mục nhỏ như sau:



Hình 4 - 5. Các biển báo trong môi trường mô phỏng

### 4.4 Xây dựng môi trường mô phỏng

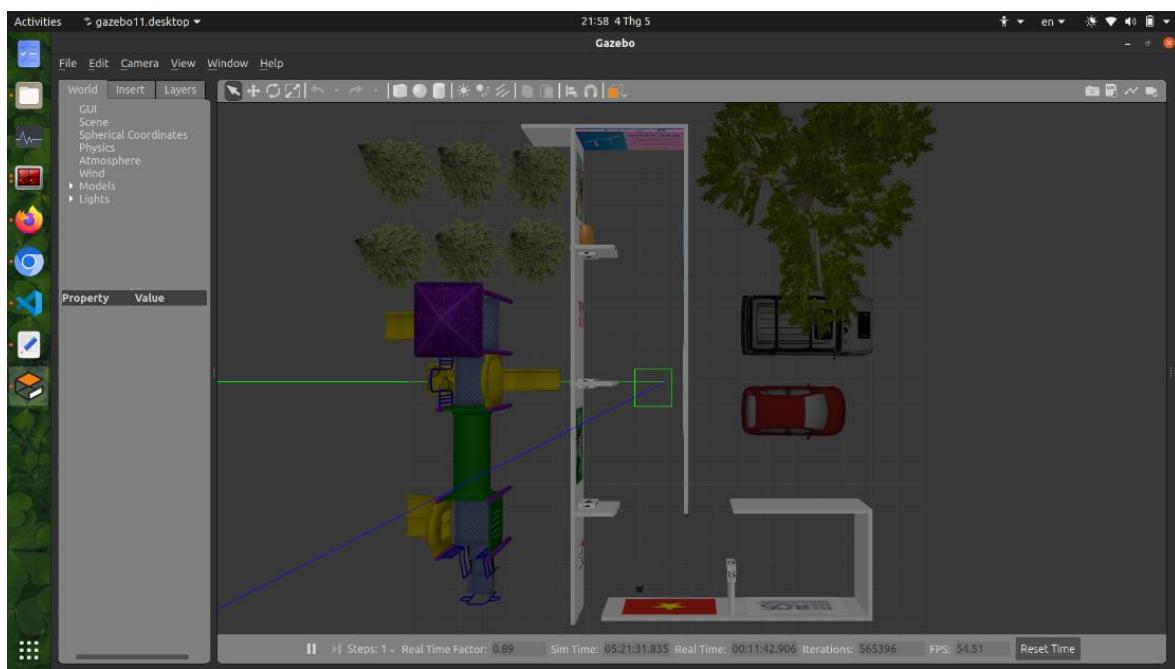
Với các model (tường, các biển báo) đã tạo và các model có sẵn, em sẽ thêm chúng vào cùng 1 môi trường, đặt tên là *empty.world* được lưu ở thư mục *dacn/src/turtlebot3\_simulations/turtlebot3\_gazebo/worlds*. Cách thức thực hiện là thêm vào tệp *empty.world* đoạn lệnh sau để có thể thêm các biển báo vào cùng model tường đã dựng:

```
<include>
  <uri>model://turtlebot3_empty_world/logo_TDTU</uri>
  <pose> 2.84 1.27 0.13 0 0 -0.356</pose>
</include>
```

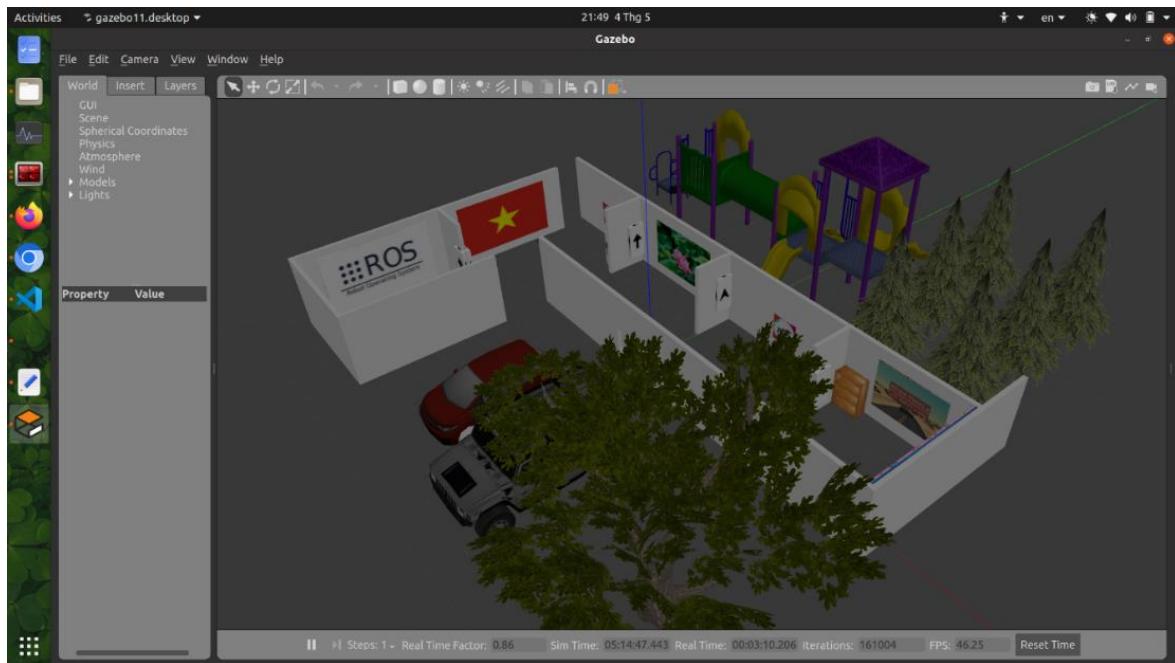
Giao diện của môi trường mà em tự xây dựng có thể quan sát tổng quát qua 2 góc nhìn như ở Hình 4 - 6 và Hình 4 - 7.

# ĐỒ ÁN CHUYÊN NGÀNH

## Trang 31



Hình 4 - 6. Môi trường mô phỏng với góc nhìn từ trên xuống

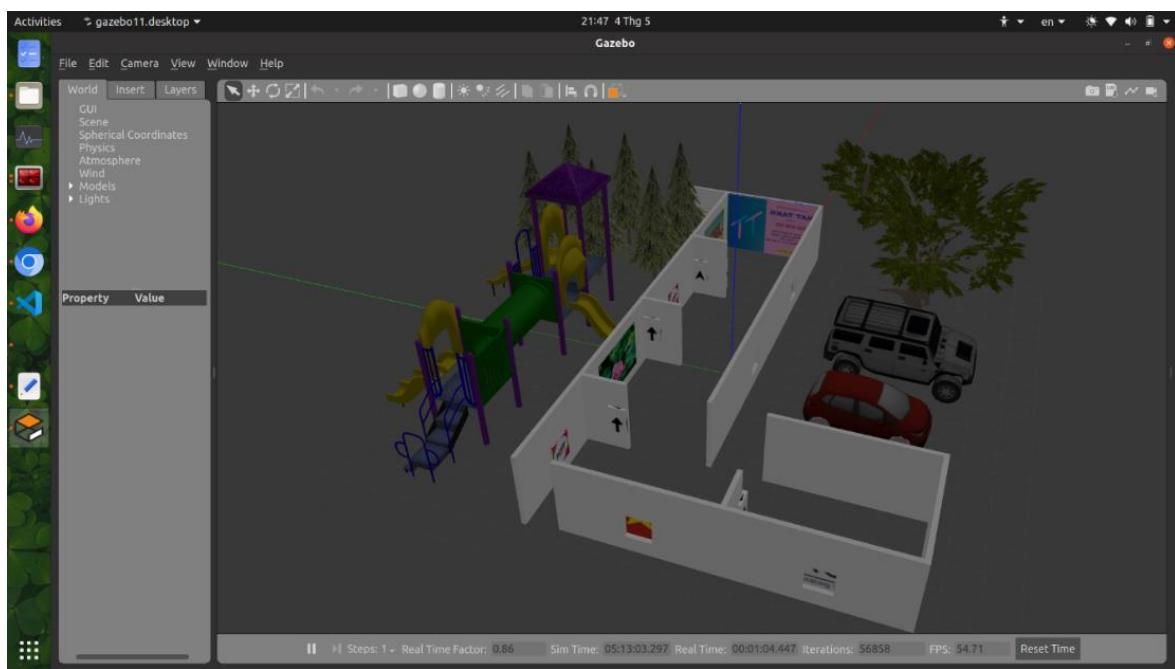


Hình 4 - 7. Môi trường mô phỏng với góc nghiêng nhìn từ điểm cuối đến điểm bắt đầu

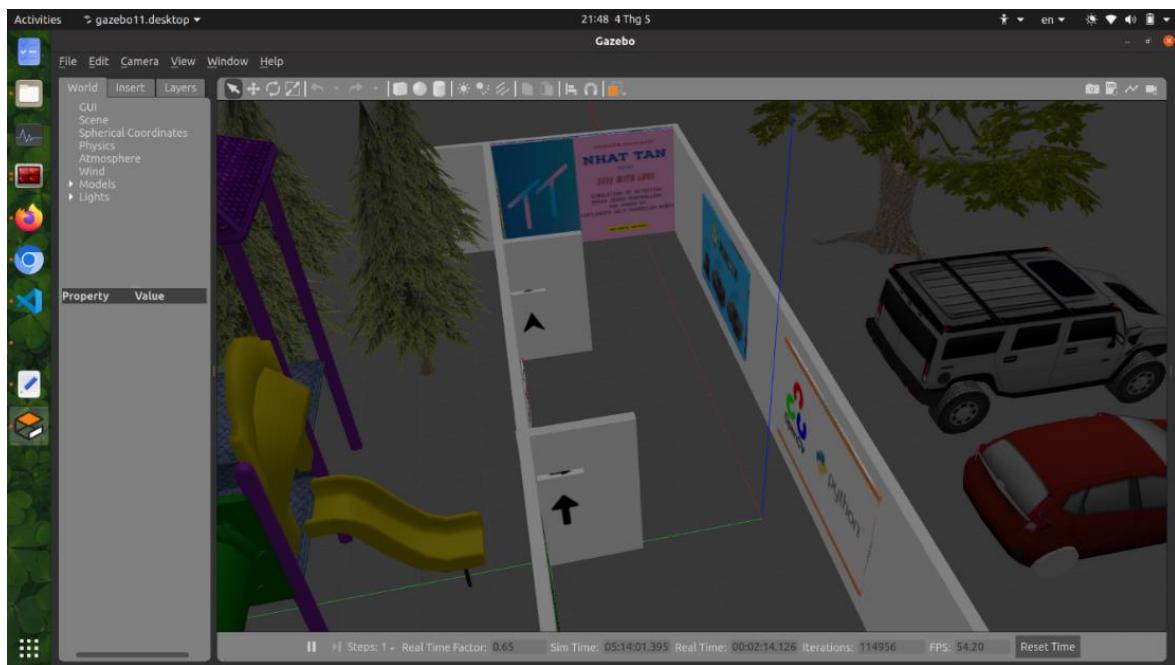
Tiếp tục với Hình 4 - 8, Hình 4 – 9, Hình 4 - 10 và Hình 4 - 11 thể hiện góc nhìn gần với các góc trong môi trường mô phỏng có gắn các biển báo.

# ĐỒ ÁN CHUYÊN NGÀNH

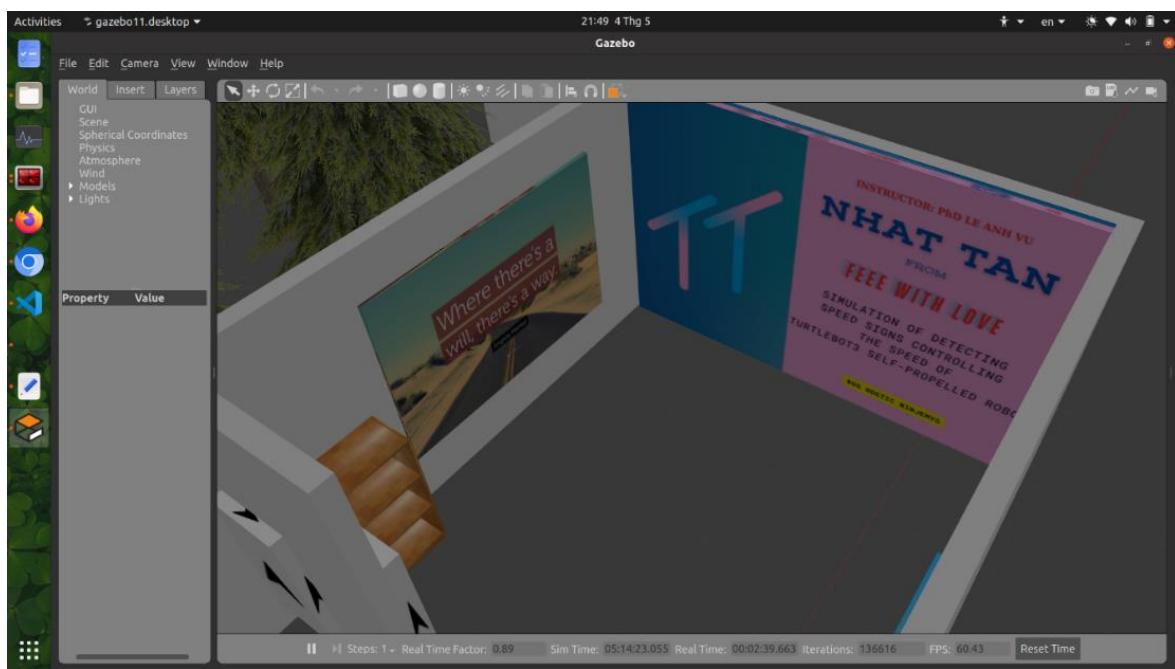
## Trang 32



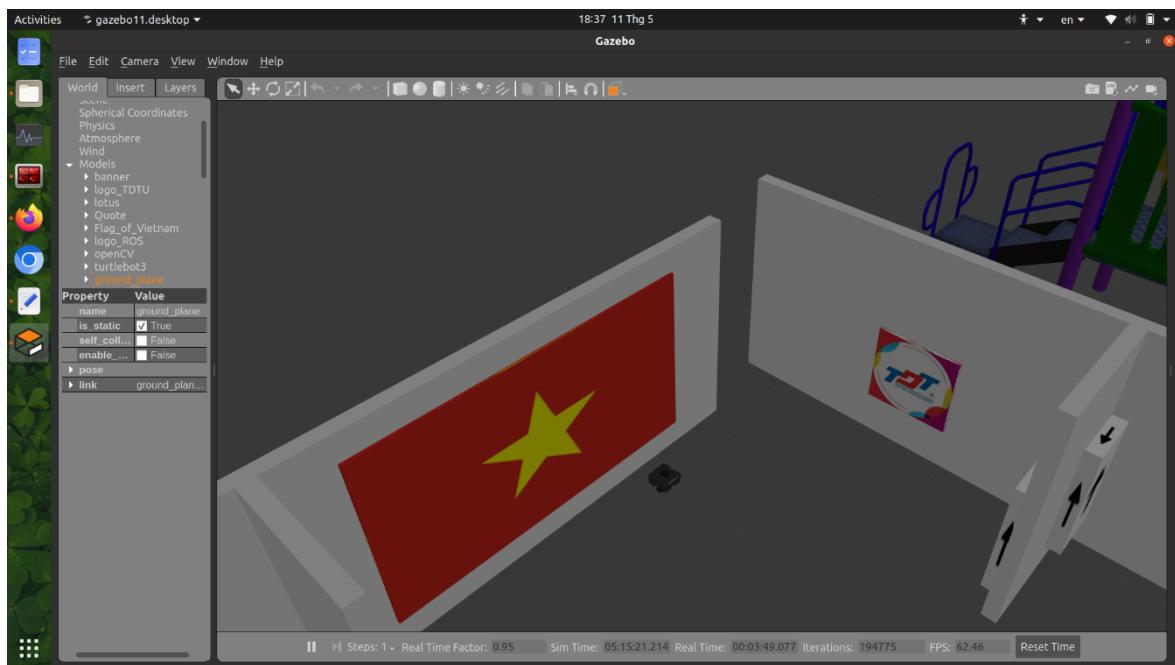
Hình 4 - 8. Môi trường mô phỏng với góc nghiêng nhìn từ điểm bắt đầu đến điểm cuối



Hình 4 - 9. Môi trường mô phỏng với góc nhìn gần các biển báo tốc độ



**Hình 4 - 10. Mô trường mô phỏng với góc nhìn gần các biển báo thông tin**

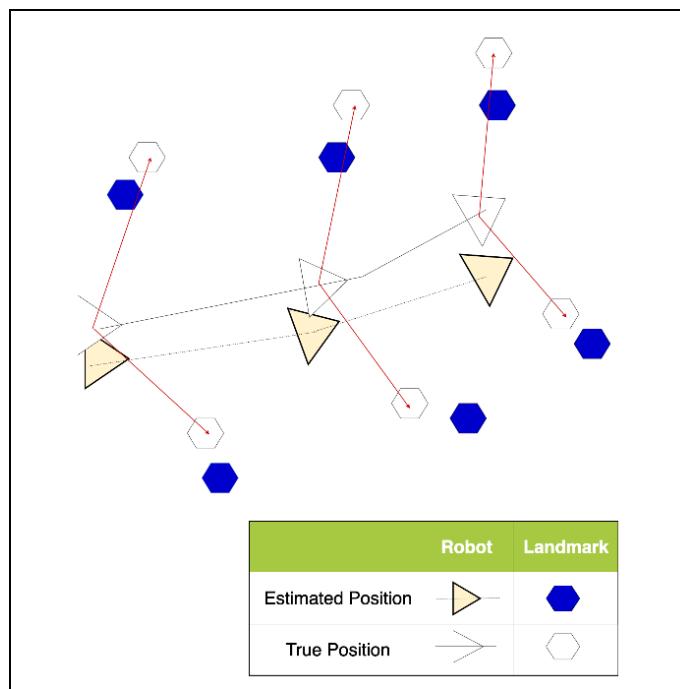


**Hình 4 - 11. Mô trường mô phỏng với góc nhìn gần biển báo hình ảnh lá cờ Việt Nam và logo TDTU**

## CHƯƠNG 5. XÂY DỰNG BẢN ĐỒ ĐỊNH TRƯỚC SLAM

### 5.1 Định nghĩa SLAM

SLAM (Simultaneous Localization and Mapping) mô tả một vấn đề trong robot trong đó robot nên tính toán vị trí của nó trên bản đồ, trong khi nó đang tạo chính xác bản đồ này của một môi trường không xác định bằng cách sử dụng dữ liệu cảm biến. SLAM là một vấn đề quan trọng và ảnh hưởng nhiều bởi không gian lớn, bởi vì nhiều trường hợp mô phỏng cần dựa trên thông tin được thu thập và cấu trúc trong quá trình bản địa hóa và lập bản đồ. Thông thường các địa điểm hành động không cung cấp bản đồ tĩnh của môi trường và không có khả năng định vị vị trí tuyệt đối của robot di động bằng cách sử dụng dữ liệu GPS. Những hạn chế về thời gian và chi phí (cost) càng gây khó khăn cho việc xây dựng bản đồ trước khi đi qua khu vực và điều hướng từ điểm bắt đầu đến vị trí mục tiêu như ví dụ ở Hình 5 - 1.



Hình 5 - 1. Cách tiếp cận lý thuyết của một tác nhân/robot cố gắng tính toán vị trí thực của nó trong khi di chuyển

(Nguồn: UCR-Robotics)

Do đó, hai nhiệm vụ này phải được thực hiện đồng thời với độ chính xác cao. Thường được sử dụng là cảm biến siêu âm hoặc LiDAR (phân biệt với Radar ở Bảng 5 - 1), nhưng cũng có những cách tiếp cận liên quan đến dữ liệu hình ảnh từ máy ảnh, ví dụ như SLAM trực quan. Quá trình chung, như được hình dung, là tăng dần vì không tồn tại bản đồ và robot chỉ có thể phát hiện một phần của môi trường lõi. Nó bắt đầu với vị trí robot là trung tâm tuyệt đối của hệ tọa độ. Từng bước, robot thu thập thông tin về khu vực và hợp nhất các khung thu thập được thành một bản đồ nhất quán toàn cầu. Trong quá trình này, robot di chuyển về phía trước và đo một số bộ phận đã biết và chưa biết. Thông tin này sau đó có thể được sử dụng để tính toán chuyển động và vị trí tuyệt đối mới của robot. Thủ tục này được lặp lại cho đến khi đo diện tích hoàn chỉnh. Mặc dù chuyển động và vị trí chính xác của robot giữa hai phép đo không bao giờ có thể tính toán chính xác, một số thuật toán có thể phát hiện các phần đã được đo của môi trường. Với cái gọi là 'Đóng vòng lặp', chất lượng cao hơn của bản đồ kết quả có thể được cung cấp.

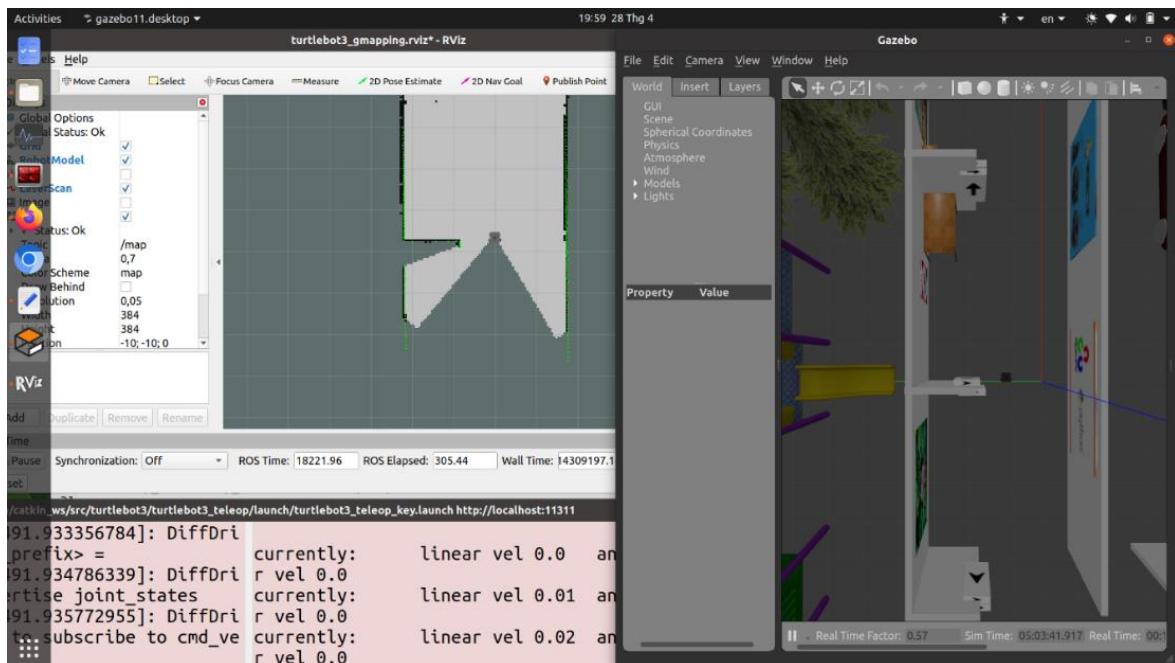
**Bảng 5 - 1. So sánh LiDAR và Radar**

Tính năng	LiDAR	Radar
<b>Loại sóng</b>	Ánh sáng	Vô tuyến
<b>Bước sóng</b>	Ngắn (nhỏ hơn 1mm)	Dài (cm đến m)
<b>Độ phân giải</b>	Cao	Thấp
<b>Độ chính xác</b>	Cao	Thấp
<b>Tầm hoạt động</b>	Ngắn (hàng trăm mét)	Dài (hàng km)
<b>Khả năng xuyên môi trường</b>	Bị ảnh hưởng bởi thời tiết, bụi bẩn	Ít bị ảnh hưởng bởi thời tiết, bụi bẩn
<b>Giá thành</b>	Cao	Thấp
<b>Ứng dụng</b>	Lập bản đồ 3D chi tiết, khảo sát địa hình, xe tự lái, robot	Hệ thống không lưu, giám sát giao thông, dự báo thời tiết

## 5.2 Thực hiện SLAM

Quá trình xây dựng bản đồ định trước SLAM gồm các bước:

- Di chuyển vào không gian làm việc đã tạo và thiết lập môi trường làm việc với lệnh: `source devel/setup.bash`.
- Chạy giả lập bản đồ với Gazebo: `export TURTLEBOT3_MODEL=waffle && roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch`.
- Mở 1 thẻ mới trong Terminal và bật chế độ điều hướng bằng phím: `export TURTLEBOT3_MODEL=waffle && roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch`.
- Chạy lệnh: `roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping` như Hình 5 - 2 để khởi chạy tiến trình tạo bản đồ môi trường xung quanh bằng robot TurtleBot3 và thuật toán Gmapping. Sau đó dùng các phím điều khiển robot đi khắp bản đồ để quét.

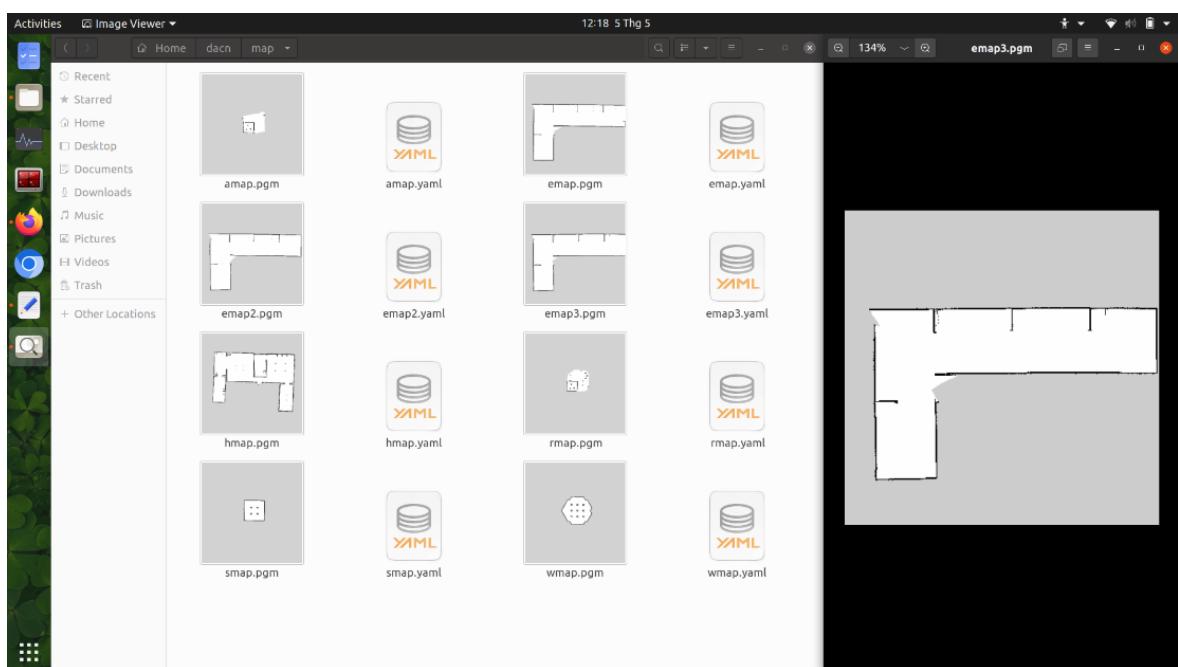


**Hình 5 - 2. Thực hiện SLAM**

- Tạo thư mục lưu bản đồ: `mkdir map`, sau đó vào thư mục rồi lưu bản đồ với lệnh: `rosrun map_server map_saver -f ~/dacn/map/map`. Các thành phần của lệnh này có nghĩa như sau:

- rosrun: Là một lệnh trong ROS dùng để chạy một nút (node) mà không cần biết package chứa nút đó ở đâu.
- map\_server: Là tên của package trong ROS. Package này cung cấp hai nút: map\_server và map\_saver. map\_server dùng để load một bản đồ từ file và xuất bản nó như là một topic ROS, trong khi map\_saver dùng để lưu một bản đồ từ topic ROS vào file.
- map\_saver: Là tên của nút trong package map\_server. Nút này lưu một bản đồ từ topic ROS vào file.
- -f ~/dacn/map/map: Là tham số cho nút map\_saver. -f chỉ ra rằng tham số tiếp theo sẽ là tên của file sẽ được lưu. ~/dacn/map/map là đường dẫn đến file sẽ được lưu. Trong trường hợp này, bản đồ sẽ được lưu vào file map trong thư mục map của workspace dacn trong thư mục home của người dùng.

Khi lưu xong, sẽ thấy có 2 file đó là .pgm (lưu trữ bản đồ dưới dạng hình ảnh bitmap) và .yaml (lưu trữ bản đồ dưới dạng dữ liệu cấu trúc để lưu trữ thông tin chi tiết về bản đồ như Hình 5 - 3, bao gồm: Kích thước bản đồ, Độ phân giải bản đồ, Dữ liệu bản đồ, Thông tin về các điểm mốc trong bản đồ, Thông tin về các vùng cấm).



**Hình 5 - 3. Các bản đồ đã lưu sau khi SLAM**

## CHƯƠNG 6. TỰ ĐỊNH VỊ BẢN THÂN ACML

### 6.1 Định nghĩa ACML

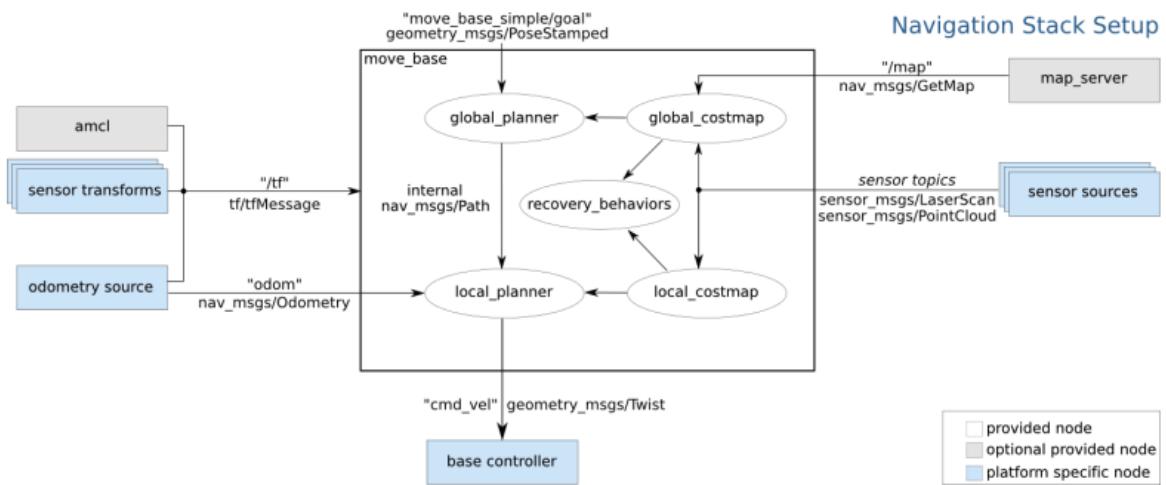
Trong lĩnh vực Robot và Trí tuệ Nhân tạo (AI), thuật ngữ ACML (Adaptable Cognitive-Motivated Meta-Learning) đề cập đến một kỹ thuật học máy meta (meta-learning) có khả năng thích ứng và được thúc đẩy bởi động cơ nhận thức. ACML kết hợp ba khái niệm chính:

- Meta-Learning (Học máy meta): Meta-learning là khả năng học cách học từ nhiều tác vụ liên quan, cho phép AI học nhanh hơn từ ít dữ liệu hơn cho các tác vụ mới. Nó tận dụng tri thức và kinh nghiệm từ các tác vụ đã học trước đó để nhanh chóng điều chỉnh cho tác vụ mới.
- Cognitive (Nhận thức): Thuật ngữ "Cognitive" trong ACML đề cập đến việc sử dụng các nguyên lý nhận thức và lý thuyết học máy dựa trên mô hình não bộ con người. Điều này giúp AI có khả năng suy luận, ra quyết định và thích nghi giống như con người.
- Motivated (Động cơ): "Motivated" nghĩa là hệ thống AI được thúc đẩy bởi các động cơ nội tại, giống như con người được thúc đẩy bởi nhu cầu, mong muốn và mục tiêu. Điều này giúp AI có hành vi mục đích và điều chỉnh chiến lược học tập của mình để đạt được mục tiêu tốt hơn.

Khi kết hợp ba khái niệm này như Hình 6 - 1, ACML cho phép các hệ thống Robot và AI có khả năng:

- Học nhanh từ ít dữ liệu hơn cho các tác vụ mới nhờ meta-learning.
- Suy luận, ra quyết định và thích nghi tốt hơn dựa trên nguyên lý nhận thức.
- Điều chỉnh chiến lược học tập và hành vi để đạt mục tiêu tốt hơn nhờ động cơ nội tại.
- ACML đặc biệt hữu ích trong các ứng dụng Robot và AI cần khả năng thích ứng mạnh mẽ với môi trường luôn thay đổi, học nhanh từ ít dữ liệu và có hành

vi mục đích. Nó giúp Robot và AI trở nên linh hoạt, phản ứng nhanh và hiệu quả hơn khi đối mặt với nhiều tình huống khác nhau.



Hình 6 - 1. Ví dụ về một stack điều hướng ROS

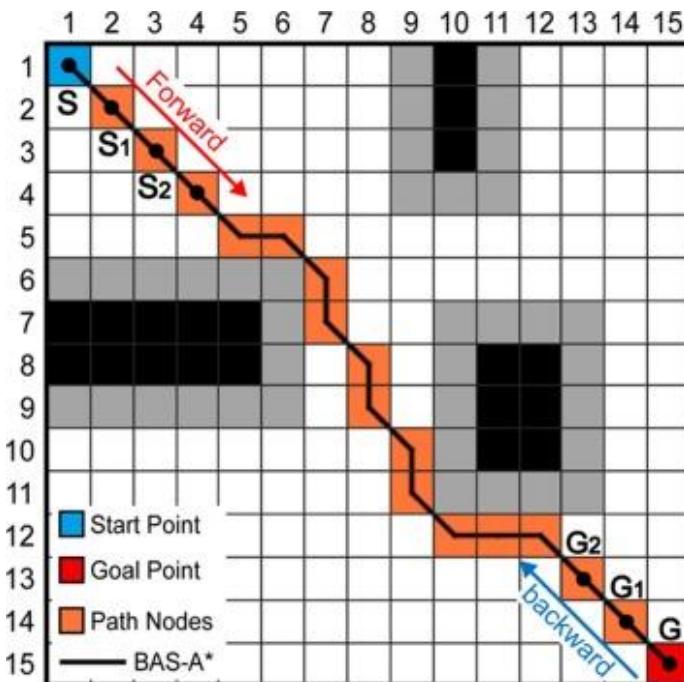
## 6.2 Các thuật toán tìm đường đi của Turtlebot3

### 6.2.1 Phân loại các thuật toán tìm đường đi

Lập kế hoạch đường đi là một phần quan trọng trong lĩnh vực robot và trí tuệ nhân tạo, giúp robot di chuyển từ vị trí hiện tại đến vị trí mục tiêu một cách an toàn và hiệu quả. Có hai loại chính:

- Lập kế hoạch đường đi toàn cục (Global Path Planning): Xác định đường đi tổng thể từ vị trí bắt đầu đến điểm đến, bao gồm việc xem xét các chướng ngại vật và môi trường xung quanh. Một số thuật toán phổ biến cho lập kế hoạch đường đi cục bộ bao gồm:
  - Tìm kiếm theo đồ thị: Mô hình hóa môi trường thành một đồ thị với các nút (điểm) và cạnh (liên kết) giữa các nút. Thuật toán tìm kiếm như DFS (Tìm kiếm theo chiều sâu) hoặc BFS (Tìm kiếm theo chiều rộng) được sử dụng để tìm đường đi từ nút bắt đầu đến nút đích.
  - Phương pháp dựa trên lưới: Chia nhỏ môi trường thành các ô nhỏ và sử dụng thuật toán tìm kiếm để tìm đường đi qua các ô. Các thuật toán phổ biến bao gồm A\* (A-star) và Dijkstra (Hình 6 - 2).

- Phương pháp lập kế hoạch theo không gian cấu hình: Biểu diễn vị trí và hướng của robot trong không gian cấu hình, sau đó sử dụng các thuật toán tối ưu hóa để tìm đường đi ngắn nhất.



**Hình 6 - 2. Ví dụ về thuật toán lập kế hoạch đường đi cục bộ bằng phương pháp dựa trên lưới**

(Nguồn: Computers & Industrial Engineering Volume 168, June 2022, 108123)

- Lập kế hoạch đường đi cục bộ (Local Path Planning): Tìm đường đi ngắn nhất trong khu vực lân cận của robot, giúp robot tránh các chướng ngại vật và điều chỉnh hướng di chuyển khi cần thiết. Một số thuật toán phổ biến cho lập kế hoạch đường đi cục bộ bao gồm:

- Cảm biến: Sử dụng thông tin từ các cảm biến như camera, lidar để phát hiện chướng ngại vật và điều chỉnh hướng di chuyển.
- Phản xạ: Dựa trên phản ứng với môi trường xung quanh, robot di chuyển theo hướng có ít chướng ngại vật nhất.
- Lực hấp dẫn nhân tạo: Sử dụng trường lực ảo để thu hút robot đến mục tiêu và đẩy nó ra khỏi chướng ngại vật.

Mỗi phương pháp thuật toán tìm đường đi đều mang những lợi ích và hạn chế cụ thể, được mô tả chi tiết trong Bảng 6 - 1 dưới đây.

**Bảng 6 - 1. So sánh 2 loại thuật toán**

Thuật toán	Ưu điểm	Nhược điểm
Toàn cục	Cung cấp đường đi tổng thể, tối ưu hóa	Tính toán phức tạp, có thể gặp khó khăn trong môi trường phức tạp
Cục bộ	Dễ triển khai, thích ứng tốt với thay đổi môi trường	Không có tầm nhìn xa, có thể bị kẹt trong tối ưu cục bộ

### 6.2.2 Thuật toán tìm kiếm A\* (A Star)

Thuật toán A\* là một thuật toán tìm đường đi toàn cục được sử dụng rộng rãi trong trí tuệ nhân tạo và robot học. Nó là một thuật toán tìm kiếm theo chiều rộng lặp lại (informed search algorithm) sử dụng hàm heuristic để ước tính khoảng cách đến đích.

Nguyên lý hoạt động: A\* hoạt động bằng cách duy trì một danh sách các nút "mở" và "đóng". Danh sách mở bao gồm các nút chưa được khám phá, trong khi danh sách đóng bao gồm các nút đã được khám phá. Thuật toán bắt đầu bằng cách thêm nút bắt đầu vào danh sách mở. Tại mỗi bước, A\* chọn nút có giá trị  $f(n)$  nhỏ nhất từ danh sách mở, trong đó  $f(n)$  được tính bằng:  $f(n) = g(n) + h(n)$ , với:

- $g(n)$ : Chi phí đường đi từ nút bắt đầu đến nút  $n$ .
- $h(n)$ : Hàm heuristic ước tính chi phí đường đi từ nút  $n$  đến đích.

Thuật toán sau đó khám phá các nút liền kề của nút đã chọn và thêm chúng vào danh sách mở nếu chúng chưa được khám phá. Nếu một nút liền kề đã có trong danh sách mở, A\* sẽ cập nhật chi phí đường đi nếu đường đi mới có chi phí nhỏ hơn. Thuật toán tiếp tục cho đến khi nút đích được tìm thấy hoặc danh sách mở trống. Nếu nút đích được tìm thấy, A\* sẽ trả về đường đi từ nút bắt đầu đến nút đích.

### 6.2.3 Thuật toán Dijkstra

Thuật toán Dijkstra cũng là một thuật toán tìm đường đi toàn cục cổ điển được sử dụng để giải quyết bài toán tìm đường đi ngắn nhất từ một điểm cho trước tới tất cả các điểm còn lại trong đồ thị có trọng số. Thuật toán này được đặt theo tên của nhà toán học người Hà Lan Edsger W. Dijkstra. Ý tưởng cơ bản của thuật toán Dijkstra như sau:

- Bước 1: Từ đỉnh gốc, khởi tạo khoảng cách tới chính nó là 0, khởi tạo khoảng cách nhỏ nhất ban đầu tới các đỉnh khác là  $+\infty$ . Ta được danh sách các khoảng cách tới các đỉnh.
- Bước 2: Chọn đỉnh  $a$  có khoảng cách nhỏ nhất trong danh sách này và ghi nhận. Các lần sau sẽ không xét tới đỉnh này nữa.
- Bước 3: Lần lượt xét các đỉnh kề  $b$  của đỉnh  $a$ . Nếu khoảng cách từ đỉnh gốc tới đỉnh  $b$  nhỏ hơn khoảng cách hiện tại đang được ghi nhận thì cập nhật giá trị và đỉnh kề  $a$  vào khoảng cách hiện tại của  $b$ .
- Bước 4: Sau khi xét tất cả đỉnh kề  $b$  của đỉnh  $a$ . Lúc này ta được danh sách khoảng cách tới các điểm đã được cập nhật. Quay lại Bước 2 với danh sách này.
- Thuật toán kết thúc khi chọn được khoảng cách nhỏ nhất từ tất cả các điểm.

Thuật toán Dijkstra có thể giải quyết bài toán tìm đường đi ngắn nhất trên đồ thị vô hướng lẫn có hướng miễn là trọng số không âm.

### 6.2.4 Thuật toán DWA

Thuật toán DWA (Dynamic Window Approach) là một thuật toán tìm đường đi cục bộ được sử dụng rộng rãi trong robot di động. Nó phù hợp với các môi trường động, nơi các chướng ngại vật có thể xuất hiện hoặc di chuyển bất ngờ.

Nguyên lý hoạt động: DWA hoạt động bằng cách tạo ra một "cửa sổ động" xung quanh robot, đại diện cho các trạng thái có thể của robot tại thời điểm hiện tại. Thuật toán sau đó đánh giá các trạng thái có thể này dựa trên các tiêu chí sau:

- Tính khả thi: Trạng thái có thể di chuyển được trong môi trường mà không va chạm với các chướng ngại vật.
- Tính tối ưu: Trạng thái dẫn đến đường đi ngắn hơn đến đích.
- Tính an toàn: Trạng thái không dẫn đến va chạm với các chướng ngại vật động.
- Thuật toán chọn trạng thái có thể tối ưu hóa sự cân bằng giữa các tiêu chí này.

Các bước thực hiện:

- Tạo cửa sổ động: Tạo một cửa sổ xung quanh robot, bao gồm các trạng thái có thể của robot tại thời điểm hiện tại.
- Đánh giá tính khả thi: Loại bỏ các trạng thái không thể di chuyển được trong môi trường.
- Đánh giá tính tối ưu: Sử dụng hàm chi phí để đánh giá độ dài của đường đi dẫn đến mỗi trạng thái.
- Đánh giá tính an toàn: Sử dụng các cảm biến để dự đoán chuyển động của các chướng ngại vật động và loại bỏ các trạng thái có thể dẫn đến va chạm.
- Chọn trạng thái tối ưu: Chọn trạng thái có thể tối ưu hóa sự cân bằng giữa tính khả thi, tính tối ưu và tính an toàn.
- Thực hiện hành động: Thực hiện hành động tương ứng với trạng thái đã chọn.

#### 6.2.5 So sánh các thuật toán

Bảng 6 - 2 sau đây cung cấp so sánh về độ phức tạp giữa ba thuật toán: DWA, A\* và Dijkstra, giúp hiểu rõ hơn về các đặc điểm và khả năng của từng thuật toán.

Bảng 6 - 2. So sánh về độ phức tạp của 3 thuật toán

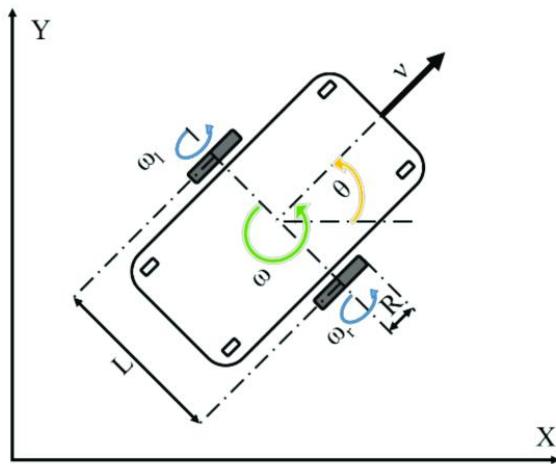
Thuật toán	Độ phức tạp
DWA	Độ phức tạp của DWA không được định rõ ràng như A* và Dijkstra vì nó phụ thuộc vào nhiều yếu tố như số lượng mục tiêu, số lượng chướng ngại vật, và tốc độ của robot.

Thuật toán	Độ phức tạp
A*	Độ phức tạp của A* trong trường hợp tốt nhất là $O(V \log V)$ hoặc $O(V \log E)$ tùy theo cách cài đặt, và trong trường hợp xấu nhất là $O(E \log V)$ .
Dijkstra	Độ phức tạp của thuật toán Dijkstra là $O(V^2)$ . Nếu sử dụng một hàng đợi ưu tiên (priority queue), ví dụ như Binary heap, và sử dụng danh sách kề thì độ phức tạp của thuật toán sẽ bị giảm xuống còn $O[(V+E)\log V]$ .

### 6.3 Mô hình động lực học truyền động vi sai 2 bánh của TurtleBot3

Mô hình động lực học truyền động vi sai 2 bánh của TurtleBot3 liên quan đến cách mà robot di chuyển dựa trên sự điều khiển độc lập của hai bánh xe (Hình 6 - 3). Trong mô hình này, hai bánh xe được đặt song song với nhau và cả 2 đều có thể quay độc lập, cho phép robot thực hiện các chuyển động phức tạp như quay 360 độ tại chỗ.

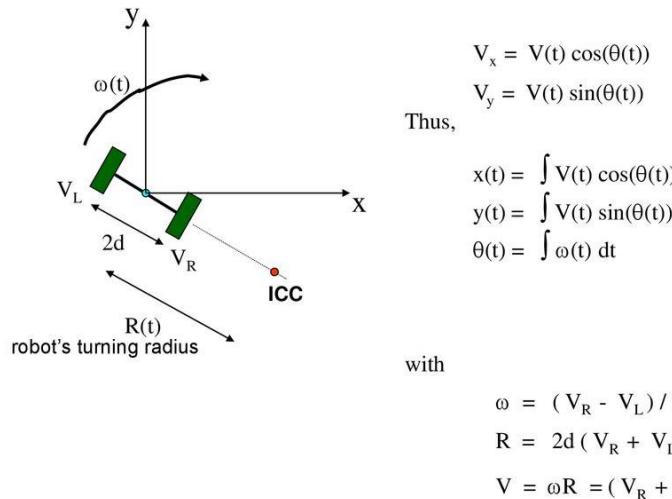
Cụ thể, khi cả hai bánh xe quay cùng một hướng với cùng một tốc độ, robot sẽ di chuyển thẳng. Khi một bánh xe quay nhanh hơn bánh kia, robot sẽ rẽ. Khi hai bánh xe quay ngược hướng nhau, robot sẽ quay tại chỗ.



**Hình 6 - 3. Mô hình truyền động vi sai của TurtleBot3**

(Nguồn: Hydrogen Research Institute)

Để điều khiển mô hình động học truyền động vi sai, cần sử dụng hai giá trị là vận tốc tuyến tính và vận tốc góc (Hình 6 - 4). Vận tốc tuyến tính là tốc độ di chuyển thẳng của robot, trong khi vận tốc góc là tốc độ quay của robot.

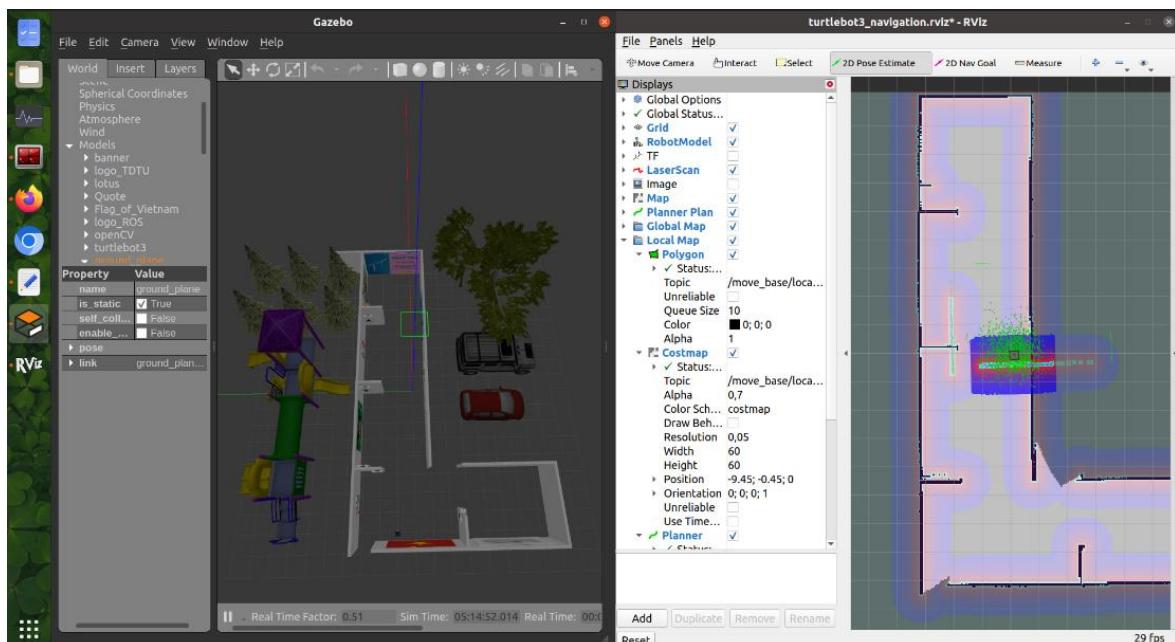


**Hình 6 - 4. Công thức truyền động vi sai của TurtleBot3**

(Nguồn: Medium)

#### 6.4 Thực hiện ACML

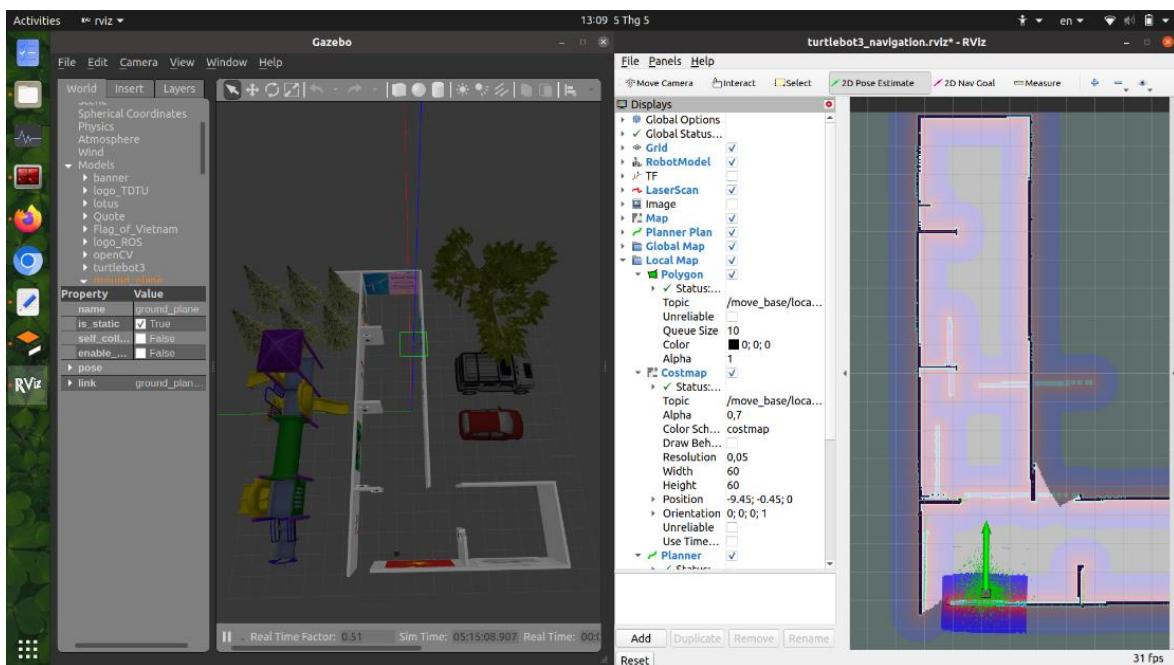
Sau khi thực hiện xây dựng bản đồ định trước SLAM, để điều khiển robot (ở đây là TurtleBot3) cần thực hiện tự định vị bản thân trong bản đồ ACML.



**Hình 6 - 5. Hình ảnh lần đầu khi khởi chạy ACML**

Cách thức thực hiện gồm các bước sau:

- Di chuyển vào không gian làm việc đã tạo và thiết lập môi trường làm việc với lệnh: `source devel/setup.bash`.
- Chạy giả lập bản đồ với Gazebo: `export TURTLEBOT3_MODEL=waffle && roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch`.
- Chạy mô phỏng điều hướng với: `export TURTLEBOT3_MODEL=waffle && roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml`.
- Khi khởi chạy, vị trí của Turtlebot3 trong Gazebo và RVIZ sẽ khác nhau như Hình 6 - 5. Cần chọn vào nút 2D Pose Estimate trong RVIZ (Hình 6 - 6) và đánh dấu lại vị trí robot giống với robot bên Gazebo.

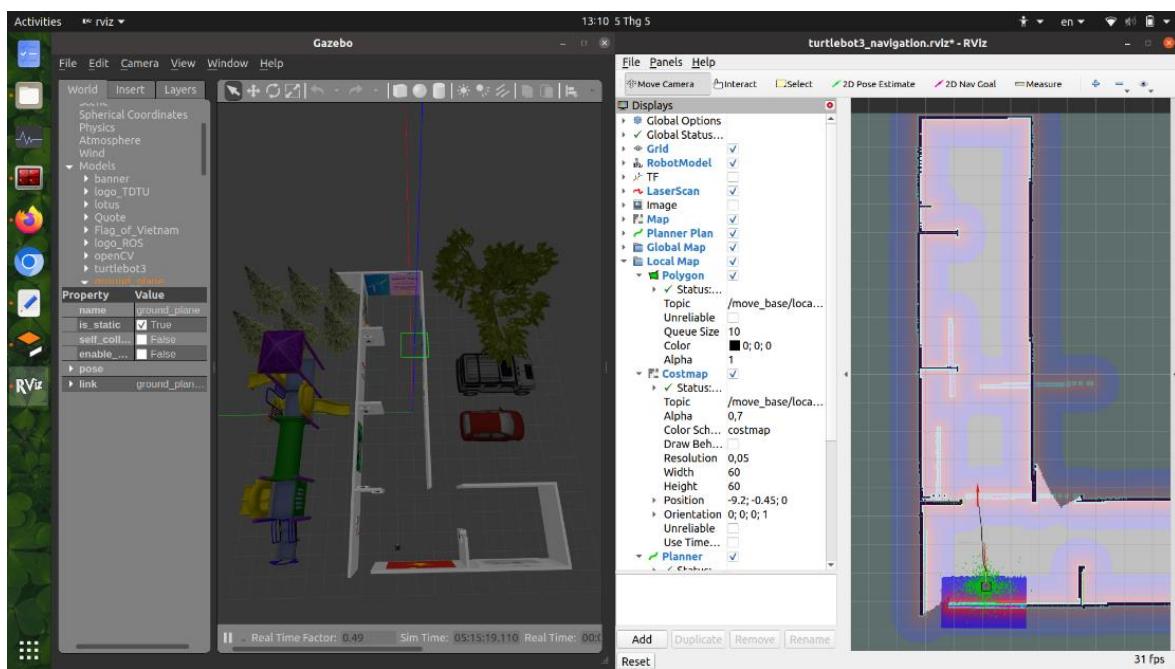


**Hình 6 - 6. Đánh dấu lại vị trí robot**

- Khi robot đã đúng vị trí, chọn nút 2D Nav Goal để đánh dấu đích để robot di chuyển đến như Hình 6 - 7.

# ĐỒ ÁN CHUYÊN NGÀNH

## Trang 47



Hình 6 - 7. Chọn đích đến cho robot tự di chuyển

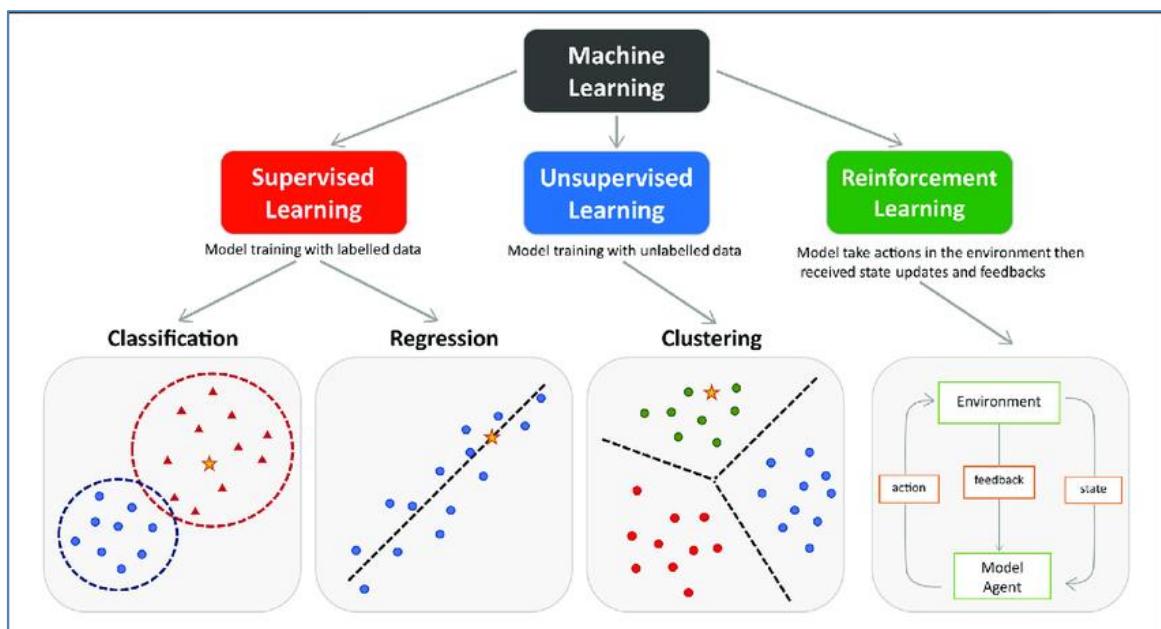
## CHƯƠNG 7. MÔ HÌNH PHÁT HIỆN BIỂN BÁO TỐC ĐỘ

### 7.1 Định nghĩa máy học, thị giác máy tính và mô hình YOLOv8

#### 7.1.1 Máy học

Máy học, hay còn gọi là học máy (machine learning), giúp cho robot có khả năng tự học từ dữ liệu và kinh nghiệm, tự điều chỉnh hành vi dựa trên những thông tin mới. Điều này làm cho robot không chỉ thực hiện các nhiệm vụ cơ học mà còn có thể nhận thức, phân tích thông tin và ra quyết định một cách linh hoạt. Từ đó có thể thấy máy học đóng vai trò vô cùng quan trọng, vì robot muốn di chuyển phải có khả năng cảm nhận, quan sát môi trường giống con người.

Nói chính xác, máy học là một lĩnh vực của trí tuệ nhân tạo tập trung vào việc phát triển các phương pháp và kỹ thuật để máy tính có thể học từ dữ liệu mà không cần phải được lập trình cụ thể. Thay vì việc chỉ dạy máy tính thực hiện các nhiệm vụ cụ thể, máy học cho phép máy tính học từ dữ liệu để tự động cải thiện hiệu suất của mình theo thời gian.



Hình 7 - 1. Phân loại máy học

(Nguồn: University College London)

Các phương pháp máy học được thể hiện như Hình 7 - 1 bao gồm học có giám sát, học không giám sát, và học tăng cường, cùng với nhiều kỹ thuật và thuật toán cụ thể trong mỗi loại học. Máy học có thể chia thành các loại sau:

- Học có giám sát (Supervised Learning): Là loại học tập mà mô hình được cung cấp một tập dữ liệu được đánh nhãn sẵn, trong đó mỗi điểm dữ liệu được gắn nhãn với kết quả mong muốn. Các loại học có giám sát phổ biến bao gồm:
  - Hồi quy (Regression): Dự đoán các giá trị số liên tục.
  - Phân loại (Classification): Mục tiêu là dự đoán lớp hoặc nhãn của các mẫu dữ liệu dựa trên các đặc trưng (features) của chúng. Các lớp có thể là nhãn, danh mục hoặc các danh mục khác nhau. Đây là loại học máy ứng dụng trong đồ án này.
- Học không giám sát (Unsupervised Learning): Là loại học tập mà mô hình được cung cấp một tập dữ liệu không được đánh nhãn. Máy học cách xác định các mẫu và cấu trúc từ dữ liệu. Các phương pháp chính của học không giám sát bao gồm:
  - Clustering (Phân cụm): Trong phân cụm, mục tiêu là chia dữ liệu thành các nhóm (clusters) sao cho các điểm dữ liệu trong cùng một nhóm có tính chất tương tự nhau và các điểm ở các nhóm khác biệt. Các thuật toán phân cụm phổ biến bao gồm K-means clustering và hierarchical clustering.
  - Dimensionality Reduction (Giảm chiều dữ liệu): Trong giảm chiều dữ liệu, mục tiêu là giảm số lượng biến đầu vào trong dữ liệu mà vẫn giữ lại những thông tin quan trọng nhất. Điều này giúp giảm chiều của không gian dữ liệu, giảm thiểu độ phức tạp tính toán và giúp hiểu được cấu trúc ẩn trong dữ liệu. Các phương pháp giảm chiều dữ liệu bao gồm Principal Component Analysis (PCA) và t-SNE (t-distributed Stochastic Neighbor Embedding).
  - Density Estimation (Ước lượng mật độ): Trong ước lượng mật độ, mục tiêu là xác định hàm mật độ xác suất của dữ liệu. Điều này có thể giúp

xác định phân phối của dữ liệu và phát hiện ra các đặc điểm quan trọng của nó. Các phương pháp ước lượng mật độ bao gồm Gaussian Mixture Models (GMM) và Kernel Density Estimation (KDE).

- Học tăng cường (Reinforcement Learning): Là một phương pháp trong máy học mà một agent (đại diện) học cách đạt được một mục tiêu trong một môi trường dựa trên việc tương tác với môi trường đó. Trong học tăng cường, agent thực hiện các hành động, và sau đó nhận phản hồi từ môi trường dưới dạng phần thưởng hoặc phạt. Mục tiêu của agent là tìm ra một chiến lược (policy) tối ưu để chọn hành động sao cho tổng phần thưởng (hoặc giá trị) được tối đa hóa trong thời gian dài.

### 7.1.2 Thị giác máy tính

Thị giác máy tính (Computer Vision) là một phân nhánh của trí tuệ nhân tạo tập trung vào việc phát triển các phương pháp và công nghệ để máy tính có thể hiểu và phân tích hình ảnh và video một cách tự động. Mục tiêu của thị giác máy tính là làm cho máy tính (và robot) có khả năng "nhìn" và "hiểu" thế giới như con người, bao gồm việc nhận diện đối tượng, phân tích hành vi, và hiểu bối cảnh từ hình ảnh và video. Các ứng dụng của thị giác máy tính bao gồm nhận diện khuôn mặt, phát hiện đối tượng, xe tự lái, y tế hình ảnh, và nhiều lĩnh vực khác. Một số loại tác vụ của thị giác máy tính:

- Phân loại ảnh (Image Classification): Phân loại ảnh theo một trong số các nhóm được xác định trước.
- Xác định vị trí vật thể (Object Localization): Xác định vị trí của một vật thể trong ảnh.
- Phát hiện vật thể (Object Detection): Nhận dạng các vật thể cụ thể trong ảnh.
- Phân đoạn ảnh (Image Segmentation): Chia một ảnh thành nhiều phân đoạn hoặc "siêu điểm ảnh".
- Tái tạo cảnh (Scene Reconstruction): Tạo mô hình 3D của một cảnh từ một loạt ảnh.

- Phát hiện sự kiện (Event Detection): Nhận dạng các sự kiện cụ thể diễn ra trong video.
- Nhận dạng hoạt động (Activity Recognition): Nhận dạng các hành động hoặc hoạt động của người hoặc vật thể trong video.
- Theo dõi video (Video Tracking): Theo dõi chuyển động của các vật thể trong video.
- Ước tính tư thế 3D (3D Pose Estimation): Ước tính tư thế của một vật thể 3D từ ảnh.
- Học hỏi (Learning): Huấn luyện mô hình để cải thiện hiệu suất của nó trên một tác vụ.
- Lập chỉ mục (Indexing): Sắp xếp hình ảnh trong cơ sở dữ liệu để tạo điều kiện truy xuất hiệu quả.
- Ước tính chuyển động (Motion Estimation): Ước tính chuyển động của các vật thể giữa các khung hình trong video.
- Điều khiển thị giác (Visual Servoing): Điều khiển robot dựa trên phản hồi hình ảnh.
- Mô hình cảnh 3D (3D Scene Modeling): Tạo mô hình 3D của một cảnh từ một loạt ảnh.
- Phục hồi ảnh (Image Restoration): Cải thiện chất lượng của ảnh bị suy giảm.

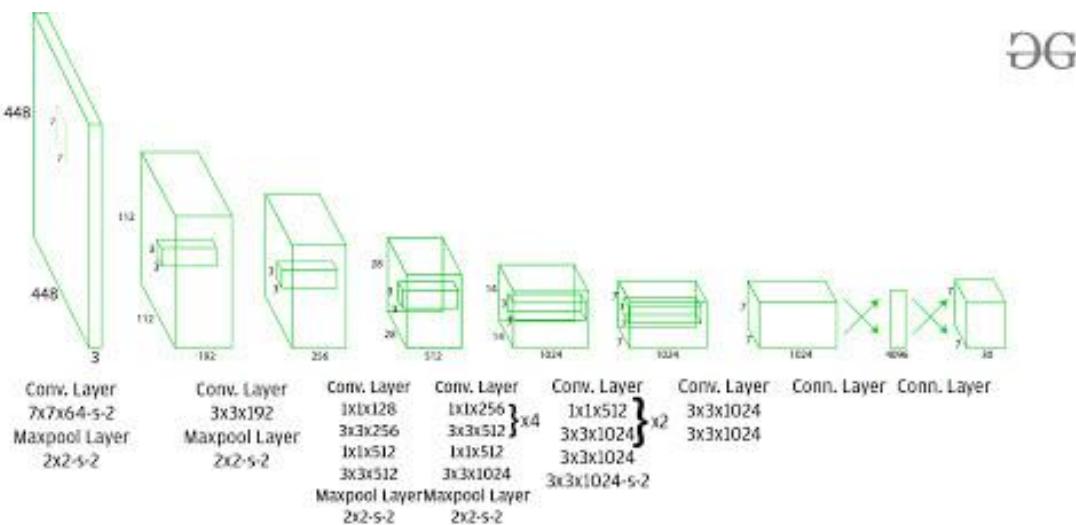
Mỗi loại thị giác máy tính đều có các ứng dụng riêng và được sử dụng để giải quyết các vấn đề khác nhau trong các lĩnh vực khác nhau như xe tự hành, hình ảnh y tế, giám sát và hơn thế nữa. Ở đồ án này, em đã thực hiện yêu cầu đề tài với phương pháp phân loại ảnh (Image Classification).

### 7.1.3 Mô hình YOLOv8

YOLO (You Only Look Once) là một thuật toán trong lĩnh vực học máy, cụ thể là học sâu (Deep Learning), được sử dụng cho nhiệm vụ phát hiện đối tượng. YOLO được giới thiệu lần đầu tiên vào năm 2015 bởi Joseph Redmon, Santosh Divvala, Ross Girshick, và Ali Farhadi.

YOLO sử dụng một mạng nơ-ron tích chập (CNN) được mô tả như Hình 7 - 2 để xử lý hình ảnh, chia hình ảnh thành lưới, thực hiện nhiều dự đoán cho mỗi ô lưới, lọc ra các dự đoán có độ tin cậy thấp, và sau đó loại bỏ các hộp ch่อง chéo để tạo ra đầu ra cuối cùng.

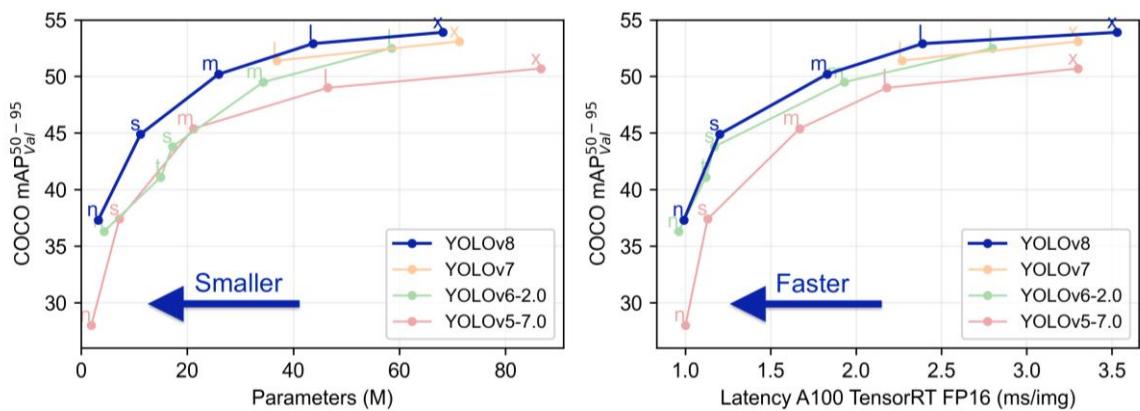
YOLO đã trở thành một hệ thống phát hiện đối tượng thời gian thực trung tâm cho các ứng dụng như robot, xe tự lái, và giám sát video. YOLO nổi tiếng vì tốc độ và độ chính xác của nó.



**Hình 7 - 2. Cấu trúc mô hình YOLO**

(Nguồn: GeeksforGeeks)

Mô hình YOLOv8 được xây dựng dựa trên các phiên bản YOLO trước đó bằng cách giới thiệu các cải tiến khác nhau về độ chính xác, tốc độ và độ tin cậy như Hình 7 - 3 đã so sánh. Một số tính năng và cải tiến chính của YOLOv8 có thể bao gồm:



**Hình 7 - 3. So sánh YOLOv8 với các phiên bản khác**

(Nguồn: Ultralytics)

- Cải tiến về kiến trúc: YOLOv8 có thể có kiến trúc mạng hiệu quả và được tối ưu hóa cho phép thời gian suy luận nhanh hơn mà không ảnh hưởng đến độ chính xác.
- Mạng Backbone: YOLOv8 có thể sử dụng kiến trúc mạng backbone khác nhau, chẳng hạn như Darknet hoặc ResNet, để trích xuất các đặc điểm từ hình ảnh đầu vào hiệu quả hơn.
- Dự đoán đa tỷ lệ: YOLOv8 có thể sử dụng các kỹ thuật dự đoán đa tỷ lệ để phát hiện các vật thể ở các tỷ lệ khác nhau trong một hình ảnh, cải thiện hiệu suất phát hiện đối với các vật thể có kích thước khác nhau.
- Các chiến lược đào tạo nâng cao: YOLOv8 có thể kết hợp các chiến lược đào tạo tiên tiến, chẳng hạn như tăng cường dữ liệu, học theo chương trình và học chuyên giao, để cải thiện khả năng tổng quát hóa và độ tin cậy của mô hình.
- Các kỹ thuật xử lý hậu: YOLOv8 có thể sử dụng các kỹ thuật xử lý hậu được cải tiến, chẳng hạn như triệt tiêu không cực đại và tinh chỉnh hộp bao quanh, để tạo ra các phát hiện vật thể chính xác và chi tiết hơn.

Nhìn chung, các phiên bản YOLOv8 nhắm đến việc cung cấp hiệu suất phát hiện vật thể tiên tiến trong khi vẫn duy trì tốc độ suy luận theo thời gian thực (Bảng 7 - 1), khiến nó phù hợp cho nhiều ứng dụng khác nhau, bao gồm giám sát, lái xe tự động và robot.

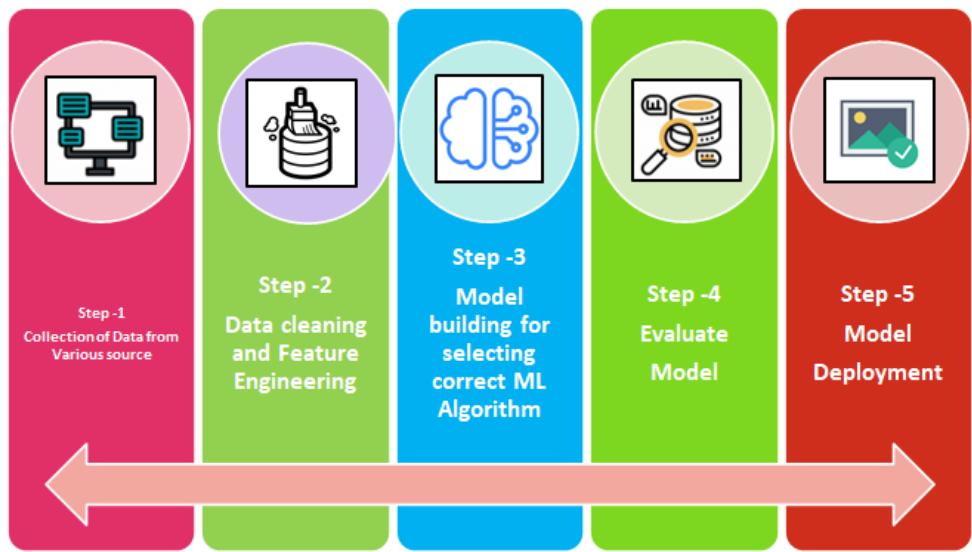
**Bảng 7 - 1. So sánh các phiên bản của YOLOv8**

Mô hình	Kích thước (pixels)	mAP val 50-	Tốc độ CPU	Tốc độ A100	Params (M)	FLOPs (B)
		95	ONNX (ms)	TensorRT (ms)		
<b>YOLOv8n</b>	640	37.3	80.4	0.99	3.2	8.7
<b>YOLOv8s</b>	640	44.9	128.4	1.20	11.2	28.6
<b>YOLOv8m</b>	640	50.2	234.7	1.83	25.9	78.9
<b>YOLOv8l</b>	640	52.9	575.2	2.39	43.7	165.2
<b>YOLOv8x</b>	640	53.9	479.1	3.53	68.2	257.8

## 7.2 Quá trình học máy

Quá trình học máy thường bao gồm các bước sau:

- Thu thập dữ liệu (Data Collection): Máy học từ dữ liệu cung cấp. Việc thu thập dữ liệu đáng tin cậy rất quan trọng để mô hình học máy có thể tìm ra các mẫu chính xác.
- Chuẩn bị dữ liệu (Data Preparation): Tại đây, chúng ta sẽ tải dữ liệu vào một nơi phù hợp và chuẩn bị nó để sử dụng trong quá trình huấn luyện học máy.
- Chọn mô hình học máy phù hợp (Selecting the Right Machine Learning Model): Có nhiều mô hình học máy khác nhau, mỗi mô hình phù hợp với một loại vấn đề cụ thể.
- Huấn luyện mô hình (Training the Model): Trong giai đoạn này, mô hình sẽ học từ dữ liệu huấn luyện để tìm ra mối quan hệ giữa đầu vào và đầu ra.
- Đánh giá hiệu suất mô hình (Evaluating Model Performance): Đánh giá mô hình trên dữ liệu kiểm tra (không được sử dụng trong quá trình huấn luyện) để xem mô hình dự đoán như thế nào với dữ liệu mới.
- Tinh chỉnh và tối ưu hóa mô hình (Tuning and Optimizing the Model): Dựa trên kết quả đánh giá, chúng ta có thể điều chỉnh các tham số của mô hình để cải thiện hiệu suất.
- Dự đoán với dữ liệu mới (Prediction): Sau khi mô hình đã được huấn luyện và tinh chỉnh, chúng ta có thể sử dụng nó để dự đoán kết quả cho dữ liệu mới.
- Triển khai mô hình (Model Deployment): Là quá trình triển khai một mô hình học máy đã được huấn luyện vào một môi trường sản xuất, nơi mà nó có thể nhận đầu vào và trả về đầu ra. Đây là một bước quan trọng trong quá trình phát triển mô hình học máy, vì nó cho phép mô hình được sử dụng trong thực tế để đưa ra dự đoán hoặc thực hiện các tác vụ cụ thể.



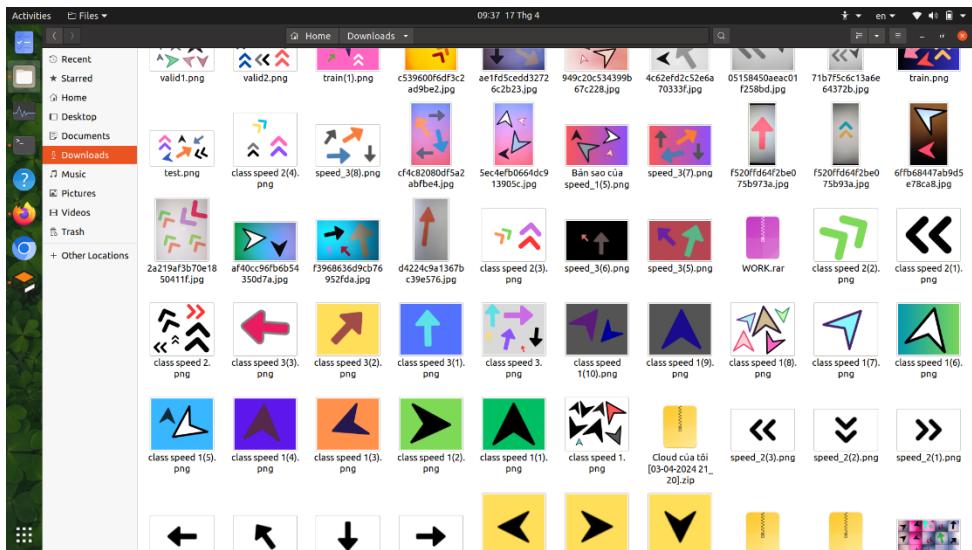
**Hình 7 - 4. Các bước trong quá trình học máy**

(Nguồn: Labellerr)

Mỗi bước trong quá trình học máy (như Hình 7 - 4) đều quan trọng và cần được thực hiện cẩn thận để đảm bảo hiệu suất tốt nhất của mô hình học máy.

### 7.3 Đào tạo mô hình phát hiện biển báo

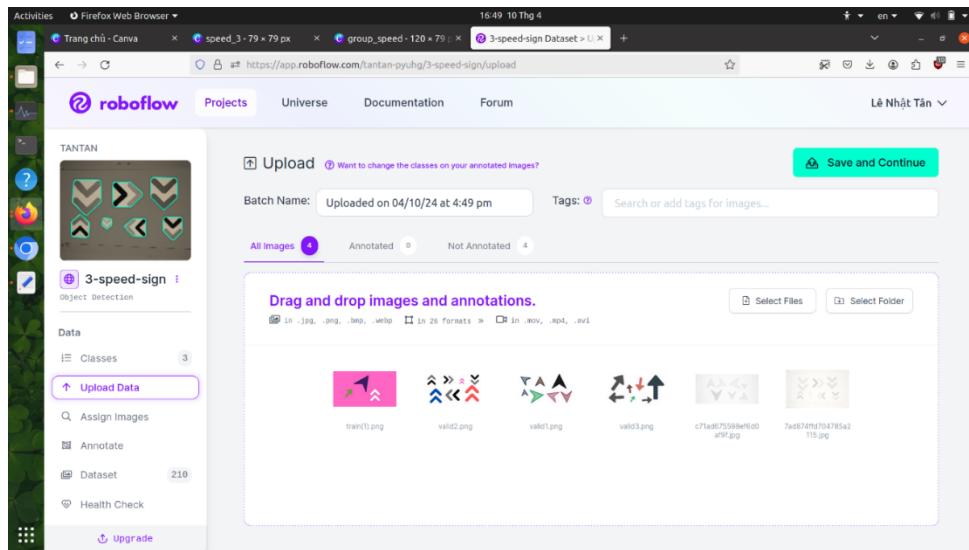
#### 7.3.1 Chuẩn bị hình ảnh



**Hình 7 - 5. Tạo số lượng hình ảnh ban đầu**

Để tăng số lượng ảnh ban đầu, em đã thay đổi cường độ màu sắc, chụp với các góc nghiêng khác nhau, cắt hình ảnh, lật ảnh, xoay ảnh, tăng/giảm độ sáng/độ tương

phản... của 3 biển báo đã thiết kế như Hình 7 - 5. Sau đó tải tất cả lên Roboflow như Hình 7 - 6.



**Hình 7 - 6. Tải hình ảnh lên Roboflow**

### 7.3.2 Dán nhãn hình ảnh

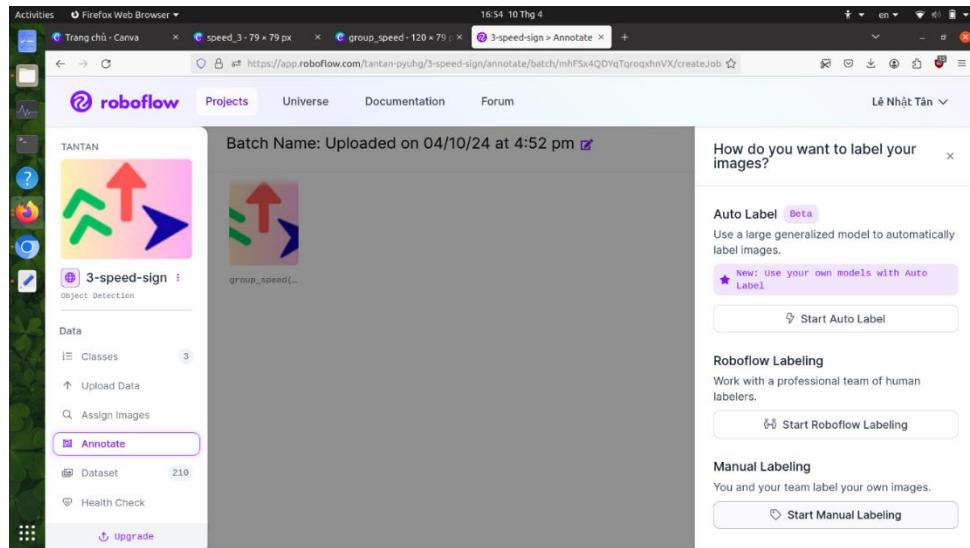
Dán nhãn dữ liệu (Labeling data) là quá trình gán nhãn hoặc gắn các nhãn (labels) vào các mẫu dữ liệu trong tập dữ liệu (Hình 7 - 7 và Hình 7 - 8). Mỗi mẫu dữ liệu thường được gắn một nhãn hoặc một tập hợp các nhãn để chỉ ra thông tin nào đó về nội dung hoặc tính chất của mẫu đó. Quá trình dán nhãn dữ liệu là một phần quan trọng của quá trình chuẩn bị dữ liệu cho các mô hình máy học, đặc biệt là trong học có giám sát (Supervised Learning), nơi mà mô hình cần được huấn luyện trên các mẫu dữ liệu đã được gán nhãn.

“Dán nhãn dữ liệu” và “dán nhãn hình ảnh” đều ý nghĩa như nhau, tuy nhiên chúng áp dụng cho các ngữ cảnh khác nhau:

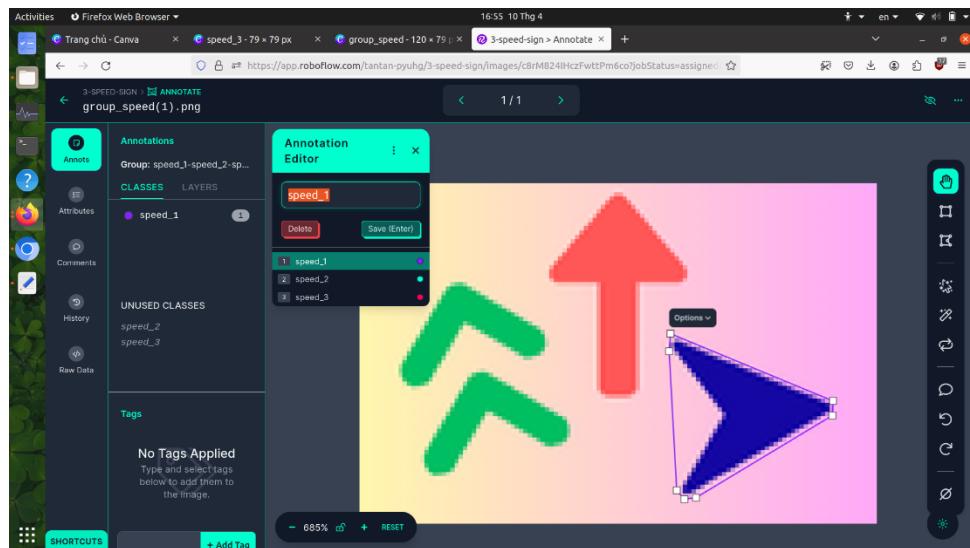
- Dán nhãn dữ liệu (Data Labeling): Là quá trình xác định dữ liệu thô (như hình ảnh, video, tệp văn bản, v.v.) và thêm một hoặc nhiều nhãn có nghĩa và chứa thông tin hữu ích để cung cấp ngữ cảnh cho mô hình máy học có thể học hỏi từ đó.
- Dán nhãn hình ảnh (Image Labeling): Là một phần của quá trình dán nhãn dữ liệu, nhưng chỉ tập trung vào việc thêm nhãn cho các hình ảnh. Nhãn có thể

cho biết ảnh chứa con chim hay ô tô, hoặc xác định vị trí của các đối tượng trong ảnh.

Vì vậy, ở đây em sử dụng cụm từ “dán nhãn hình ảnh” để phù hợp với ngữ cảnh, với sự hỗ trợ của công cụ Roboflow.

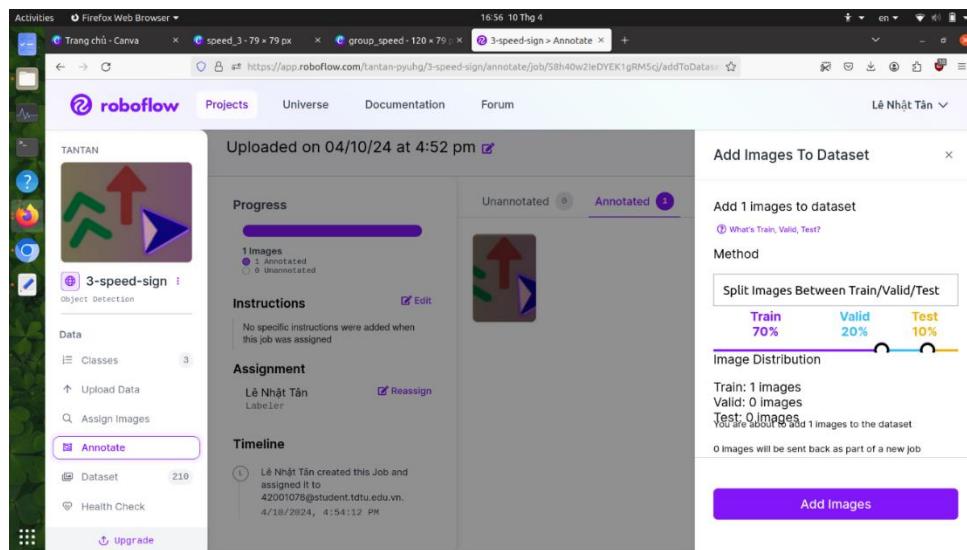


Hình 7 - 7. Chọn phương thức dán nhãn dữ liệu trên Roboflow

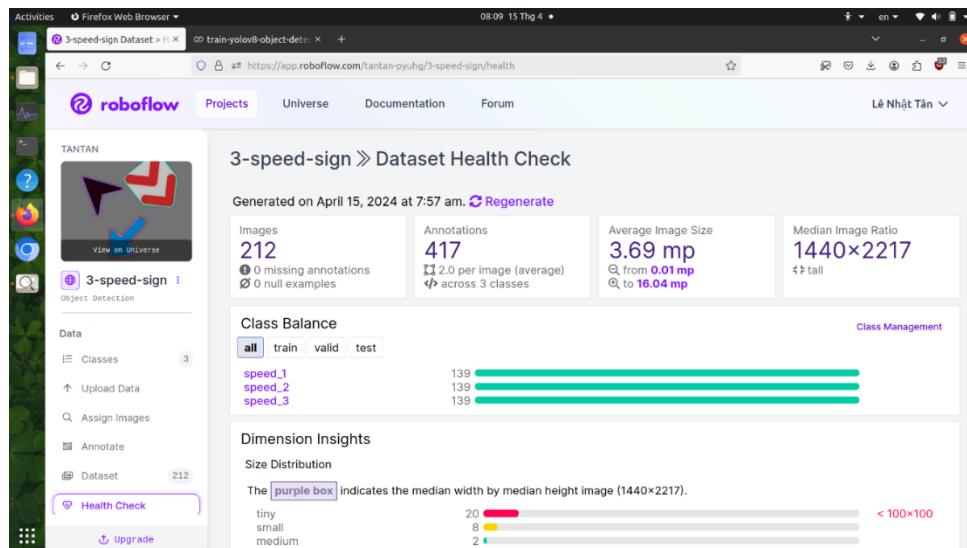


Hình 7 - 8. Thực hiện dán nhãn dữ liệu trên Roboflow

Sau khi dán nhãn dữ liệu (hình ảnh), cần phân loại các hình ảnh thành 3 tệp dữ liệu: Train, Valid, Test với tỉ lệ lần lượt tốt nhất là 70%, 20% và 10% như Hình 7 - 9. Ngoài ra em có thể kiểm tra số lượng hình ảnh với công cụ có sẵn *Health Check* trên Roboflow như Hình 7 - 10.



**Hình 7 - 9. Phân loại tệp hình ảnh sau khi dán nhãn**



**Hình 7 - 10. Kiểm tra số lượng và phân loại hình ảnh**

### 7.3.3 Tiền xử lý ảnh

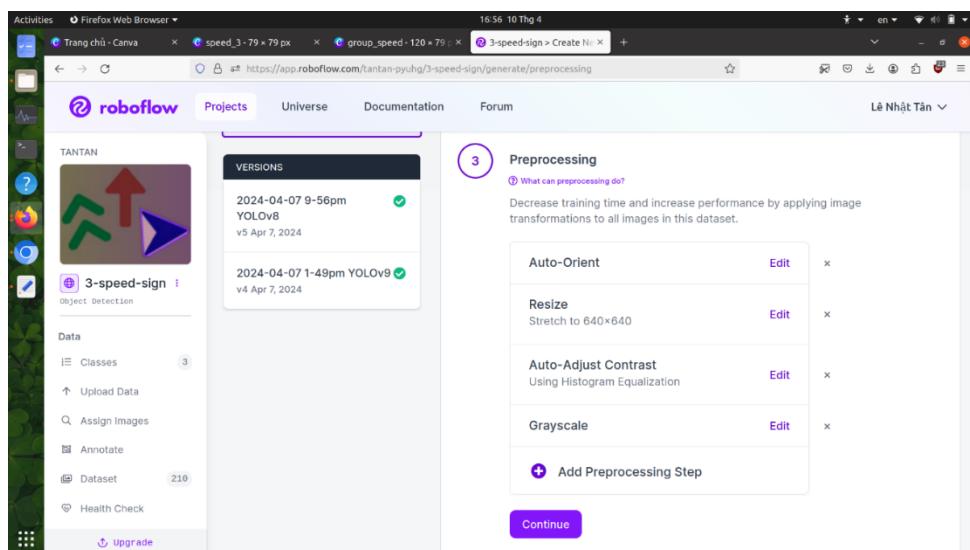
Tiền xử lý ảnh (Image preprocessing) là quá trình chuẩn bị và xử lý dữ liệu hình ảnh trước khi đưa vào mô hình máy học hoặc học sâu. Mục tiêu của tiền xử lý ảnh là làm cho dữ liệu hình ảnh trở nên dễ xử lý hơn và cải thiện hiệu suất của mô hình. Em đã áp dụng các bước tiền xử lý ảnh (Hình 7 - 11) như:

- Auto-Orient: Bước này chỉnh sửa hướng của hình ảnh để đảm bảo nó được hiển thị đúng vị trí.

- Resize: Hình ảnh được thay đổi kích thước theo một kích thước cụ thể là 640x640 pixels.
- Auto-Adjust Contrast: Bước này điều chỉnh độ tương phản của hình ảnh một cách tự động bằng cách sử dụng cân bằng histogram để nâng cao chất lượng hình ảnh. Cân bằng histogram (Histogram Equalization) là một kỹ thuật xử lý ảnh, nhằm cải thiện chất lượng ảnh bằng cách điều chỉnh histogram của ảnh về mức cân bằng. Trong lĩnh vực xử lý ảnh, histogram là biểu đồ tần xuất được dùng để thống kê số lần xuất hiện các mức sáng trong ảnh. Cân bằng histogram giúp “kéo dãn” phân bố giá trị pixel, làm cho nó không bị co cụm tại một khoảng hẹp. Điều này rất hữu ích trong các tình huống mà ảnh bị tối hoặc quá sáng do điều kiện chiếu sáng. Cân bằng histogram là một phương pháp tiền xử lý ảnh mạnh mẽ, giúp cải thiện chất lượng dữ liệu và do đó cải thiện hiệu suất của mô hình học sâu. Công thức cơ bản của cân bằng histogram là:

$$K(i) = \frac{Z(i) - \min(Z)}{\max(Z) - \min(Z)} * 255$$

- Grayscale: Hình ảnh màu được chuyển đổi thành hình ảnh xám, nơi nó được biểu diễn bằng các màu xám.

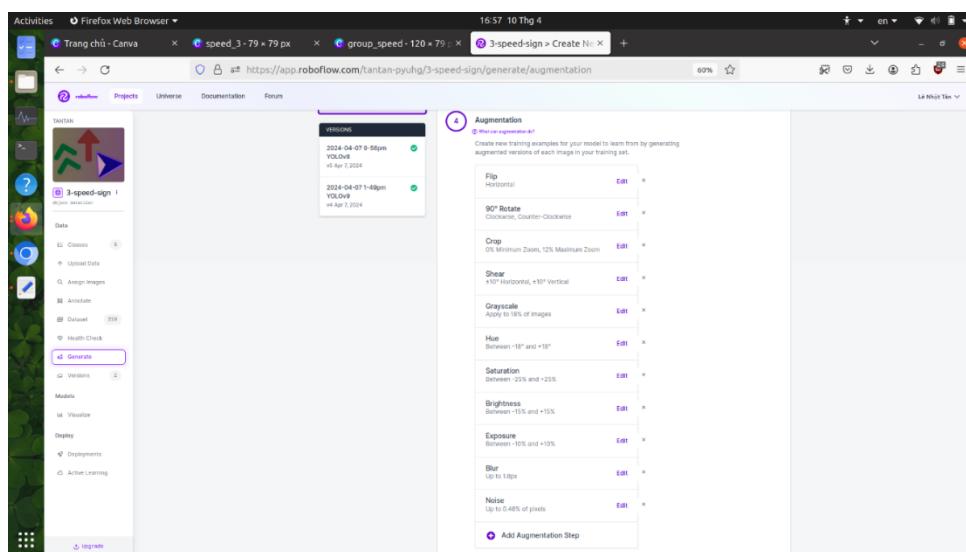


Hình 7 - 11. Các bước tiền xử lý ảnh trên Roboflow

### 7.3.4 Tăng cường hình ảnh

Tăng cường dữ liệu (Data Augmentation) là một kỹ thuật phổ biến trong máy học, đặc biệt là trong việc huấn luyện các mô hình học sâu (deep learning). Kỹ thuật này nhằm mục đích tạo ra các phiên bản mới của dữ liệu huấn luyện từ dữ liệu gốc ban đầu bằng cách áp dụng các biến đổi hoặc biến đổi lên dữ liệu.

Mục tiêu của tăng cường dữ liệu là tăng cường sự đa dạng và sự biểu diễn của dữ liệu huấn luyện, giúp mô hình học được các đặc trưng và mẫu phổ biến hơn, từ đó cải thiện khả năng tổng quát hóa của mô hình. Chính vì thế em cũng đã áp dụng một số bước để tăng cường dữ liệu như Hình 7 - 12.



**Hình 7 - 12. Tăng cường số lượng hình ảnh trên Roboflow**

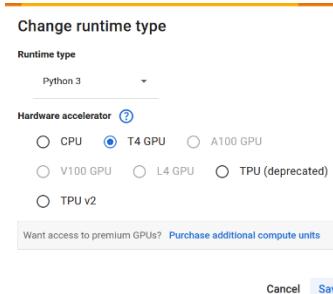
Tăng cường dữ liệu có vai trò rất quan trọng trong đào tạo. Dưới đây là một số lợi ích của việc tăng cường dữ liệu:

- Nâng cao hiệu năng của mô hình: Các kỹ thuật tăng cường dữ liệu giúp các tập dữ liệu trở nên phong phú hơn khi tạo ra nhiều biến thể của dữ liệu hiện có.
- Giảm thiểu Overfitting (tình trạng quá khóp) trong dữ liệu đào tạo: Việc tăng cường dữ liệu giúp ngăn ngừa tình trạng quá khóp khi đào tạo các mô hình ML.

- Cải thiện quyền riêng tư dữ liệu: Nếu cần đào tạo mô hình học sâu về dữ liệu nhạy cảm.
- Giảm sự phụ thuộc vào dữ liệu: Việc thu thập và chuẩn bị khối lượng lớn dữ liệu cho đào tạo có thể tốn kém và tốn thời gian.

### 7.3.5 Thực hiện đào tạo hình ảnh trên nền tảng Google Colab

Google Colab, hay còn gọi là Google Colaboratory, là một nền tảng trực tuyến được cung cấp miễn phí bởi Google, cho phép chạy các dòng code Python thông qua trình duyệt; đặc biệt phù hợp với Data analysis, machine learning và giáo dục. Người dùng có thể sử dụng tài nguyên máy tính từ CPU tốc độ cao và cả GPUs và cả TPUs đều được cung cấp miễn phí. Vì là dịch vụ miễn phí, nên Colab sẽ có những thứ tự ưu tiên trong việc sử dụng tài nguyên hệ thống, cũng như giới hạn thời gian sử dụng.

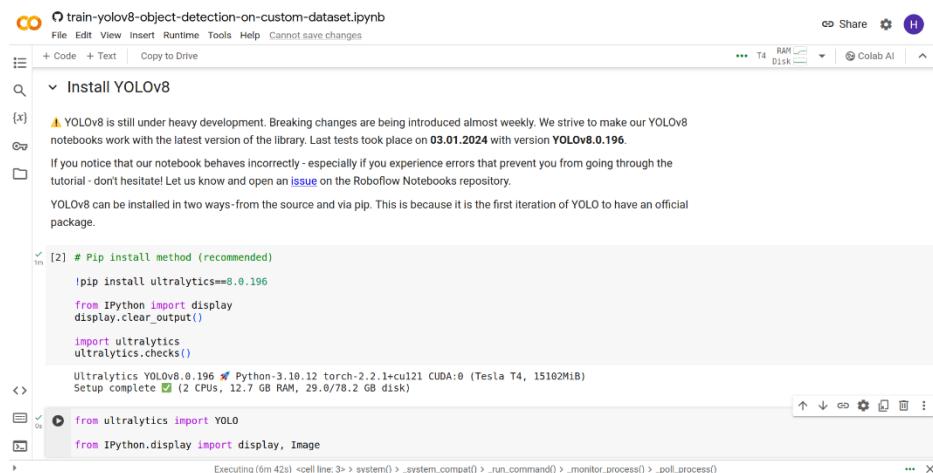


**Hình 7 - 13. Chọn sử dụng GPU trên Google Colab**

Môi trường mà Google Colab cung cấp để thực thi mã Python được gọi là Notebook, với các lệnh ‘cell’. Khi truy cập vào Notebook, đầu tiên chọn *runtime type* là GPU để sử dụng GPU cho quá trình thực thi như Hình 7 - 13. Sau đó về YOLOv8 và dataset vào ổ đĩa ảo trên Google Colab như Hình 7 - 14 và Hình 7 – 16, nhưng trước đó cần sao chép API từ Roboflow như Hình 7 - 15.

# ĐỒ ÁN CHUYÊN NGÀNH

## Trang 62



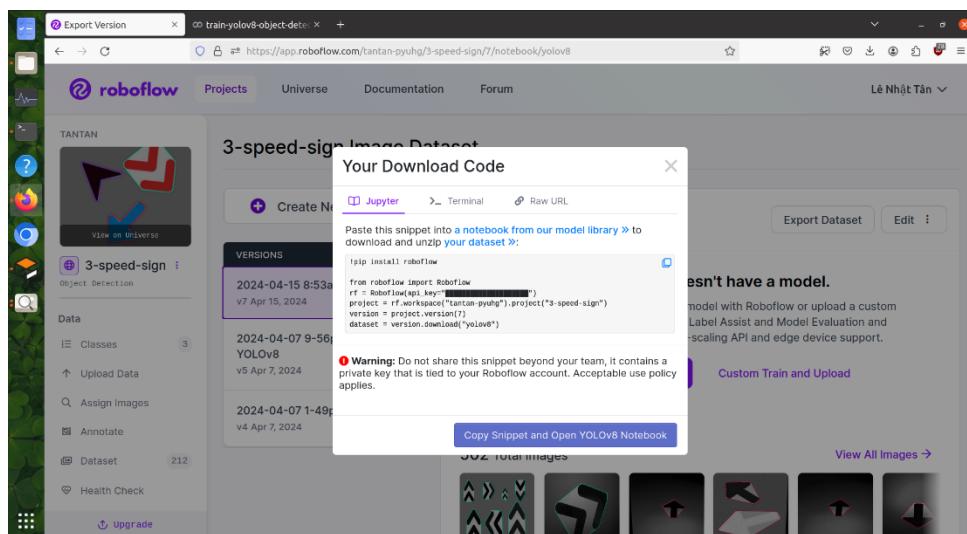
```
# Pip install method (recommended)
!pip install ultralytics==8.0.196
from IPython import display
display.clear_output()

import ultralytics
ultralytics.checks()

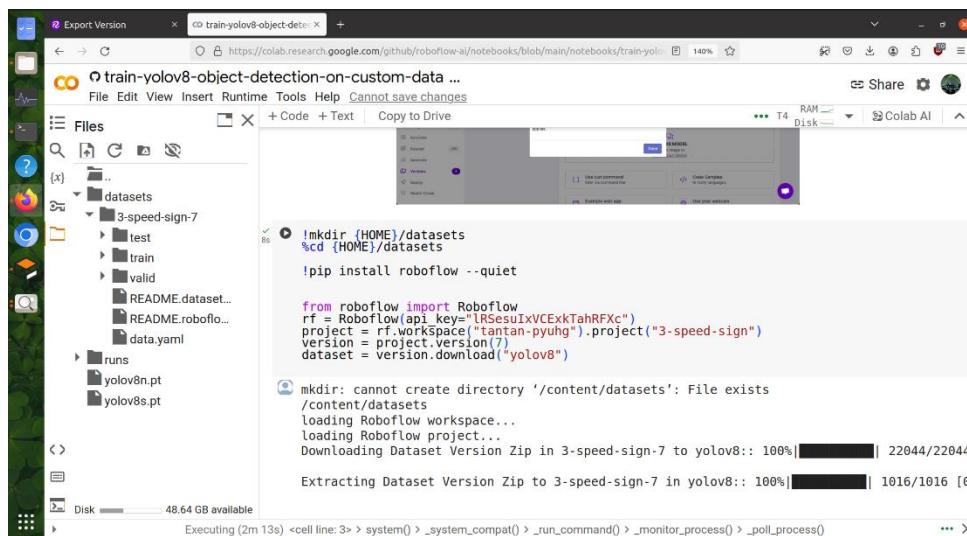
# Ultralytics YOLOv8.0.196 ✅ Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Setup complete (2 CPUs, 12.7 GB RAM, 29.0/78.2 GB disk)

from ultralytics import YOLO
from IPython.display import display, Image
```

Hình 7 - 14. Tải về YOLOv8 trên Google Colab



Hình 7 - 15. Sao chép API để tải về tệp hình ảnh dán nhãn từ Roboflow

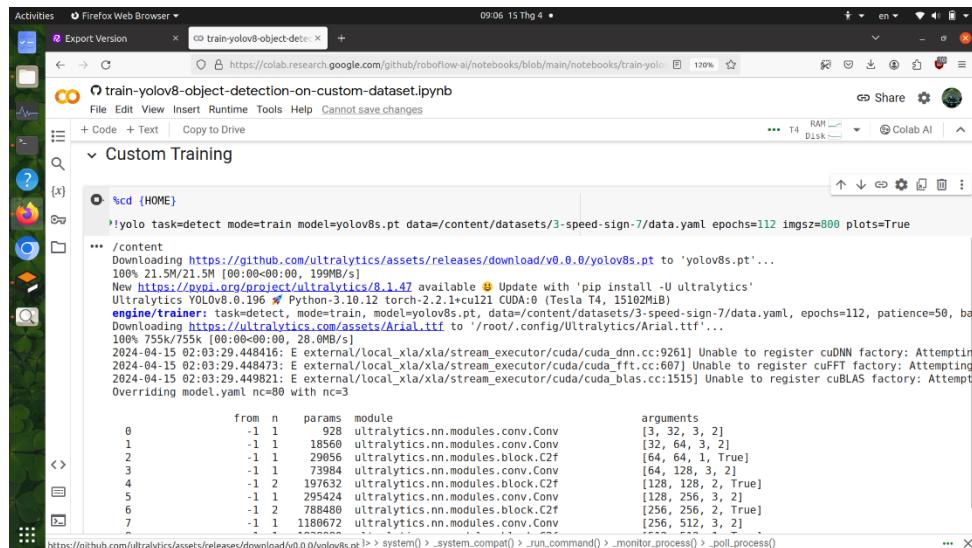


```
!mkdir {HOME}/datasets
%cd {HOME}/datasets
!pip install roboflow --quiet

from roboflow import Roboflow
rf = Roboflow(api_key="1RSesuIxVCEExkTahRFx")
project = rf.workspace("tantan-pyuhg").project("3-speed-sign")
version = project.version(7)
dataset = version.download("yolov8")
```

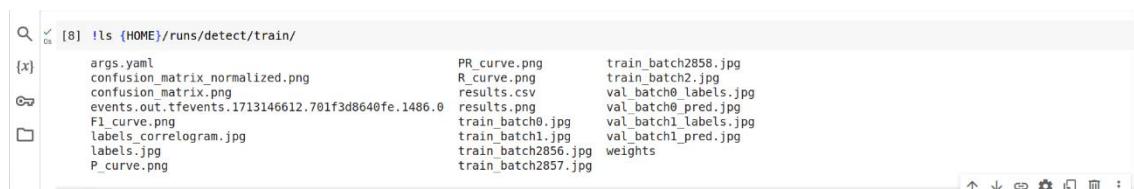
Hình 7 - 16. Tải về tệp hình ảnh dán nhãn trên Google Colab

Thực hiện đào tạo hình ảnh với model yolov8s (Hình 7 - 17) với số lần duyệt qua dataset (epochs) là 112 để đảm bảo độ chính xác của mô hình. Sau đó kiểm tra lại các tệp đã được tạo ra trong quá trình đào tạo như Hình 7 - 18.



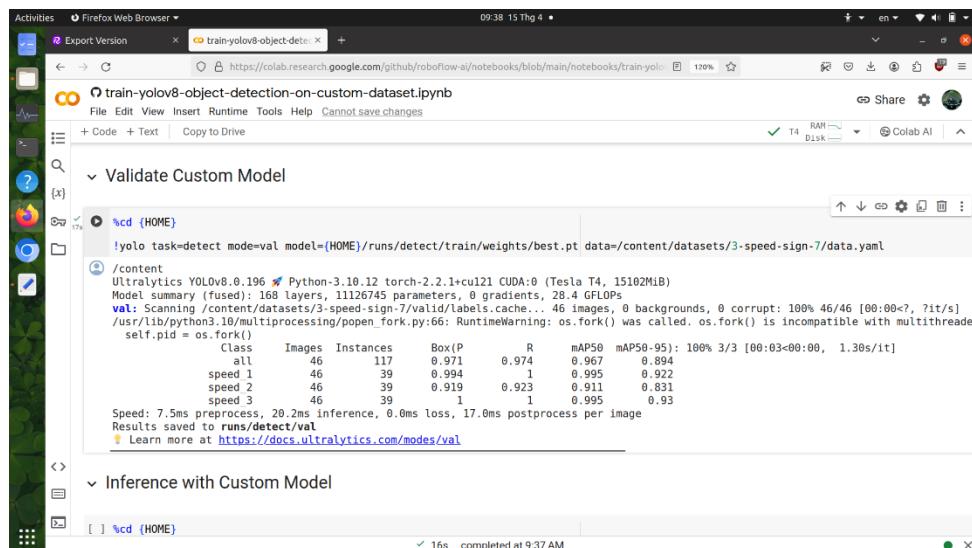
```
%cd {HOME}
!yolo task=detect mode=train model=yolov8s.pt data=/content/datasets/3-speed-sign-7/data.yaml epochs=112 imgsz=800 plots=True
...
from n   params module                                     arguments
0      -1  1     928 ultralytics.nn.modules.conv.Conv    [3, 32, 3, 2]
1      -1  1     18560 ultralytics.nn.modules.conv.Conv  [32, 64, 3, 2]
2      -1  1     29056 ultralytics.nn.modules.block.C2f  [64, 64, 1, True]
3      -1  1     73984 ultralytics.nn.modules.conv.Conv  [128, 128, 3, 2]
4      -1  2     197632 ultralytics.nn.modules.block.C2f [128, 128, 2, True]
5      -1  1     295424 ultralytics.nn.modules.conv.Conv  [128, 256, 3, 2]
6      -1  2     788480 ultralytics.nn.modules.block.C2f [256, 256, 2, True]
7      -1  1     1186672 ultralytics.nn.modules.conv.Conv [256, 512, 3, 2]
```

**Hình 7 - 17. Thực hiện đào tạo hình ảnh trên Google Colab**



```
[8] !ls {HOME}/runs/detect/train/
args.yaml
confusion_matrix_normalized.png
confusion_matrix.png
events.out.tfevents.1713146612.701f3d8640fe.1486.0
F1_curve.png
labels_correlogram.jpg
labels.jpg
P_curve.png
PR_curve.png
R_curve.png
results.csv
results.png
train_batch0.jpg
train_batch1.jpg
train_batch2856.jpg
train_batch2857.jpg
val_batch0_labels.jpg
val_batch0_pred.jpg
val_batch1_labels.jpg
val_batch1_pred.jpg
weights
```

**Hình 7 - 18. Kiểm tra các tệp mới được tạo sau khi thực hiện đào tạo hình ảnh trên Google Colab**

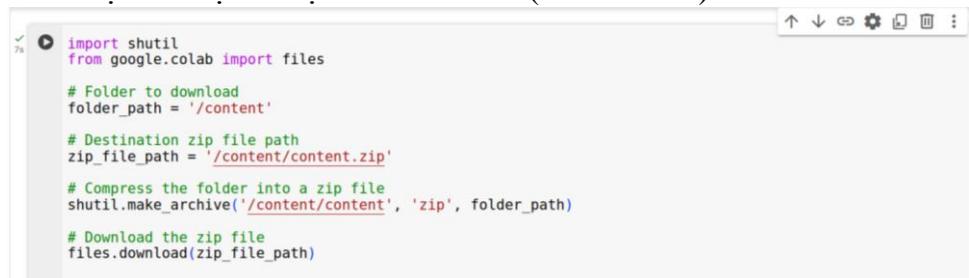


Class	Images	Instances	Box(P)	R	mAP50	mAP50-95:
all	46	117	0.971	0.974	0.967	0.894
speed 1	46	39	0.994	1	0.995	0.922
speed 2	46	39	0.919	0.923	0.911	0.831
speed 3	46	39	1	1	0.995	

**Hình 7 - 19. Thực hiện thẩm định sau khi đào tạo**

Sau khi đào tạo, cần thực hiện thẩm định (validation) như Hình 7 - 19. Mục đích của việc này là để đánh giá mức độ tổng quát của mô hình, tức là khả năng của mô hình khi áp dụng vào dữ liệu mà nó chưa từng thấy. Điều này giúp ngăn chặn hiện tượng overfitting, khi mô hình quá khớp với dữ liệu huấn luyện và không thể tổng quát hóa tốt trên dữ liệu mới. Kết quả của quá trình validation thường được dùng để điều chỉnh các tham số của mô hình, như tốc độ học, số lượng epoch,...

Chạy thêm lệnh sau để tải về thư mục content (Hình 7 - 20), là toàn bộ dữ liệu liên quan đến dữ liệu đào tạo đã tạo trên ổ đĩa ảo (Hình 7 - 21).



```

import shutil
from google.colab import files

# Folder to download
folder_path = '/content'

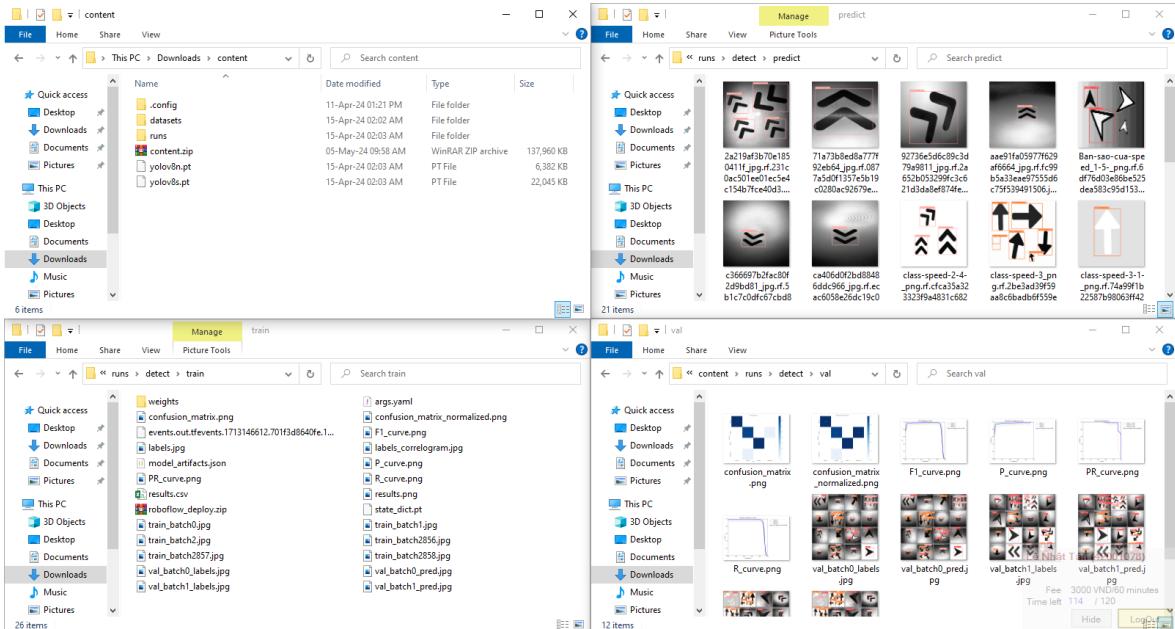
# Destination zip file path
zip_file_path = '/content/content.zip'

# Compress the folder into a zip file
shutil.make_archive('/content/content', 'zip', folder_path)

# Download the zip file
files.download(zip_file_path)

```

**Hình 7 - 20.** Tải về toàn bộ dữ liệu trên Google Colab



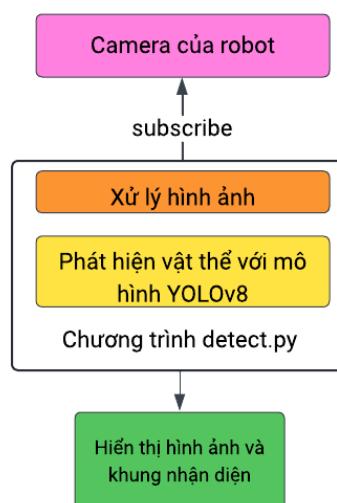
**Hình 7 - 21.** Bên trong thư mục chứa toàn bộ dữ liệu trên Google Colab đã tải về

## CHƯƠNG 8. THỰC THI PHẦN MỀM

### 8.1 Chương trình phát hiện biển báo tốc độ

#### 8.1.1 Sơ đồ nguyên lý chương trình phát hiện biển báo tốc độ

Sơ đồ nguyên lý của chương trình phát hiện biển báo tốc độ *detect.py* được biểu diễn như Hình 8 - 1.



Hình 8 - 1. Sơ đồ nguyên lý chương trình phát hiện biển báo tốc độ

#### 8.1.2 Mô tả hoạt động chương trình phát hiện biển báo tốc độ

Chương trình *detect.py* khi được khởi chạy sẽ subscribe vào topic */camera/rgb/image\_raw* để nhận message tên là *image\_msg* từ camera. Khi nhận được hình ảnh, đầu tiên chương trình sẽ chuyển đổi định dạng ảnh từ ROS sang OpenCV cũng như chuyển đổi không gian màu từ RGB sang BGR. Sau đó chương trình sẽ thu nhỏ kích thước ảnh để dễ xử lý và thực hiện nhận diện hình ảnh với mô hình YOLOv8 để phát hiện biển báo tốc độ. Chương trình cũng hiển thị một cửa sổ để hiển thị hình ảnh từ camera thu được và khoanh vùng nếu phát hiện biển báo tốc độ.

Trước khi khởi chạy *detect.py*, không có đối tượng nào subscribe vào topic */camera/rgb/image\_raw* (Hình 8 - 2). Có thể kiểm tra bằng cách chạy hai lệnh sau:

- *rostopic info /cmd\_vel*: Được sử dụng để hiển thị thông tin chi tiết về topic */cmd\_vel*. Thông tin này bao gồm loại message được sử dụng và các node đang publish hoặc subscribe topic. Topic */cmd\_vel* thường được sử dụng để điều khiển robot di chuyển.
- *rostopic info /camera/rgb/image\_raw*: Được sử dụng để hiển thị thông tin chi tiết về topic */camera/rgb/image\_raw*. Topic */camera/rgb/image\_raw* thường được sử dụng để truyền tải dữ liệu hình ảnh từ camera. message được gửi đến topic này thường là của loại *sensor\_msgs/Image*, bao gồm dữ liệu hình ảnh và thông tin liên quan.

```
thesun@thesun-lap:~$ cd dacn
thesun@thesun-lap:~/dacn$ rostopic info /camera/rgb/i
mage_raw
Type: sensor_msgs/Image
Publishers:
* /gazebo (http://localhost:40539/)
Subscribers: None
```

```
thesun@thesun-lap:~$ cd dacn
thesun@thesun-lap:~/dacn$ rostopic info /cmd_vel
Type: geometry_msgs/Twist
Publishers:
* /move_base (http://localhost:40627/)
Subscribers:
* /gazebo (http://localhost:40539/)
```

Hình 8 - 2. Topic */camera/rgb/image\_raw* và */cmd\_vel* trước khi khởi chạy *detect.py*

Sau khi khởi chạy *detect.py*, ta thấy có thêm một subscriber đăng ký vào topic */camera/rgb/image\_raw* như Hình 8 - 3.

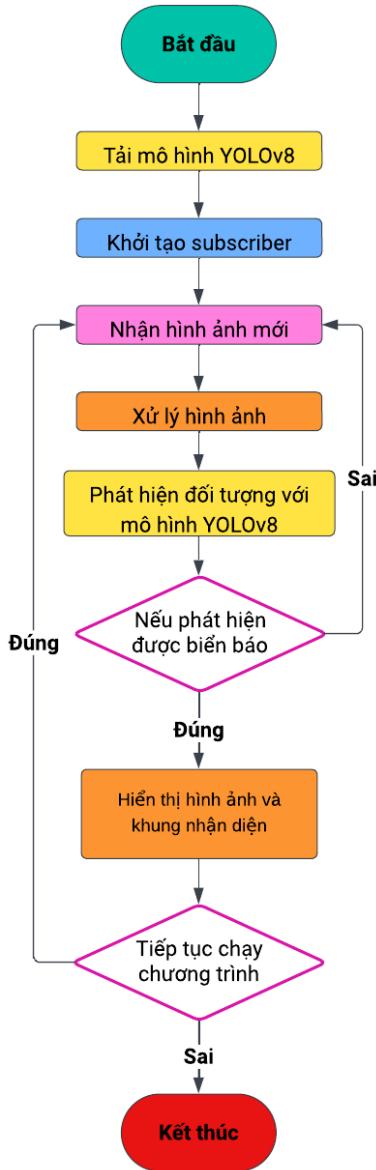
```
thesun@thesun-lap:~/dacn$ rostopic info /camera/rgb/i
mage_raw
Type: sensor_msgs/Image
Publishers:
* /gazebo (http://localhost:40539/)
Subscribers:
* /image_subscriber_6885_1714910861985 (http://local
host:38803/)
```

```
thesun@thesun-lap:~/dacn$ rostopic info /cmd_vel
Type: geometry_msgs/Twist
Publishers:
* /move_base (http://localhost:40627/)
Subscribers:
* /gazebo (http://localhost:40539/)
```

Hình 8 - 3. Topic */camera/rgb/image\_raw* và */cmd\_vel* sau khi khởi chạy *detect.py*

### 8.1.3 Lưu đồ thuật toán chương trình phát hiện biển báo tốc độ

Lưu đồ thuật toán của chương trình phát hiện biển báo tốc độ *detect.py* được biểu diễn như Hình 8 - 4 dưới đây.

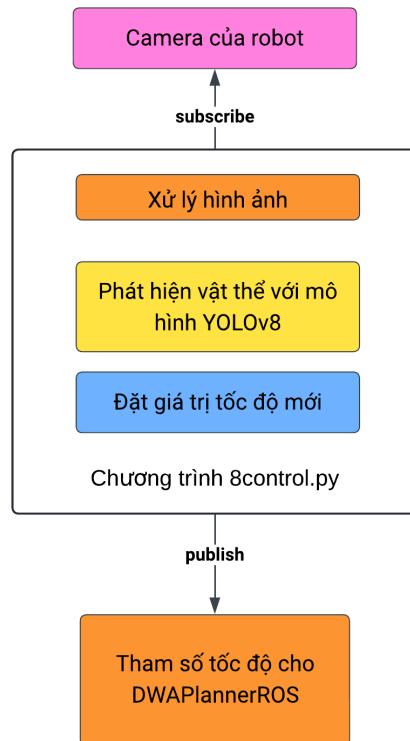


Hình 8 - 4. Lưu đồ thuật toán chương trình phát hiện biển báo tốc độ

## 8.2 Chương trình phát hiện biển báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3

### 8.2.1 Sơ đồ nguyên lý chương trình phát hiện biển báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3

Sơ đồ nguyên lý chương trình phát hiện biển báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3 `8control.py` được biểu diễn như Hình 8 - 5.



**Hình 8 - 5. Sơ đồ nguyên lý chương trình phát hiện biến báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3**

### **8.2.2 Mô tả hoạt động chương trình phát hiện biến báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3**

Chương trình *8control.py* khi được khởi chạy sẽ subscribe vào topic */camera/rgb/image\_raw* để nhận message tên là *image\_msg* từ camera (Hình 8 - 6).

```

thesun@thesun-lap:~/dacn$ rostopic info /camera/rgb/image_raw
Type: sensor_msgs/Image
Publishers:
* /gazebo (http://localhost:40539/)
Subscribers:
* /image_subscriber_6885_1714910861985 (http://localhost:38803/)
* /image_subscriber_7617_1714911059484 (http://localhost:33235/)

thesun@thesun-lap:~/dacn$ rostopic info /cmd_vel
Type: geometry_msgs/Twist
Publishers:
* /move_base (http://localhost:40627/)
Subscribers:
* /gazebo (http://localhost:40539/)

thesun@thesun-lap:~/dacn$ 
thesun@thesun-lap:~/dacn$ 

```

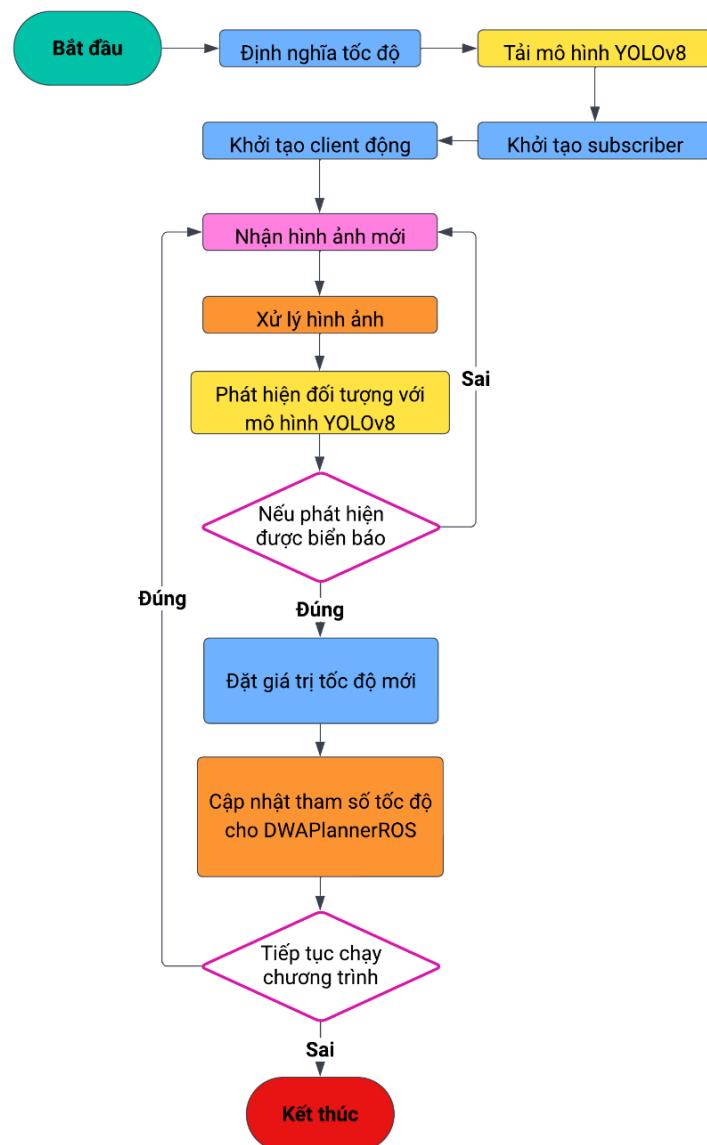
**Hình 8 - 6. Topic */camera/rgb/image\_raw* và */cmd\_vel* sau khi khởi chạy *detect.py* và *8control.py***

Khi nhận được hình ảnh, đầu tiên chương trình sẽ chuyển đổi định dạng ảnh từ ROS sang OpenCV cũng như chuyển đổi không gian màu từ RGB sang BGR. Sau đó chương trình sẽ thu nhỏ kích thước ảnh để dễ xử lý và thực hiện nhận diện hình ảnh với mô hình YOLOv8 để phát hiện biến báo tốc độ và publish message chứa thông

tin về tốc độ mới đến node `/move_base/DWAPlannerROS` để điều chỉnh tham số `max_vel_x`. Từ đó tốc độ của robot tự hành Turtlebot3 sẽ được thay đổi mỗi khi phát hiện biến báo tốc độ mới (giữ tốc độ trước đó cho đến khi phát hiện biến báo mới).

### 8.2.3 Lưu đồ thuật toán chương trình phát hiện biến báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3

Lưu đồ thuật toán của chương trình phát hiện biến báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3 `8control.py` được biểu diễn như Hình 8 - 7 dưới đây.

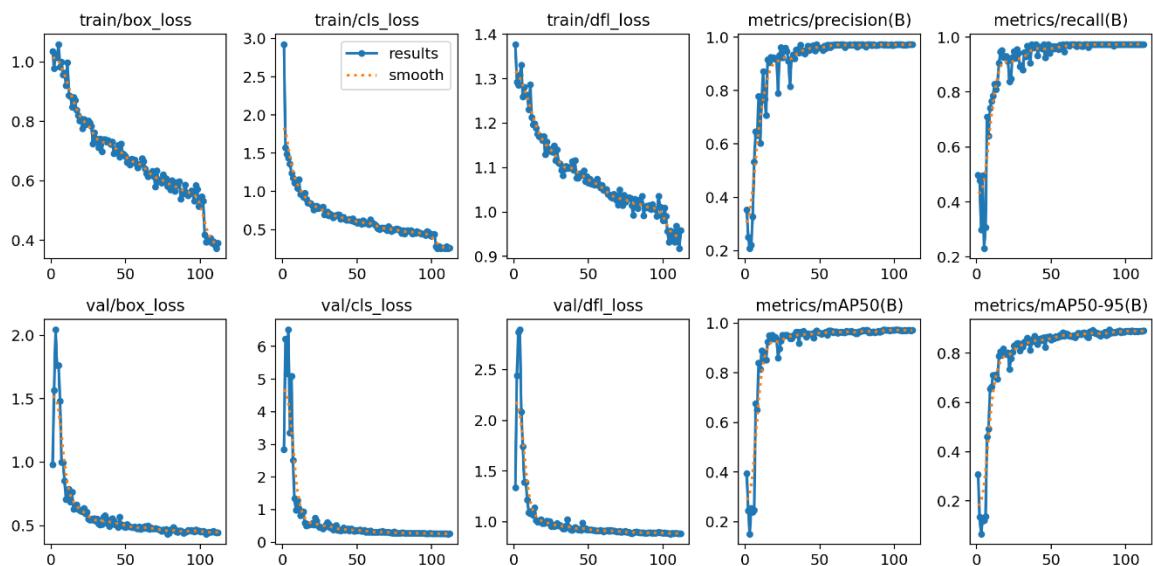


**Hình 8 - 7. Lưu đồ thuật toán chương trình phát hiện biến báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3**

## CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG

### 9.1 Mô hình phát hiện biển báo sau khi đào tạo

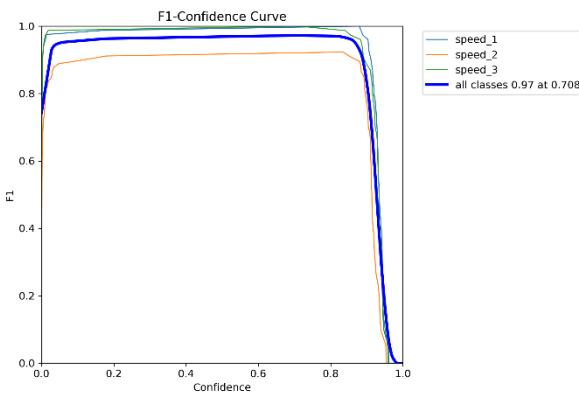
Sau khi đào tạo với YOLOv8, em thu được một mô hình YOLOv8 đã được huấn luyện là *best.pt*, mô hình này có thể được sử dụng để phát hiện các đối tượng trong hình ảnh và video. Hình 9 - 1 là hình ảnh *results.png* thu được sau khi đào tạo.



Hình 9 - 1. Kết quả tổng quan quá trình đào tạo

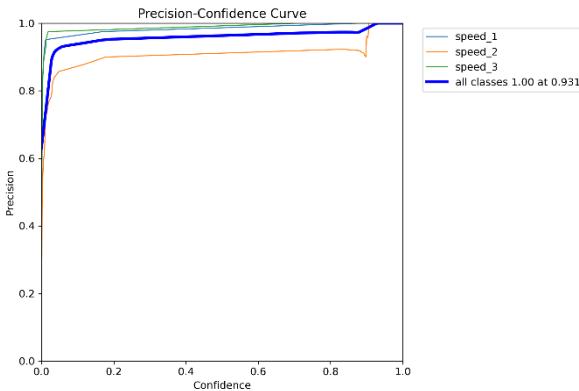
Ngoài ra, em cũng sẽ thu được các kết quả dự đoán, thường được lưu trong một thư mục cụ thể. Các kết quả này cho phép đánh giá hiệu suất của mô hình đã được đào tạo, có thể kể đến như:

- F1\_curve (Hình 9 - 2): Là một chỉ số đánh giá quan trọng thường được sử dụng trong các tác vụ phân loại để đánh giá hiệu suất của mô hình. Nó kết hợp độ chính xác (precision) và độ phủ (recall) thành một giá trị duy nhất.



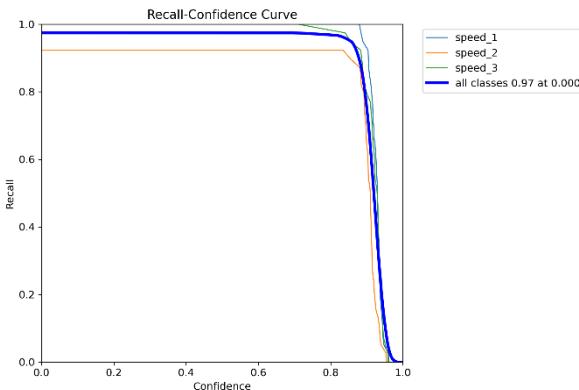
Hình 9 - 2. Biểu đồ thể hiện giá trị F1

- P\_curve (Hình 9 - 3): Đây là biểu đồ thể hiện giá trị Precision trong quá trình huấn luyện và kiểm thử mô hình.



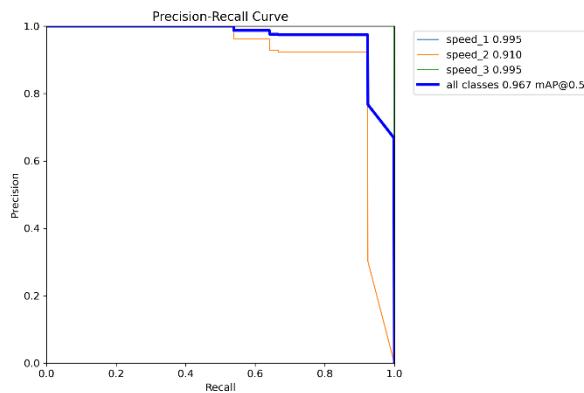
Hình 9 - 3. Biểu đồ thể hiện giá trị Precision

- R\_curve (Hình 9 - 4): Là biểu đồ thể hiện giá trị Recall hoặc Learning curve (đường học) trong quá trình huấn luyện và kiểm thử mô hình.



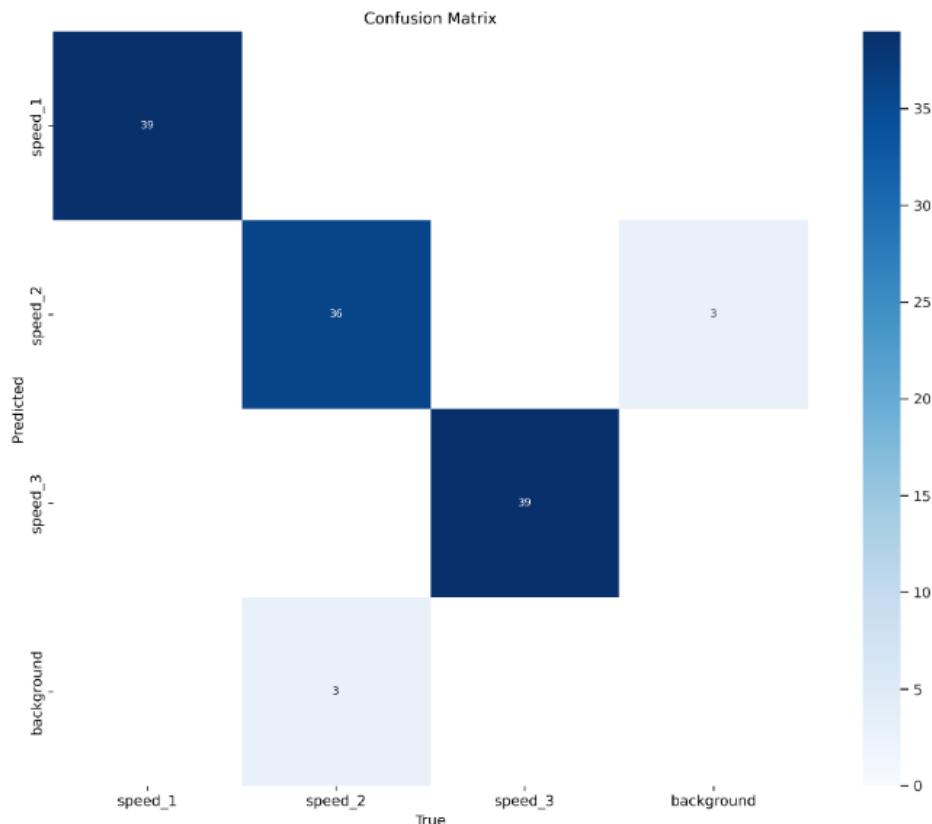
Hình 9 - 4. Biểu đồ thể hiện giá trị Recall

- PR\_curve (Hình 9 - 5): Đây là biểu đồ thể hiện mối quan hệ giữa Precision và Recall. Biểu đồ PR chứa giá trị Precision trên trục y và giá trị Recall trên trục x2.



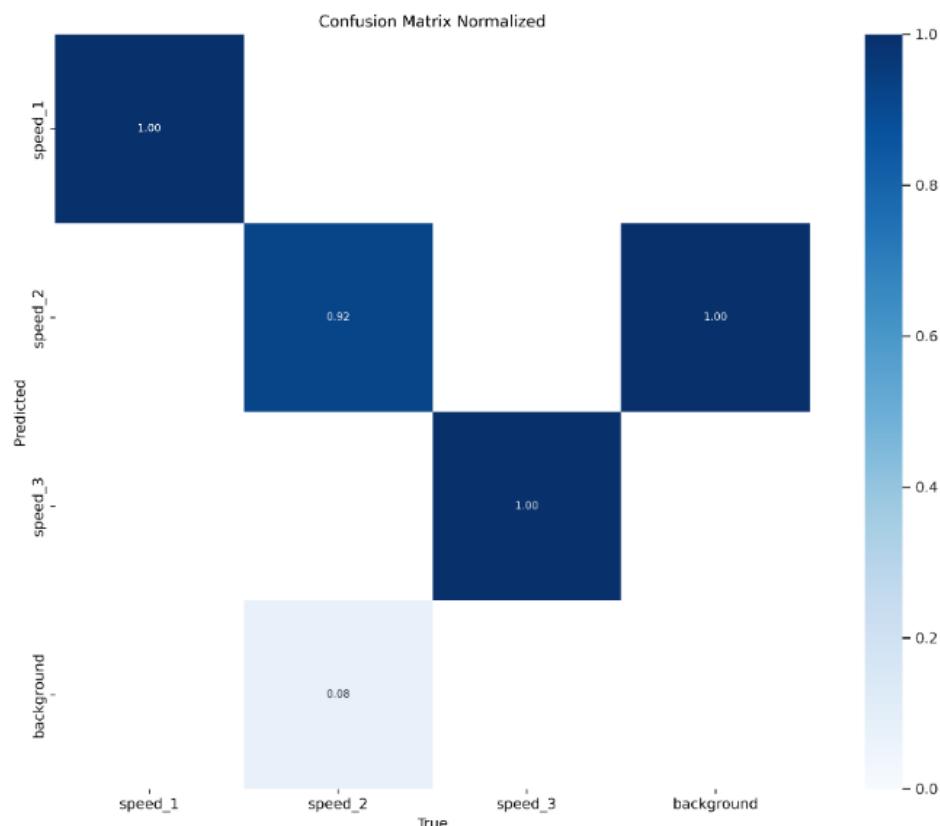
Hình 9 - 5. Biểu đồ thể hiện mối quan hệ giữa Precision và Recall

- confusion\_matrix (Hình 9 - 6): Ma trận nhầm lẫn là một công cụ đánh giá hiệu suất của mô hình phân loại. Nó cung cấp thông tin chi tiết về số lượng dự đoán đúng và sai dựa trên dự đoán của mô hình.



Hình 9 - 6. Ma trận nhầm lẫn

- confusion\_matrix\_normalized (Hình 9 - 7): Đây là phiên bản chuẩn hóa của ma trận nhầm lẫn, giúp dễ dàng so sánh hiệu suất giữa các mô hình khác nhau hoặc giữa các lớp khác nhau trong cùng một mô hình.



Hình 9 - 7. Phiên bản chuẩn hóa của ma trận nhầm lẫn

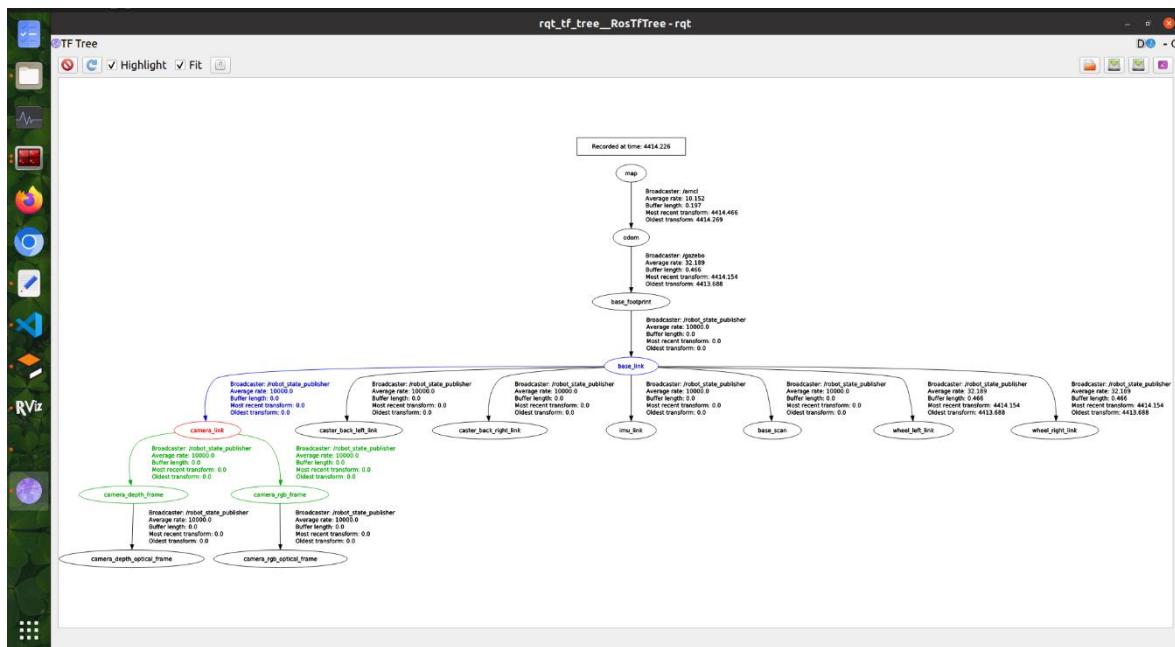
Mô hình phát hiện biển báo sau khi được đào tạo đã cho ra các chỉ số kết quả như sau:

- mAP = 96.7% (đây là chỉ số để đo hiệu suất của các mô hình thị giác máy tính, mAP bằng với trung bình của chỉ số Average Precision trên tất cả các lớp trong một mô hình).
- Precision = 97.1%.
- Recall = 97.4%.

## 9.2 Cây TF hoàn chỉnh của chương trình mô phỏng

Cây TF hoàn chỉnh của chương trình mô phỏng có hình dạng như ở Hình 9 - 8. Khi quan sát cây TF hoàn chỉnh này có thể thấy được mối quan hệ giữa các khung tọa độ, các khung tọa độ đó là:

- map.
- odom.
- base\_footprint.
- base\_link gồm các khung tọa độ khác như: camera\_link (gồm camera\_depth\_frame + camera\_depth\_optical\_frame + camera\_rgb\_frame + camera\_rgb\_optical\_frame), caster\_back\_left\_link, caster\_back\_right\_link, imu\_link, base\_scan, wheel\_left\_link, wheel\_right\_link.

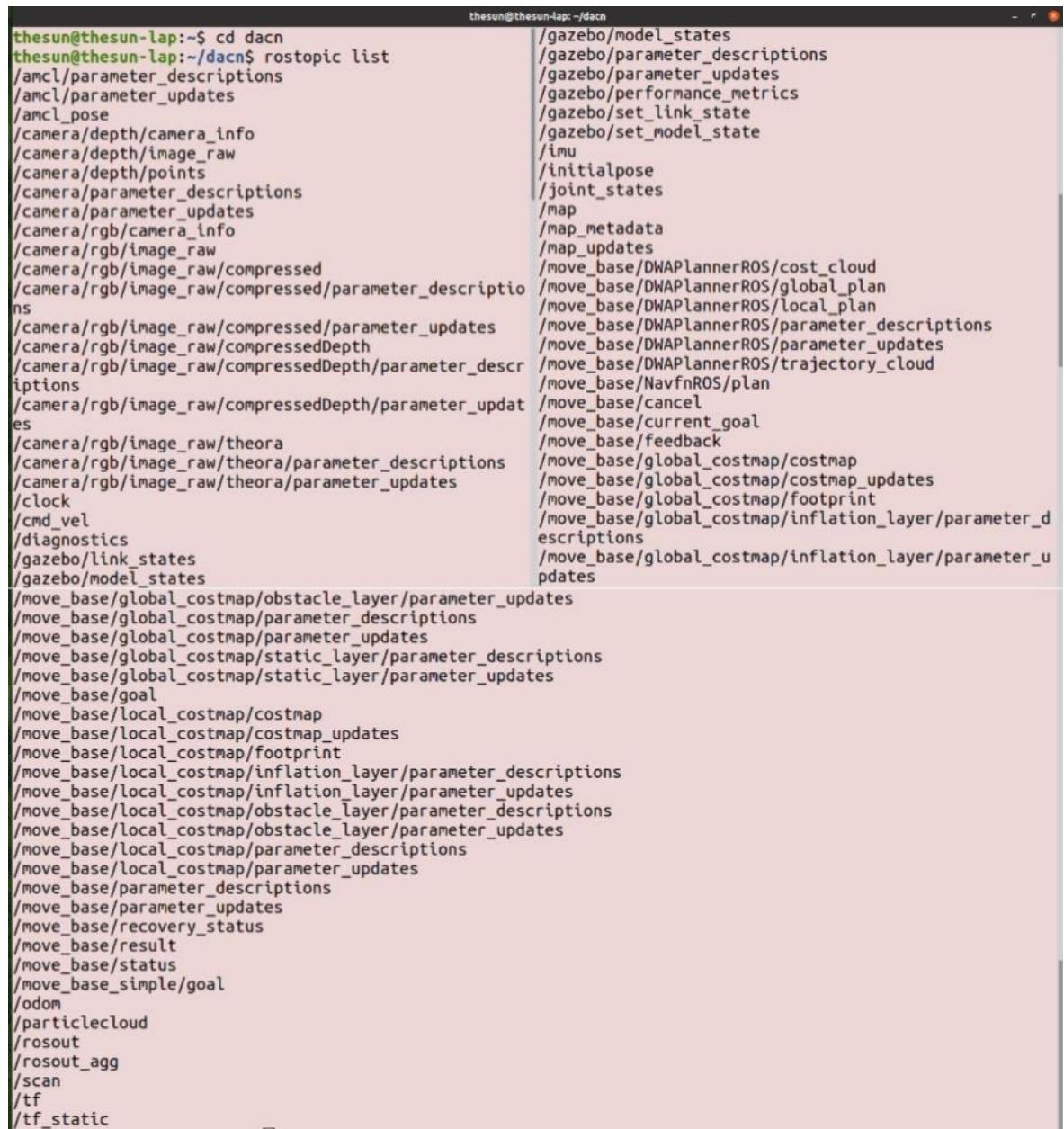


Hình 9 - 8. Cây TF hoàn chỉnh của đồ án

## 9.3 Các topic được sử dụng

Khi khởi chạy mô phỏng robot, sẽ có các topic hoạt động. Để xem danh sách các topic đang sử dụng, cần sử dụng lệnh *rostopic list*.

Nhìn vào Hình 9 - 9 chúng ta thấy có đầy đủ các topic đang được sử dụng như là topic cung cấp hình ảnh từ camera, topic cung cấp vị trí, topic điều khiển của gói *move\_base*, topic lập đường đi cho robot, topic theo dõi tốc độ robot.

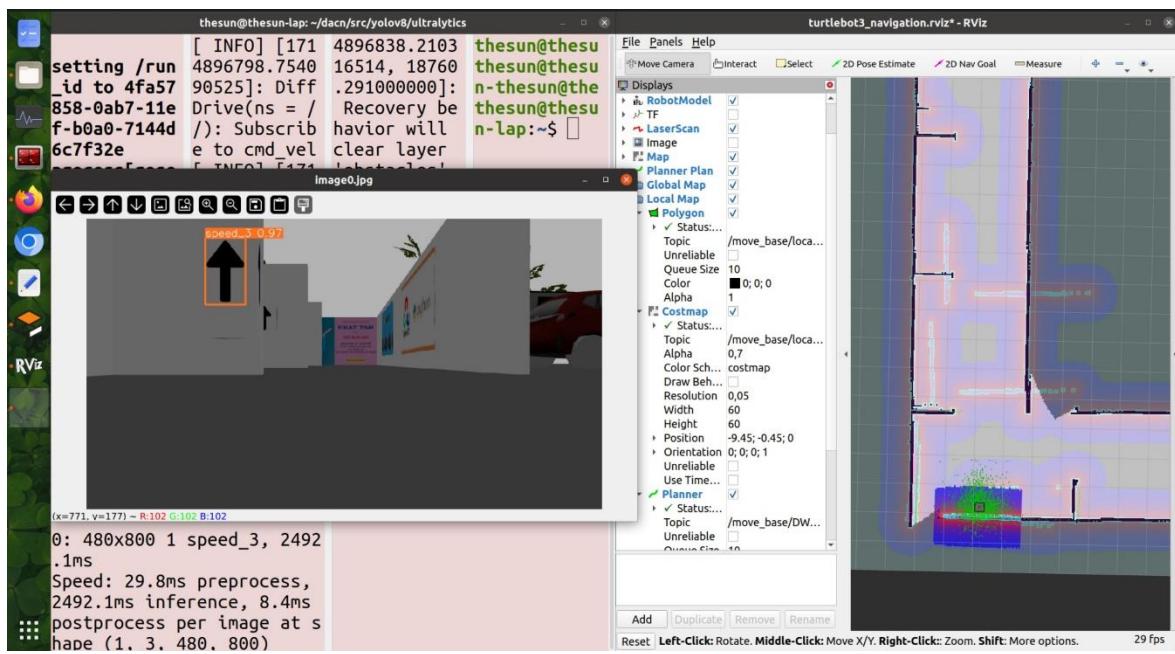


```
thesun@thesun-lap:~$ cd dacn
thesun@thesun-lap:~/dacn$ rostopic list
/amcl/parameter_descriptions
/amcl/parameter_updates
/amcl_pose
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/parameter_descriptions
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates
/camera/rgb/image_raw/theora
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
/clock
/cmd_vel
/diagnostics
/gazebo/link_states
/gazebo/model_states
/move_base/global_costmap/obstacle_layer/parameter_descriptions
/move_base/global_costmap/parameter_descriptions
/move_base/global_costmap/parameter_updates
/move_base/global_costmap/static_layer/parameter_descriptions
/move_base/global_costmap/static_layer/parameter_updates
/move_base/goal
/move_base/local_costmap/costmap
/move_base/local_costmap/costmap_updates
/move_base/local_costmap/footprint
/move_base/local_costmap/inflation_layer/parameter_descriptions
/move_base/local_costmap/inflation_layer/parameter_updates
/move_base/local_costmap/obstacle_layer/parameter_descriptions
/move_base/local_costmap/obstacle_layer/parameter_updates
/move_base/local_costmap/parameter_descriptions
/move_base/local_costmap/parameter_updates
/move_base/parameter_descriptions
/move_base/parameter_updates
/move_base/recovery_status
/move_base/result
/move_base/status
/move_base_simple/goal
/odom
/particlecloud
/rosout
/rosout_agg
/scan
/tf
/tf_static
```

Hình 9 - 9. Danh sách các topic đang sử dụng

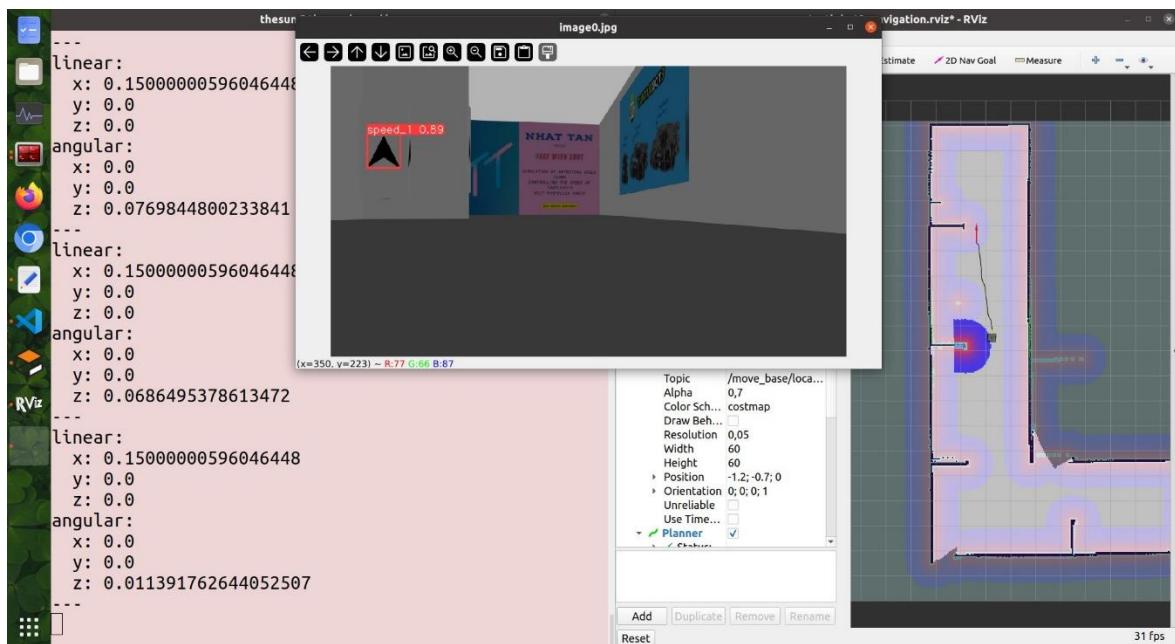
#### 9.4 Kết quả mô phỏng phát hiện biển báo tốc độ

Khi khởi chạy chương trình *detect.py* để phát hiện biển báo tốc độ, thu được kết quả như Hình 9 - 10.



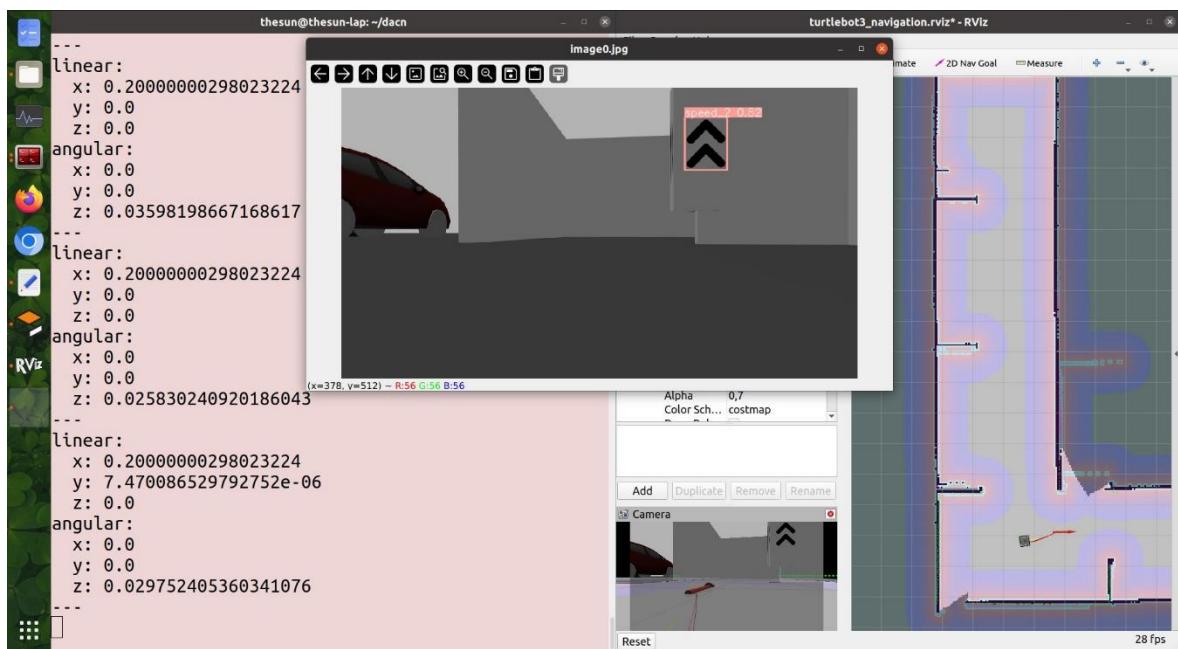
**Hình 9 - 10. Kết quả thực thi chương trình phát hiện biển báo tốc độ**

## 9.5 Kết quả mô phỏng phát hiện biển báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3

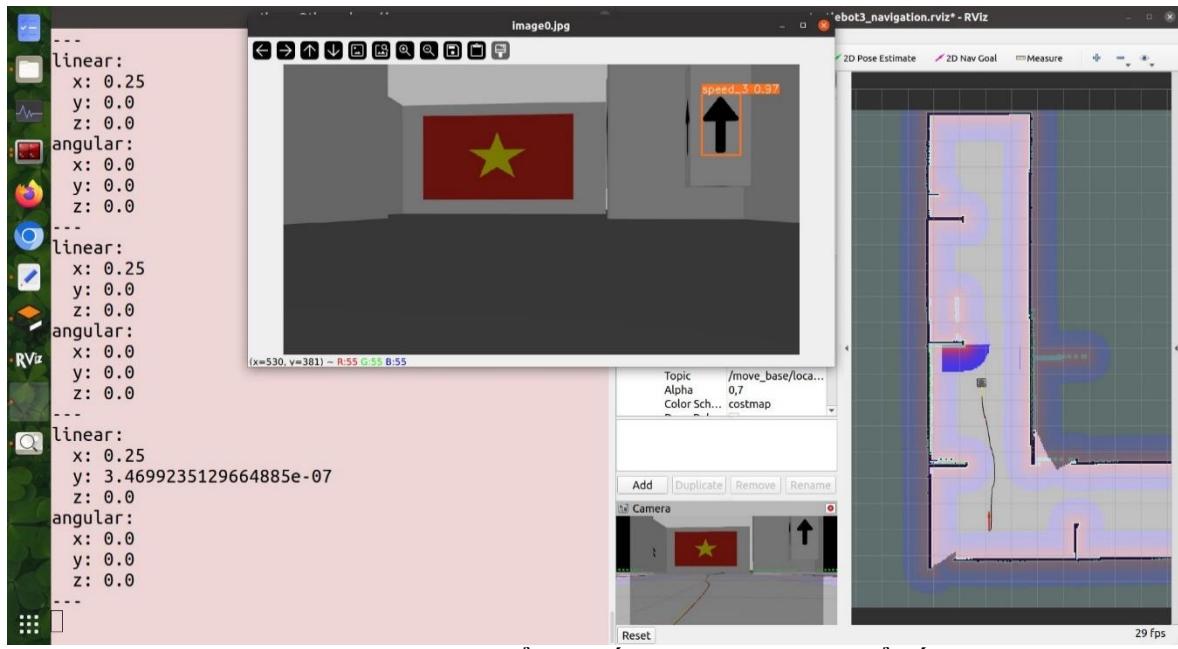


**Hình 9 - 11. Chương trình phát hiện được biến báo tốc độ speed\_1 và thay đổi tốc độ robot thành 0.15 m/s**

Khi khởi chạy chương trình *8control.py* để phát hiện biển báo tốc độ điều khiển tốc độ robot tự hành Turtlebot3, thu được các kết quả như Hình 9 - 11, Hình 9 - 12 và Hình 9 - 13.



**Hình 9 - 12. Chương trình phát hiện được biển báo tốc độ speed\_2 và thay đổi tốc độ robot thành 0.20 m/s**



**Hình 9 - 13. Chương trình phát hiện được biển báo tốc độ speed\_3 và thay đổi tốc độ robot thành 0.25 m/s**

## **CHƯƠNG 10. KẾT LUẬN**

### **10.1 Kết luận**

Đồ án hoàn thành và đạt được những kết quả như sau:

- Tự thiết kế được môi trường mô phỏng cho robot trong Gazebo, môi trường hoạt động ổn định.
- Xây dựng được bản đồ định trước SLAM (bằng phương pháp gmapping).
- Định vị được robot (ACML) trong bản đồ đã định trước, robot có thể tự động di chuyển điểm điểm đến đích đã chọn.
- Thiết kế được các biển báo và đưa vào môi trường mô phỏng.
- Đào tạo được mô hình phát hiện vật thể với độ chính xác 97%.
- Chương trình có thể phát hiện tất cả các biển báo trong các tình huống được mô phỏng và hiển thị đúng kết quả đã phát hiện.
- Chương trình có thể điều khiển chính xác tốc độ của robot tự hành Turtlebot3 theo kết quả phát hiện biển báo, với 3 tốc độ là:
  - 0.25 m/s.
  - 0.20 m/s.
  - 0.15 m/s.

### **10.2 Ưu điểm**

Những ưu điểm của đồ án khi được khai thác thực tế có thể kể đến như:

- Tăng cường khả năng tự động hóa: Việc tích hợp khả năng phát hiện biển báo và điều chỉnh tốc độ một cách tự động giúp robot hoạt động hiệu quả hơn trong môi trường thực tế.
- Cải thiện an toàn giao thông: Phát hiện biển báo tốc độ giúp robot tuân thủ các quy định về tốc độ, từ đó giảm thiểu rủi ro va chạm và tăng cường an toàn cho người dùng và robot.

Ngoài ra, khi xét về mặt mô phỏng thì đồ án đã thể hiện được một số tính chất:

- **Ứng dụng công nghệ tiên tiến:** Sử dụng các thuật toán như học máy và xử lý ảnh để nhận diện biển báo, cho thấy sự tiên tiến trong ứng dụng công nghệ.
- **Phát triển kỹ năng lập trình và thiết kế hệ thống:** Đồ án tạo cơ hội cho sinh viên và nhà nghiên cứu phát triển kỹ năng trong lĩnh vực lập trình và thiết kế hệ thống tự động hóa.

### **10.3 Nhược điểm**

Bên cạnh những ưu điểm có được, đồ án tồn tại một số nhược điểm như:

- **Còn sai sót trong việc nhận diện vật thể:** Hệ thống xảy ra trường hợp nhận diện nhầm vật thể, nhưng xác suất xảy ra lỗi nhận diện này rất thấp và không ảnh hưởng nhiều đến hiệu suất tổng thể của hệ thống.
- **Chưa thiết kế được robot riêng mà phải dựa vào gói TurtleBot3 có sẵn.**

### **10.4 Ứng dụng**

Đồ án “Mô phỏng phát hiện biển báo tốc độ điều khiển tốc độ robot tự hành TurtleBot3” có thể có nhiều ứng dụng thực tế, đặc biệt trong lĩnh vực tự động hóa và robot học. Một số ứng dụng tiềm năng có thể kể đến như:

- **Tự động hóa giao thông:** Robot có thể được sử dụng để giám sát và điều khiển tốc độ xe cộ trong các khu vực có giới hạn tốc độ, giúp tăng cường an toàn giao thông.
- **Nghiên cứu và giáo dục:** TurtleBot3 là một nền tảng phổ biến cho việc giảng dạy và nghiên cứu về robot học, và đồ án này có thể được sử dụng như một phần của chương trình học hoặc dự án nghiên cứu.
- **Phát triển robot thông minh:** Việc tích hợp khả năng phát hiện biển báo và điều chỉnh tốc độ có thể giúp phát triển các thuật toán thông minh hơn cho robot tự hành, từ đó cải thiện khả năng tự động hóa và độc lập của chúng.
- **Ứng dụng trong công nghiệp:** Robot có thể được sử dụng trong các nhà máy và kho bãi để tuân thủ các quy định về tốc độ di chuyển, đảm bảo an toàn cho người lao động và hàng hóa.

Ngoài ra, việc mô phỏng trong môi trường kiểm soát cũng giúp nhà phát triển hiểu rõ hơn về cách thức hoạt động của robot trong các tình huống thực tế và cách thức phản ứng với các biến số môi trường, từ đó có thể tối ưu hóa và cải thiện hiệu suất làm việc của robot trong tương lai.

## 10.5 Hướng phát triển

Đồ án có thể đạt được hiệu suất sử dụng tốt hơn khi được phát triển những góc độ sau:

- Lập trình robot tự hành di chuyển đa điểm.
- Cải thiện thuật toán phát hiện biển báo: Tăng cường số lượng hình ảnh đào tạo và để nâng cao độ chính xác và tốc độ phản hồi của hệ thống phát hiện biển báo.
- Tối ưu hóa thuật toán điều khiển tốc độ: Sử dụng các kỹ thuật điều khiển tiên tiến để cải thiện khả năng điều chỉnh tốc độ của robot một cách mượt mà và chính xác.
- Phát triển giao diện người dùng: Tạo ra một giao diện người dùng thân thiện, cho phép người dùng dễ dàng theo dõi và điều chỉnh các thông số của robot.
- Mở rộng khả năng phát hiện và điều khiển: Mở rộng khả năng phát hiện của robot để nhận diện nhiều loại biển báo khác nhau, không chỉ giới hạn ở biển báo tốc độ mà còn có các biển báo điều khiển robot.

Nếu áp dụng đồ án vào việc tạo ra một sản phẩm thực tế, cần phát triển ở các khía cạnh như:

- Nâng cấp phần cứng: Sử dụng các cảm biến có nhạy tốt, camera với độ phân giải cao hơn để tăng khả năng nhận diện và xử lý thông tin từ môi trường. Dùng vi xử lý mạnh hơn để tăng tốc độ nhận diện.
- Tích hợp với các hệ thống tự động khác: Kết nối và làm việc cùng với các hệ thống tự động hóa khác trong môi trường thực tế như xe tự lái hoặc hệ thống giao thông thông minh.

## **TÀI LIỆU THAM KHẢO**

### **Tiếng Anh**

- Basheer, M. M., & Varol, A. (2019, November). *An overview of robot operating system forensics*. In 2019 1st International Informatics and Software Engineering Conference (UBMYK)
- Cacace, J., & Joseph, L. (2018). *Mastering Ros for Robotics Programming, second edition: Design, build, and simulate complex robots using the robot operating system*, 2nd edition. Packt Publishing.
- Çalış, M. (2020). *Roboter mit Ros: Bots Konstruieren und MIT open source Programmieren*. dpunkt.verlag.
- Koubaa, A. (2021). *Robot Operating System (ROS): The complete reference*. Springer.
- Quigley, M., Gerkey, B., & Smart, W. D. (2015). *Programming Robots with ROS*. O'Reilly & Associates Incorporated.

**PHỤ LỤC A MÃ NGUỒN PYTHON CHƯƠNG TRÌNH PHÁT HIỆN  
BIỂN BÁO TỐC ĐỘ**

```
#!/usr/bin/python3

import rospy
import cv2
from cv_bridge import CvBridge, CvBridgeError
from sensor_msgs.msg import Image
from ultralytics import YOLO

model = YOLO("/home/thesun/dacn/src/yolov8/ultralytics/best.pt")

def callback(image_msg):
    """This function is called to handle the subscribed messages
    Args:
        image_msg (Image): message type Image from sensor_msgs
    """
    try:
        cv_image = bridge.imgmsg_to_cv2(image_msg)
        cv_image = cv2.cvtColor(cv_image, cv2.COLOR_RGB2BGR)

        scale_percent = 50
        width = int(cv_image.shape[1] * scale_percent / 100)
        height = int(cv_image.shape[0] * scale_percent / 100)
        small_image = cv2.resize(cv_image, (width, height))


```

```
results = model(small_image, conf=0.79, show=True) # predict on the resized  
image with confidence threshold  
  
cv2.imshow('YOLO Detection', results)  
cv2.waitKey(10)  
  
except CvBridgeError as error:  
    print(error)  
  
if __name__=="__main__":  
    bridge = CvBridge()  
    rospy.init_node("image_subscriber", anonymous=True)  
    print("Subscribe images from topic /camera/rgb/image_raw ...")  
    image_subscriber = rospy.Subscriber("/camera/rgb/image_raw", Image, callback)  
  
try:  
    # spin() simply keeps python from exiting until this node is stopped  
    rospy.spin()  
except KeyboardInterrupt:  
    print("Finished!")
```

**PHỤ LỤC B MÃ NGUỒN PYTHON CHƯƠNG TRÌNH PHÁT HIỆN  
BIỂN BÁO TỐC ĐỘ ĐIỀU KHIỂN TỐC ĐỘ ROBOT TỰ HÀNH  
TURTLEBOT3**

```
#!/usr/bin/python3

import rospy
import cv2
from cv_bridge import CvBridge, CvBridgeError
from sensor_msgs.msg import Image
from ultralytics import YOLO
from dynamic_reconfigure.client import Client

speed_1 = 0.15
speed_2 = 0.20
speed_3 = 0.25

model = YOLO("/home/thesun/dacn/src/yolov8/ultralytics/best.pt")
bridge = CvBridge()
dynamic_param_client = None

def callback(image_msg):
    """This function is called to handle the subscribed messages
    Args:
        image_msg (Image): message type Image from sensor_msgs
    """
    global dynamic_param_client

    try:
        cv_image = bridge.imgmsg_to_cv2(image_msg)
```

```
cv_image = cv2.cvtColor(cv_image, cv2.COLOR_RGB2BGR)

scale_percent = 50 # adjust as needed
width = int(cv_image.shape[1] * scale_percent / 100)
height = int(cv_image.shape[0] * scale_percent / 100)
small_image = cv2.resize(cv_image, (width, height))

results = model(small_image, conf=0.79, verbose=False)

if results:
    detections = results[0].boxes.data.tolist()

    for detection in detections:
        class_id = int(detection[5])
        if class_id == 0:
            new_speed = speed_1
        if class_id == 1:
            new_speed = speed_2
        if class_id == 2:
            new_speed = speed_3

        if dynamic_param_client:
            config = {'max_vel_x': new_speed}
            dynamic_param_client.update_configuration(config)

    except CvBridgeError as error:
        print(error)

if __name__ == "__main__":
```

```
rospy.init_node("image_subscriber", anonymous=True)
print("Subscribe images from topic /camera/rgb/image_raw ...")

dynamic_param_client = Client("/move_base/DWAPlannerROS")
image_subscriber = rospy.Subscriber("/camera/rgb/image_raw", Image, callback)

try:
    rospy.spin()
except KeyboardInterrupt:
    print("Finished!")
```