



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

-----♦-----

ĐỒ ÁN CHUYÊN NGÀNH  
KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG

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

GVHD  
TS. Lê Anh Vũ

HK 2 2023-2024

SVTH  
Lê Nhật Tân



# Nội dung trình bày

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 BIỂN BÁO TỐC ĐỘ.
8. THỰC THI PHẦN MỀM.
9. KẾT QUẢ MÔ PHỎNG.
10. KẾT LUẬN.



1!

# GIỚI THIỆU ĐỀ TÀI



# 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.

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.

## 1.2 Mục tiêu tổng quát của đề tài

- 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.



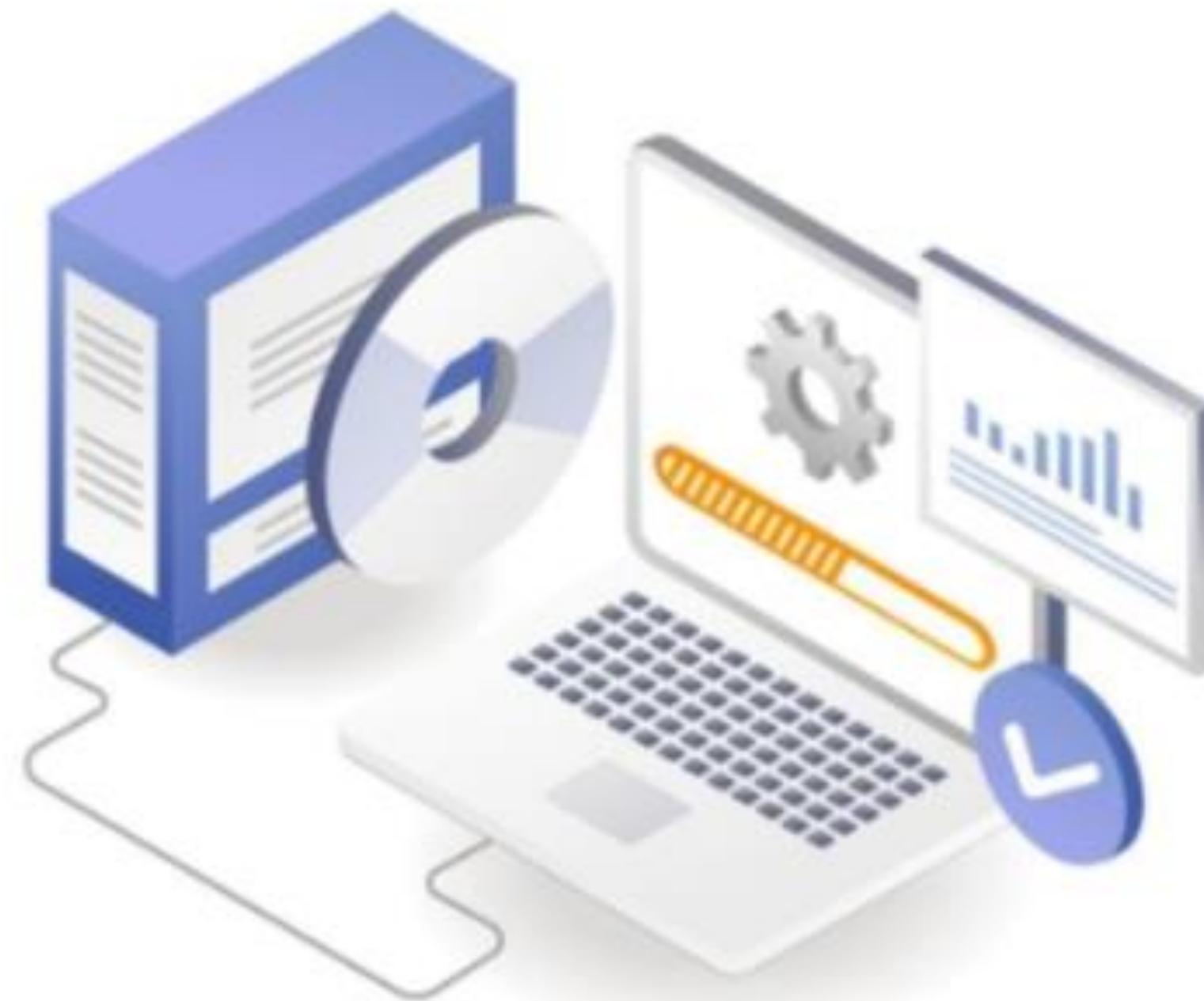
- Cài đặt Ubuntu 20.04, ROS1 Neotic cùng với Gazebo và RVIZ.
- Tải và sử dụng các gói mô phỏng của Turtlebot3.
- Tạo môi trường làm việc để các thành phần (robot và môi trường giả lập) giao tiếp với nhau.
- Xây dựng bản đồ mô phỏng trong Gazebo để gắn các biển báo và mô phỏng robot tự hành Turtlebot3.
- Xây dựng bản đồ định vị 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 điểm 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

## 1.3 Mục tiêu cụ thể của đề tài





# CÀI ĐẶT MÔI TRƯỜNG



# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG



Cài đặt Ubuntu 20.04



# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

## 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



**ROS được thiết kế theo mô-đun vì hầu hết các hệ thống robot được chia thành nhiều thành phần khác nhau**

# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG



GAZEBO

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

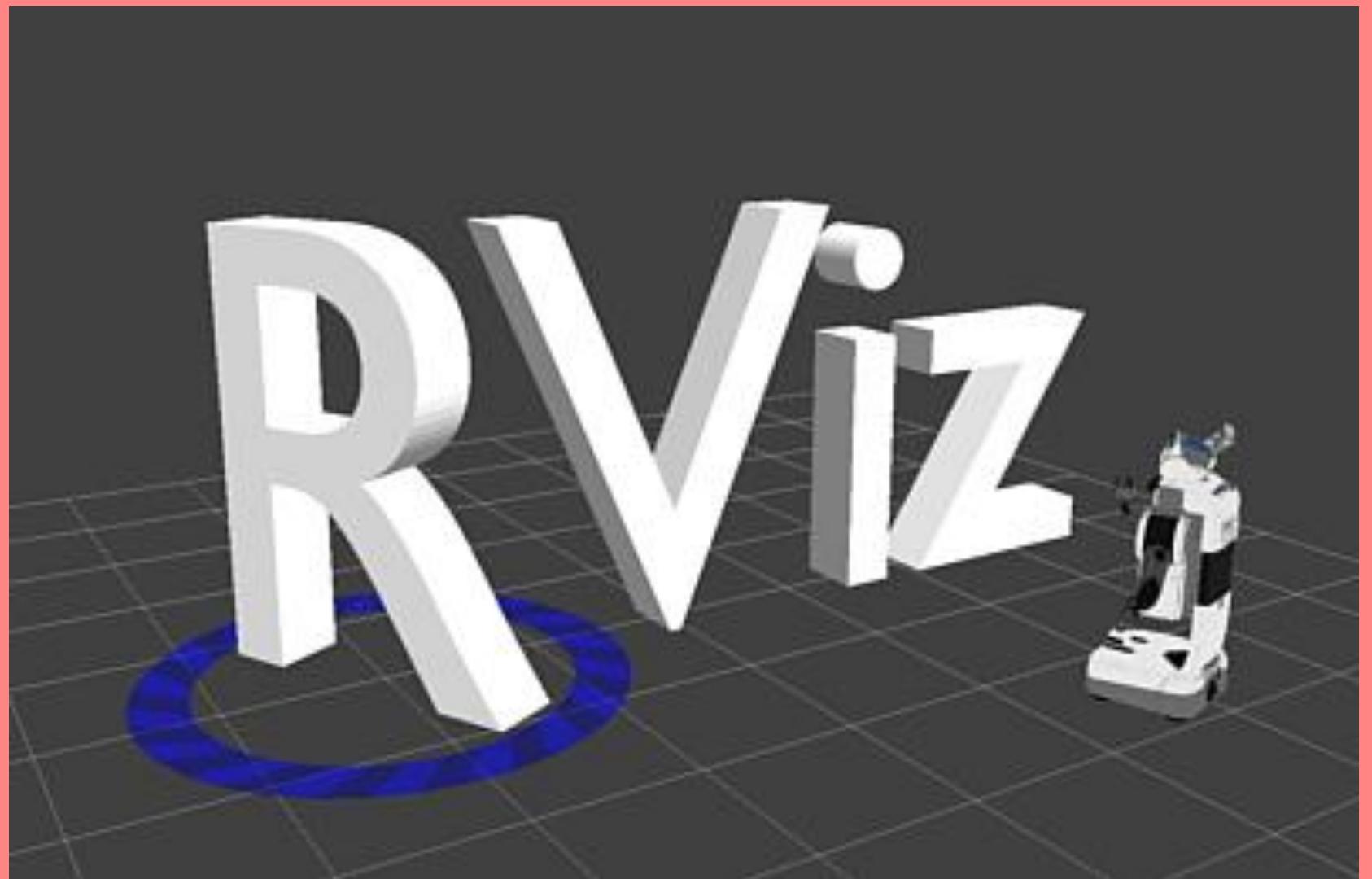
Khi cài đặt ROS, Gazebo được mặc định cài đặt kèm theo, nhưng vẫn có thể cài đặt riêng biệt với lệnh: `sudo apt-get install gazebo11`, với phiên bản mới nhất là Gazebo 11

# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

## 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`.



# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

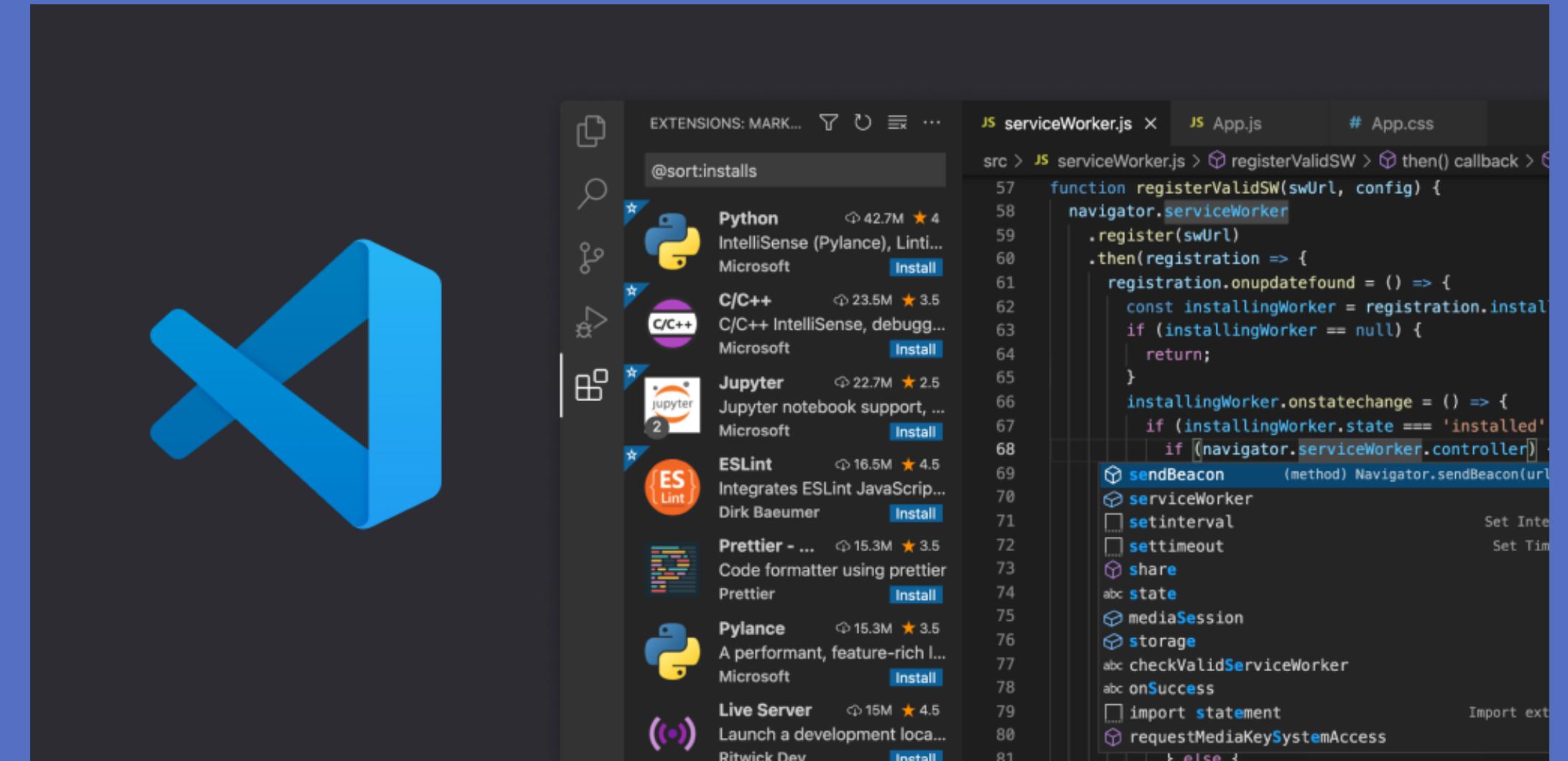
Visual Studio Code (VS Code) là một trình soạn thảo mã nguồn miễn phí, mã nguồn mở được phát triển bởi Microsoft. Nó có sẵn cho Windows, macOS và LINUX.

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



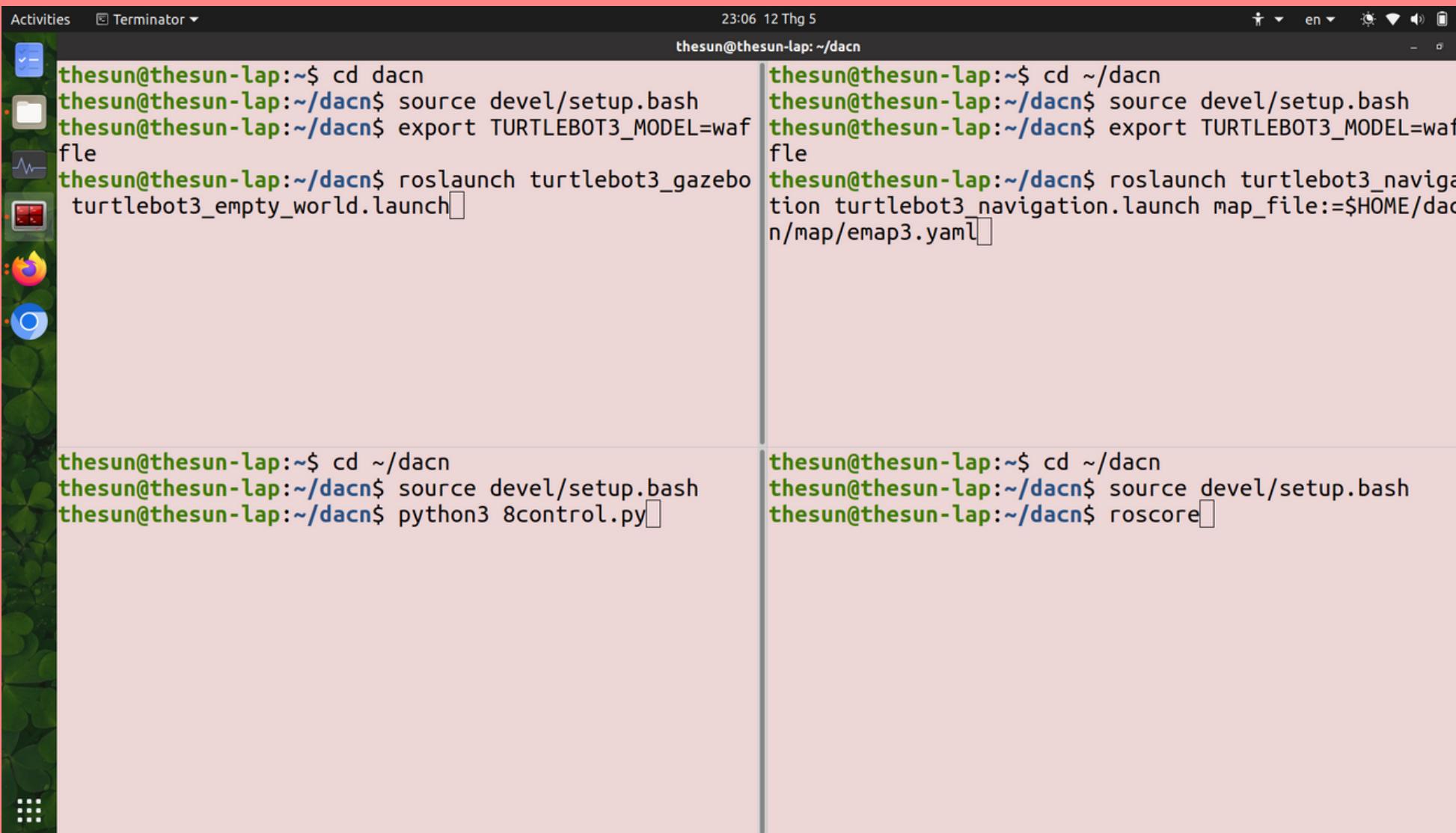
## Cài đặt VS Code



The screenshot shows the Visual Studio Code interface. On the left, there's a large blue 'X' logo. On the right, the code editor displays a snippet of JavaScript code related to service workers. The Extensions sidebar on the left lists several extensions: Python, C/C++, Jupyter, ESLint, Prettier, Pylance, and Live Server, each with its name, size, rating, and an 'Install' button. The main code editor area shows a file named 'serviceWorker.js' with some code and line numbers from 57 to 81.

# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

## Thiết lập môi trường làm việc cho ROS



```
Activities Terminator 23:06 12 Thg 5 thesun@thesun-lap:~/dacn
thesun@thesun-lap:~$ cd dacn
thesun@thesun-lap:~/dacn$ source devel/setup.bash
thesun@thesun-lap:~/dacn$ export TURTLEBOT3_MODEL=waffle
thesun@thesun-lap:~/dacn$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch

thesun@thesun-lap:~/dacn$ cd ~dacn
thesun@thesun-lap:~/dacn$ source devel/setup.bash
thesun@thesun-lap:~/dacn$ export TURTLEBOT3_MODEL=waffle
thesun@thesun-lap:~/dacn$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/dacn/map/emap3.yaml

thesun@thesun-lap:~/dacn$ cd ~dacn
thesun@thesun-lap:~/dacn$ source devel/setup.bash
thesun@thesun-lap:~/dacn$ python3 8control.py

thesun@thesun-lap:~/dacn$ cd ~dacn
thesun@thesun-lap:~/dacn$ source devel/setup.bash
thesun@thesun-lap:~/dacn$ roscore
```

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

# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

## 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.`

# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

## Cài đặt Gói Turtlebot

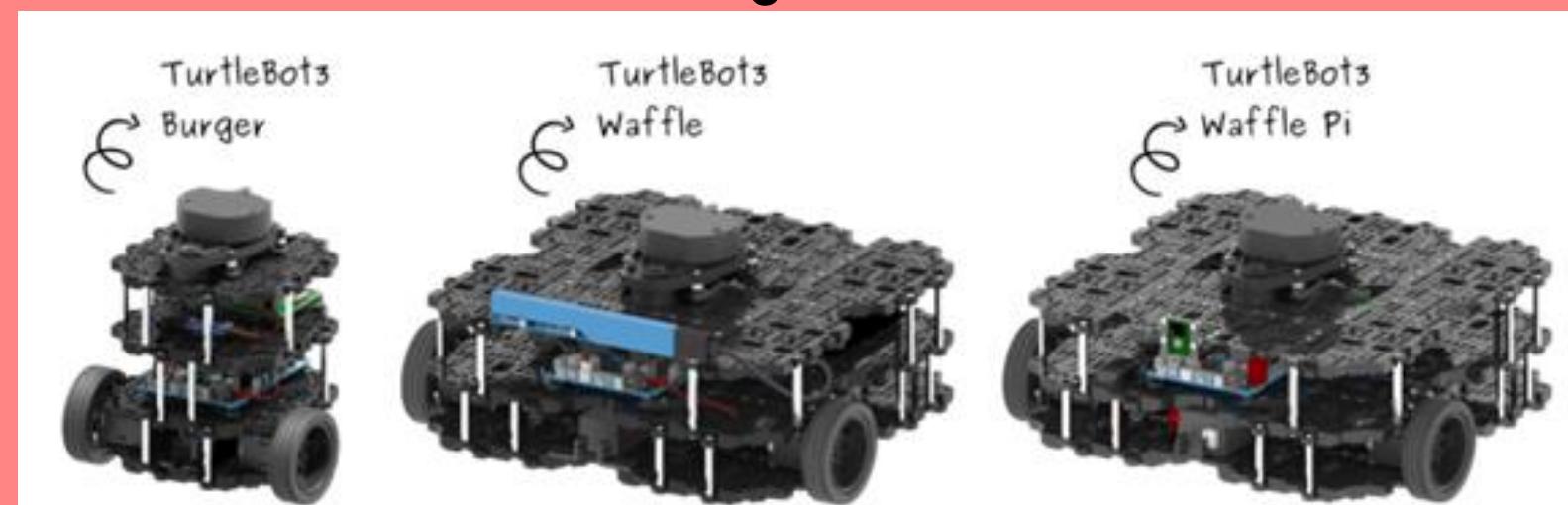
TurtleBot là một bộ dụng cụ robot cá nhân, giá rẻ với phần mềm mã nguồn mở.  
Được tạo ra bởi Melonee Wise và Tully Foote tại Willow Garage vào tháng 11 năm 2010

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

Hiện nay có  
4 thế hệ  
TurtleBots

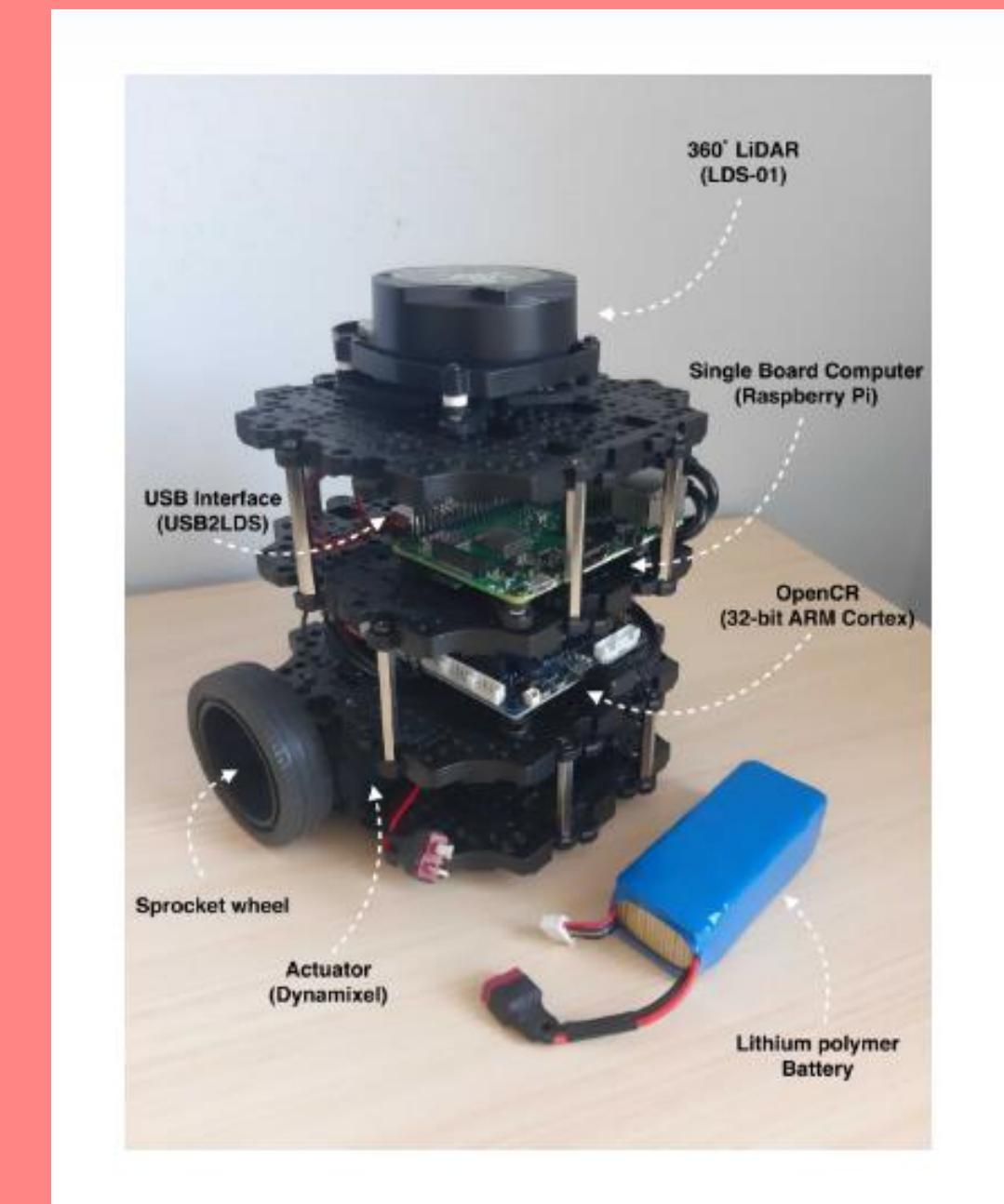
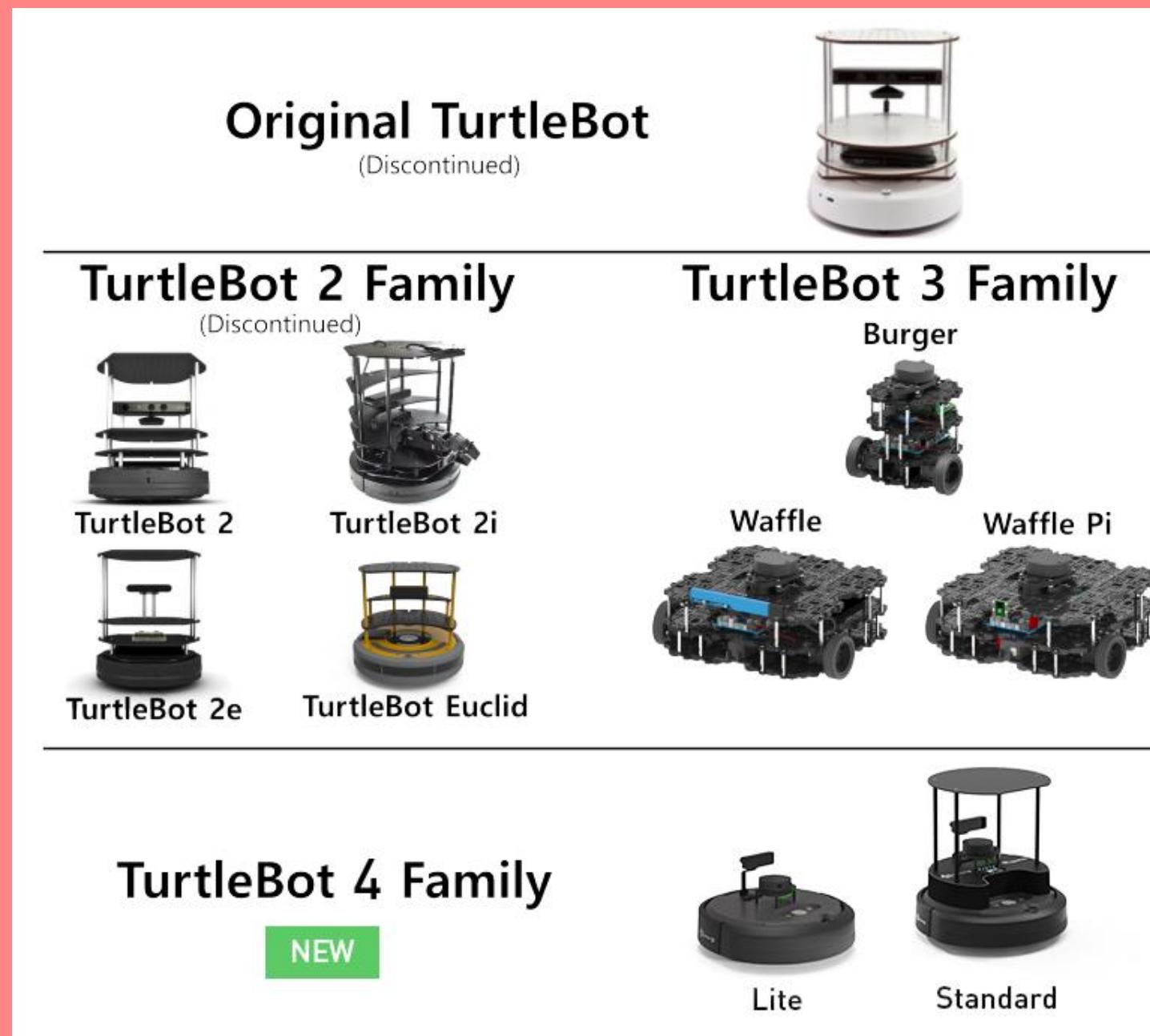
Công nghệ cốt lõi của TurtleBot là SLAM và ACML, giúp nó phù hợp với các robot dịch vụ gia đình. TurtleBot có thể chạy các thuật toán SLAM để xây dựng bản đồ và di chuyển xung quanh phòng. Ngoài ra, nó có thể được điều khiển từ xa

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



# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

## Cài đặt Gói Turtlebot



# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

## Cài đặt Gói Turtlebot

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

# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

## 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](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”

# CHƯƠNG 2. CÀI ĐẶT MÔI TRƯỜNG

## 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.

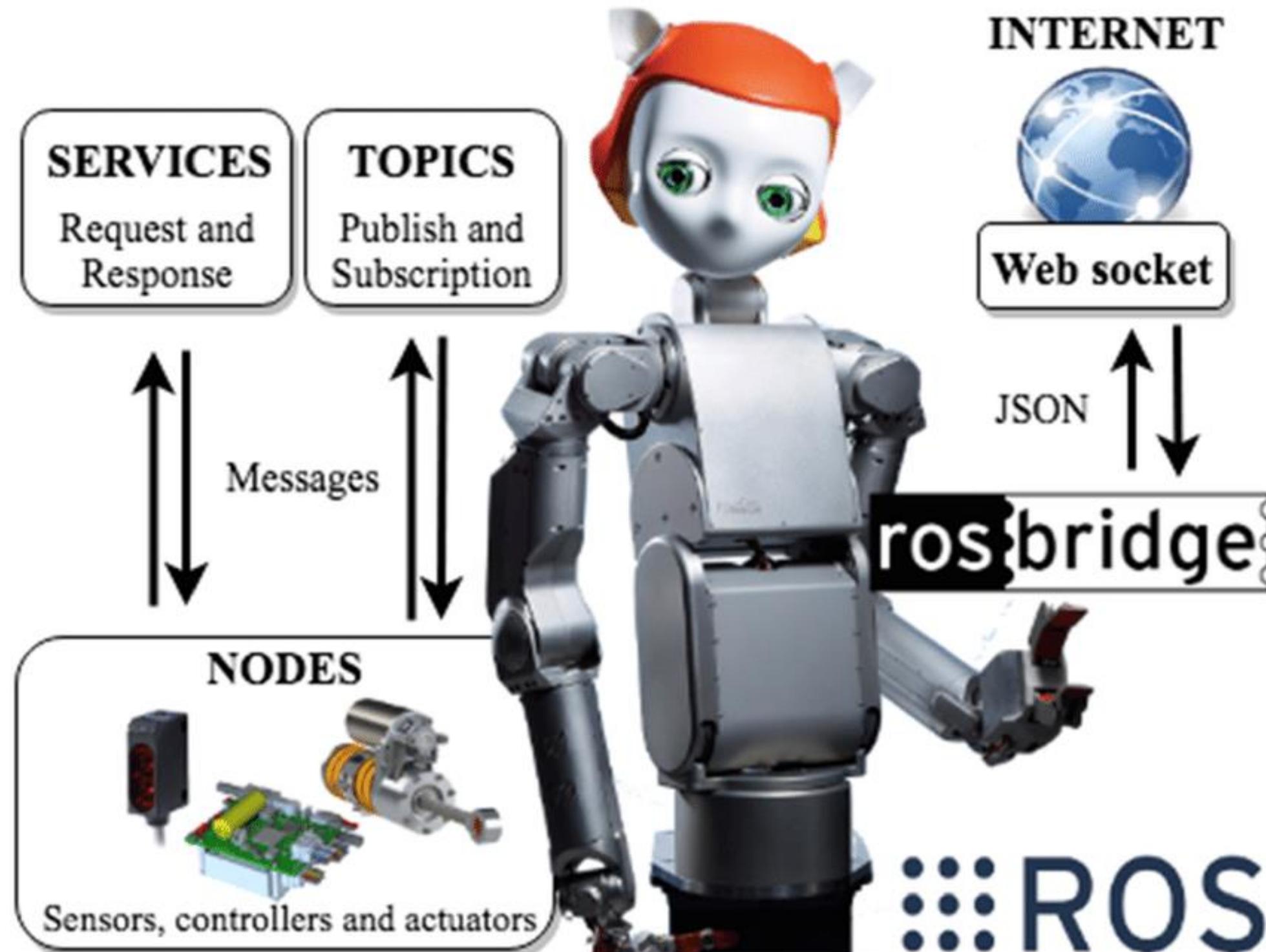
Tiếp đến tạo hai chương trình để thực hiện mô phỏng:

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.



# CẤU TRÚC CỦA ROS



ROS

# CHƯƠNG 3. CẤU TRÚC CỦA ROS

## Node

Trong ROS một Node là một quá trình thực hiện một tác vụ cụ thể.

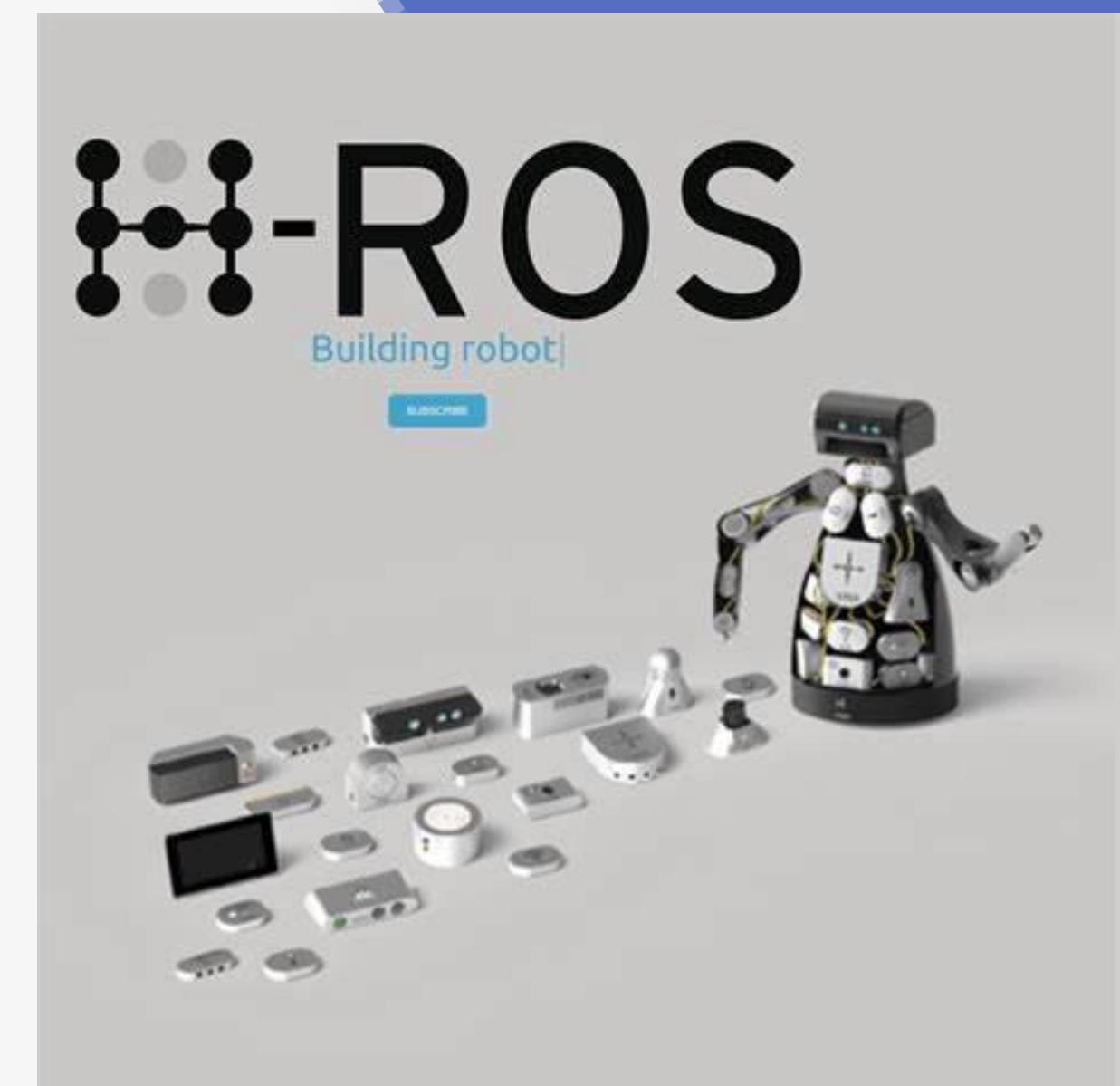
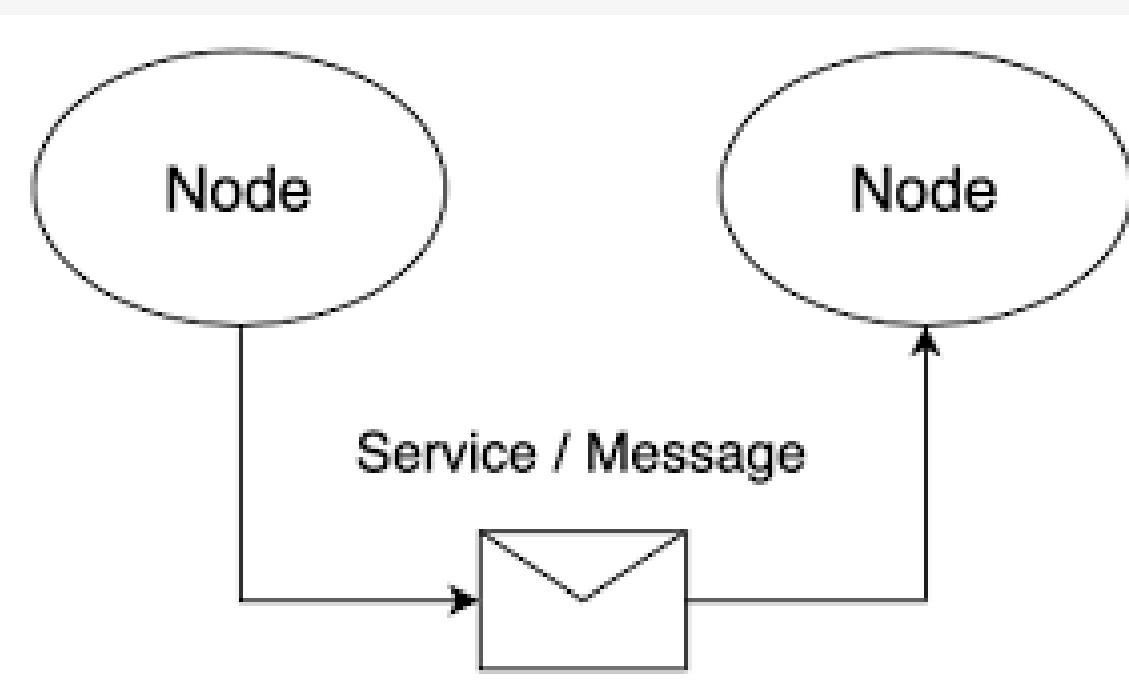
Các Node có

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

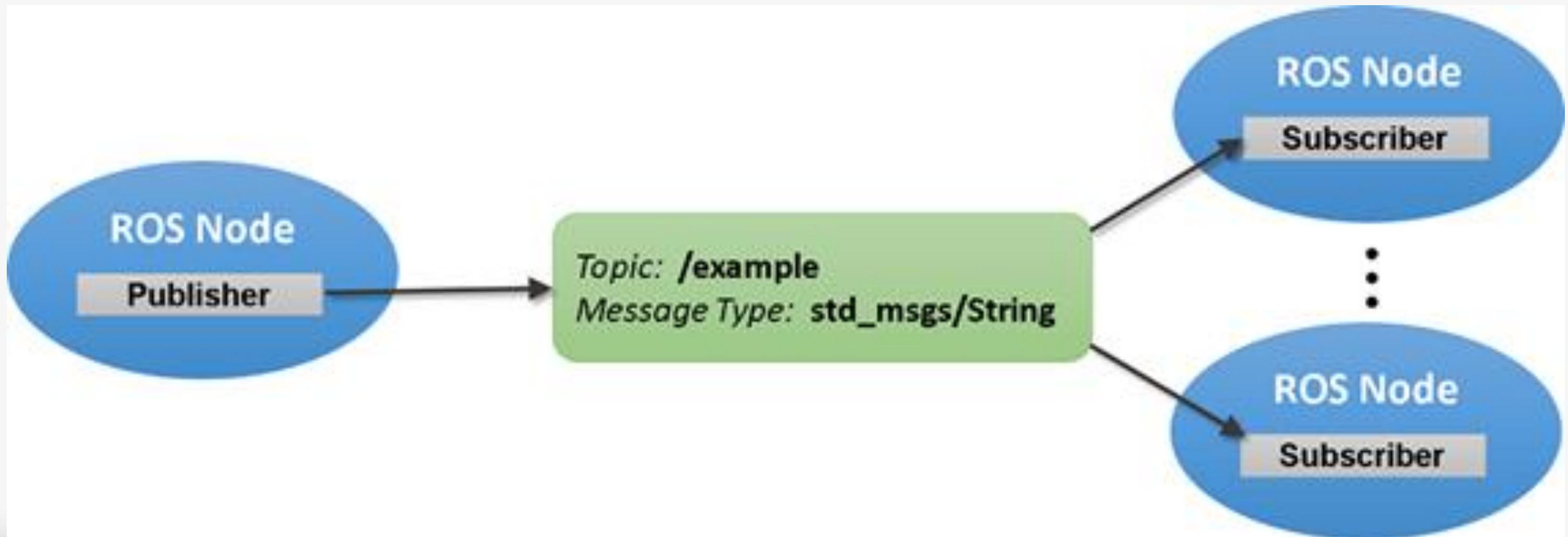
vận tốc đến thiết bị điều khiển động cơ. Các nodes này có thể giao  
tiếp và tương tác  
với nhau qua Master.



# CHƯƠNG 3. CẤU TRÚC CỦA ROS

## Topic

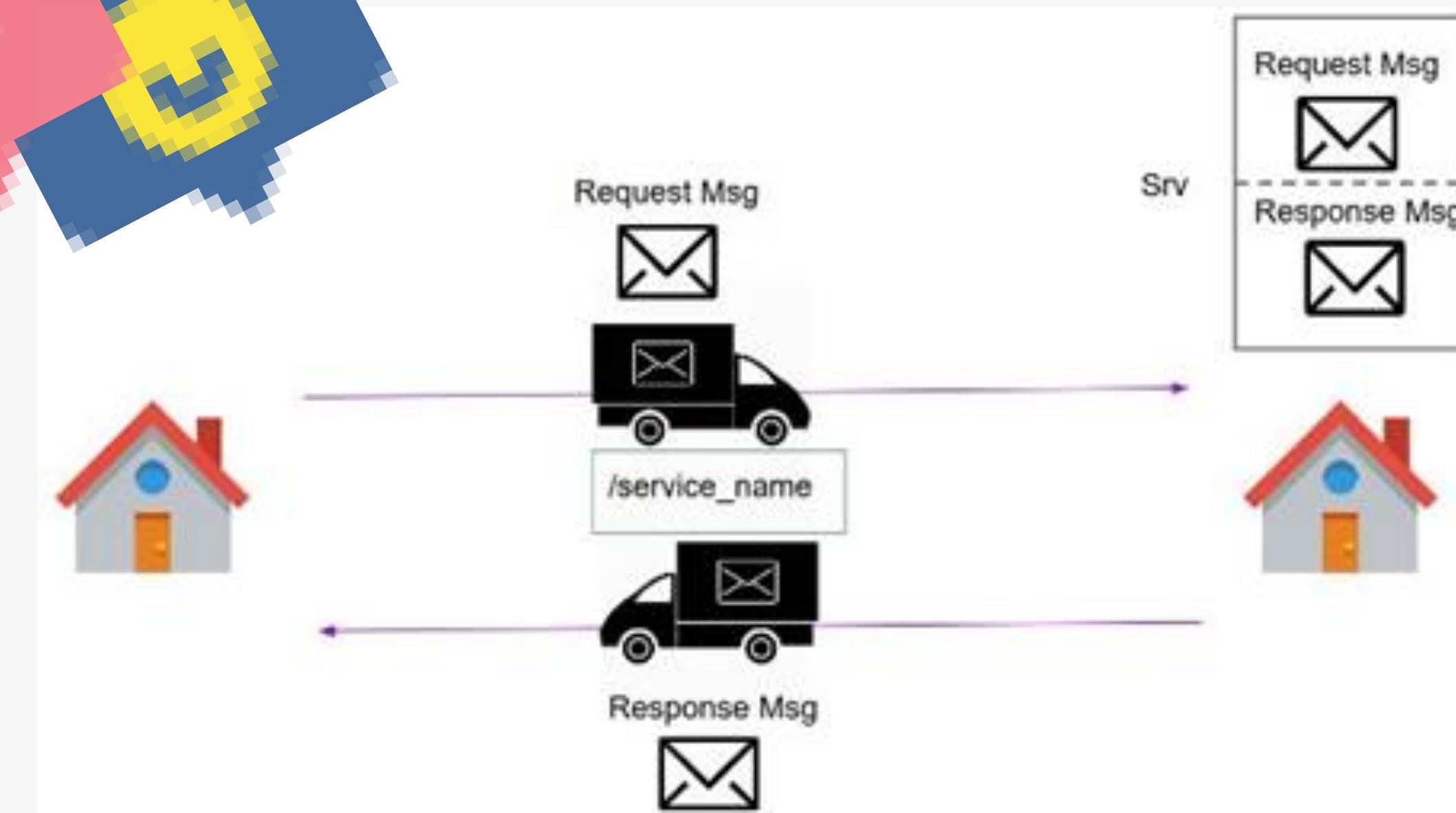
Trong ROS, Topic là một hệ thống truyền tải dữ liệu dựa trên một hệ thống publish và subscribe. Một hoặc nhiều node có thể publish (truyền) dữ liệu cho một topic và một hoặc nhiều node khác có thể subscribe (nhận) để nhận dữ liệu từ topic đó



# CHƯƠNG 3. CẤU TRÚC CỦA ROS

Message

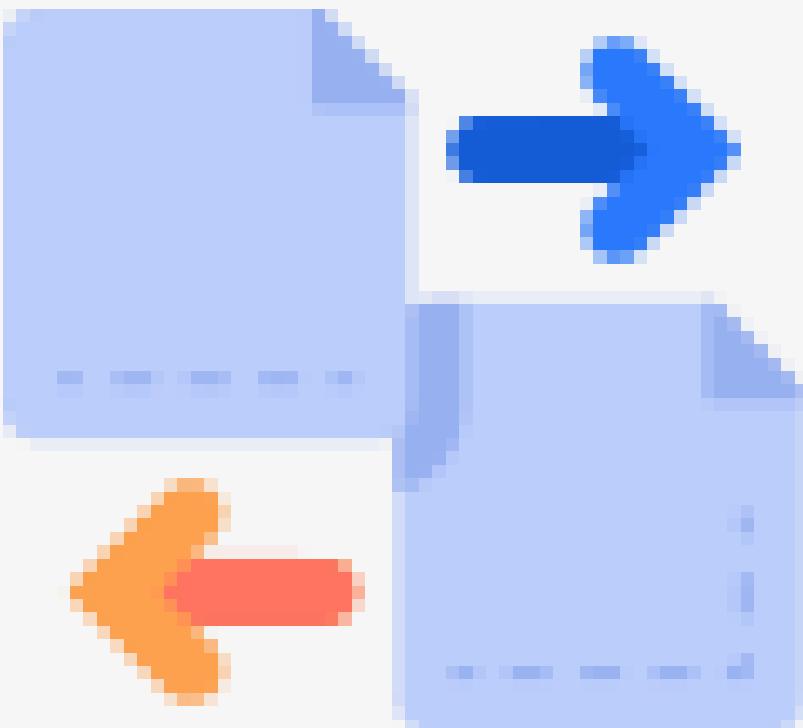
Message là một cấu trúc dữ liệu mà các node sử dụng để trao đổi với nhau thông qua topic. Message bao gồm sự kết hợp của nhiều kiểu dữ liệu. Các node tương tác với nhau bằng cách send và receive ROS message.



# CHƯƠNG 3. CẤU TRÚC CỦA ROS

## 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.



## 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.

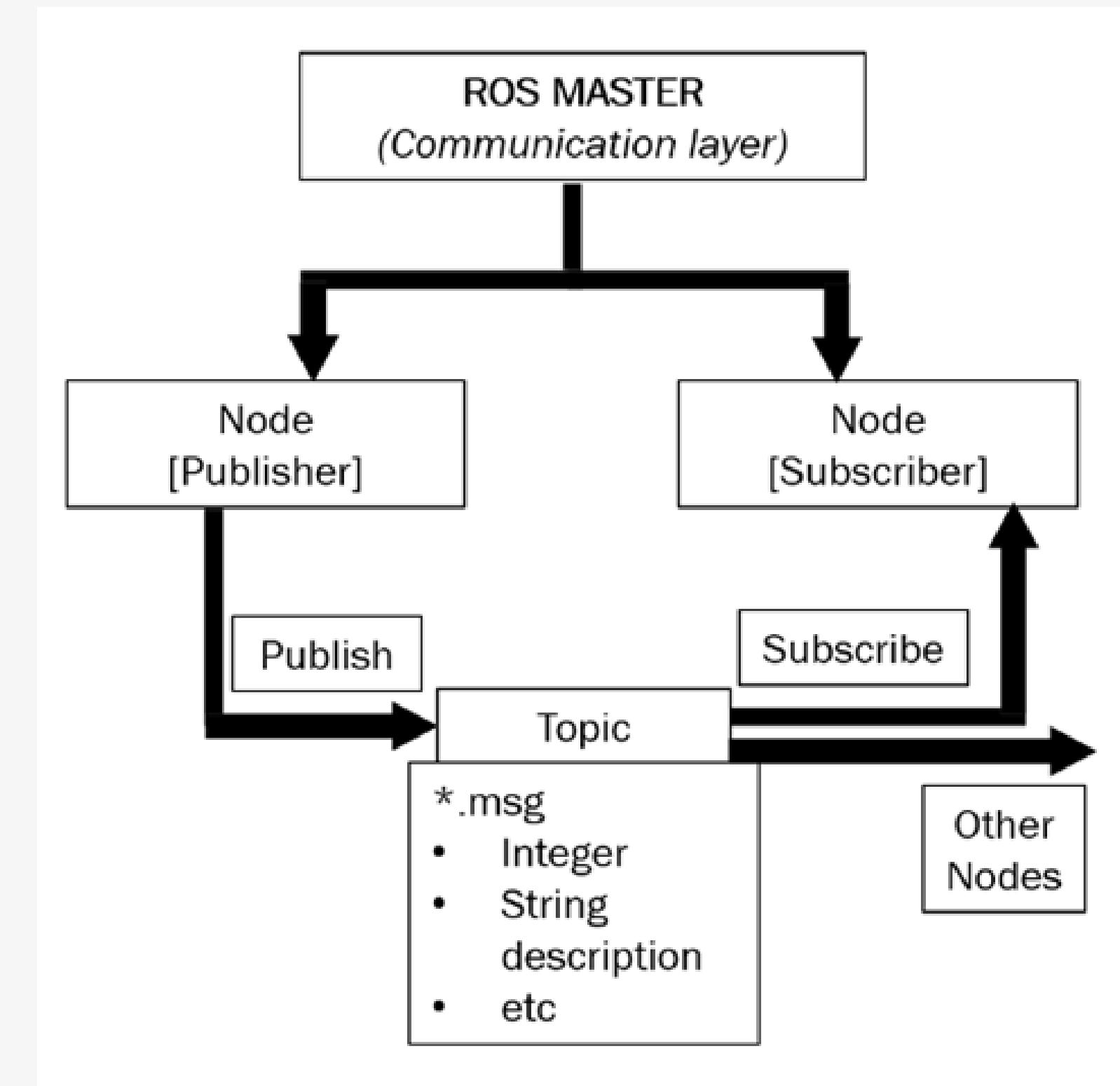
# CHƯƠNG 3. CẤU TRÚC CỦA ROS

## 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`.

# CHƯƠNG 3. CẤU TRÚC CỦA ROS

## Cấu trúc hoàn chỉnh của ROS



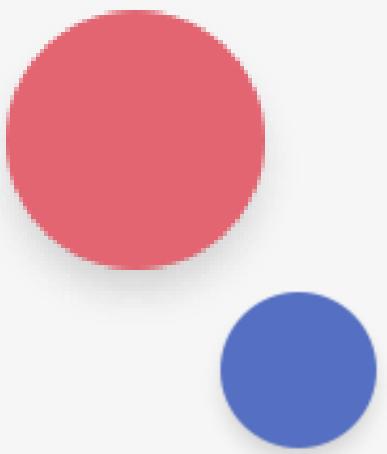
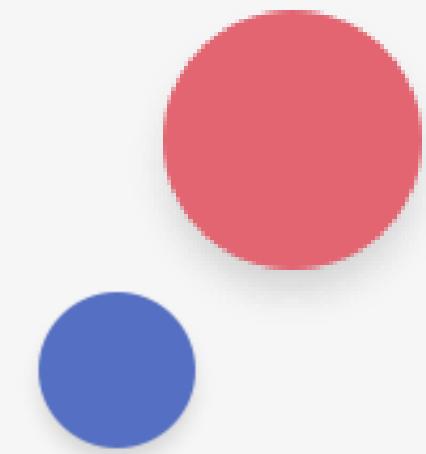


# XÂY DỰNG MÔI TRƯỜNG MÔ PHÒNG TRONG GAZEBO



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

### Xây dựng tường



Đầu tiên em sẽ xây dựng tường bao bọc xung quanh để tạo không gian di chuyển của TurtleBot, 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:

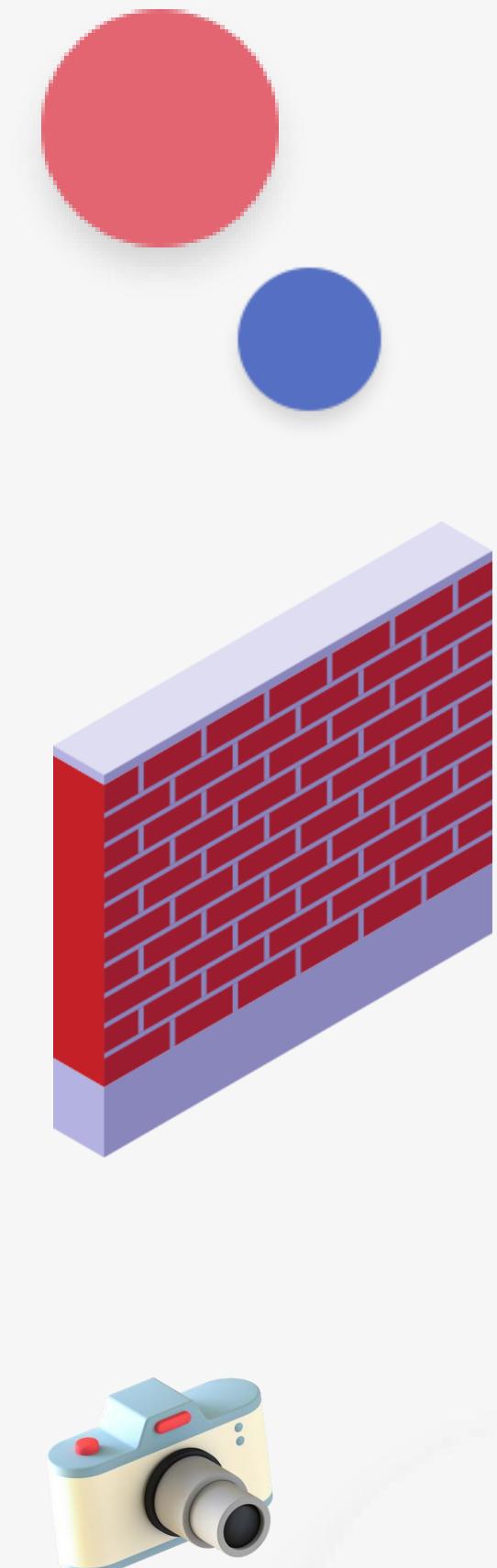
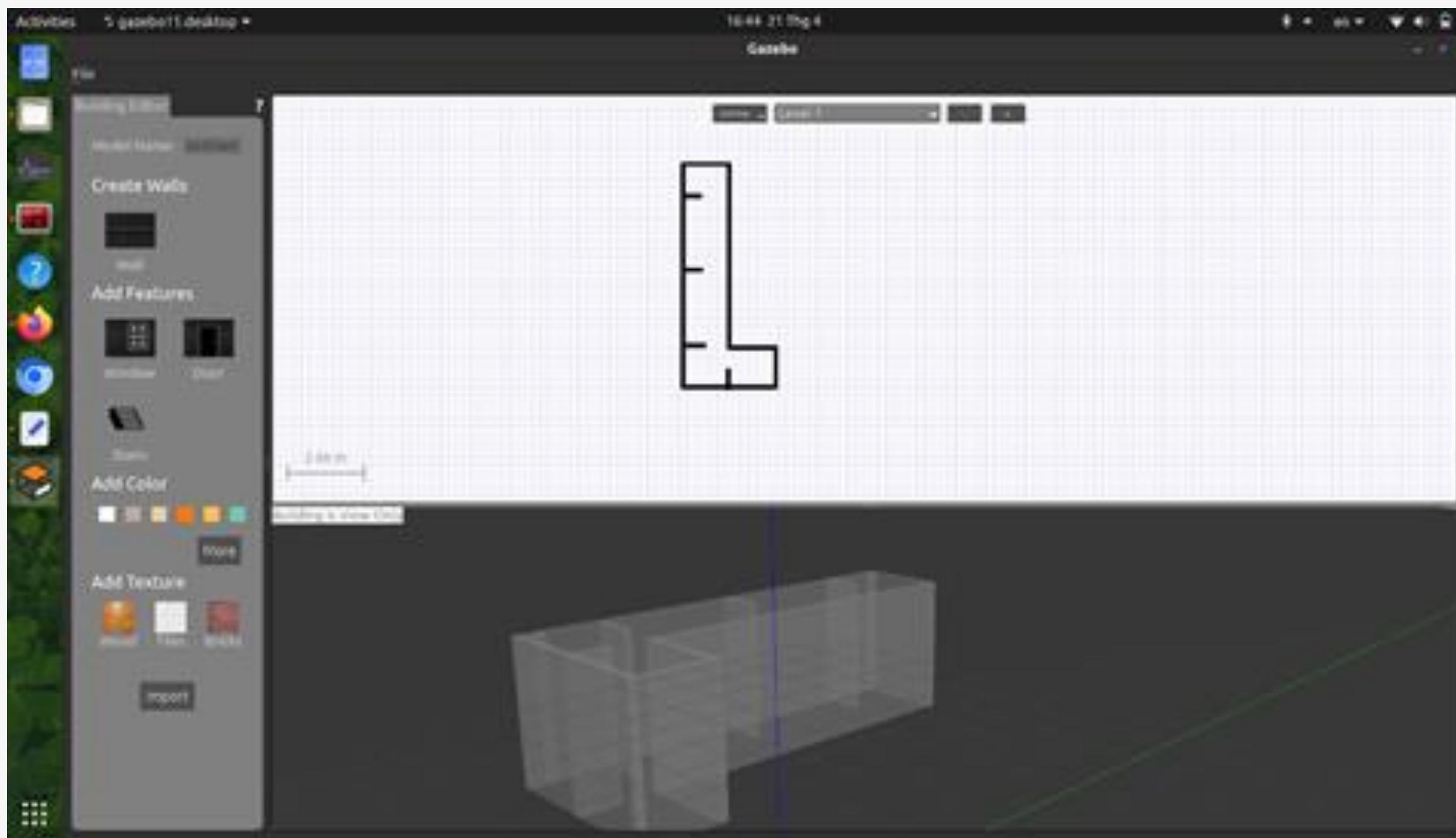


■  `model.config`: Đây là tệp cấu hình chứa thông tin meta về mô hình. Tệp này cung cấp cho Gazebo các thông tin cần thiết để nạp mô hình vào môi trường mô phỏng.

■ `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

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

Xây dựng tường



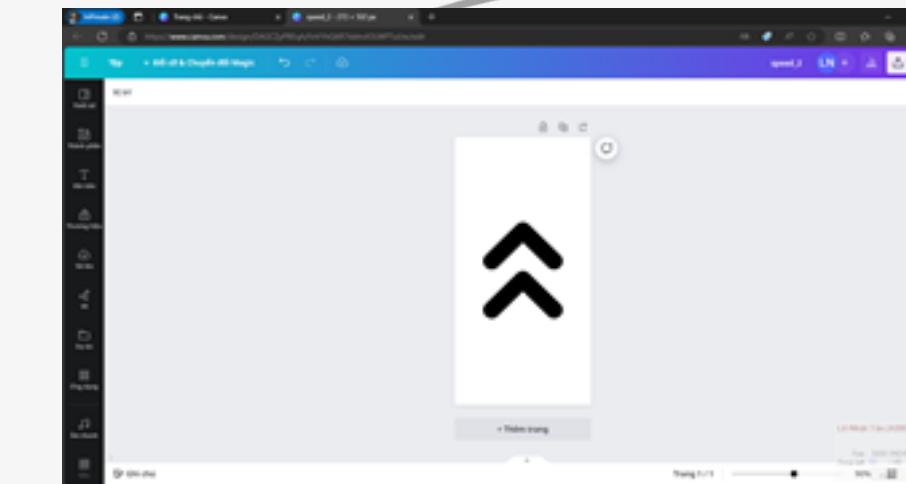
# CHƯƠNG 4. XÂY DỰNG MÔI TRƯỜNG MÔ PHÒNG TRONG GAZEBO

## Xây dựng các biển báo tốc độ

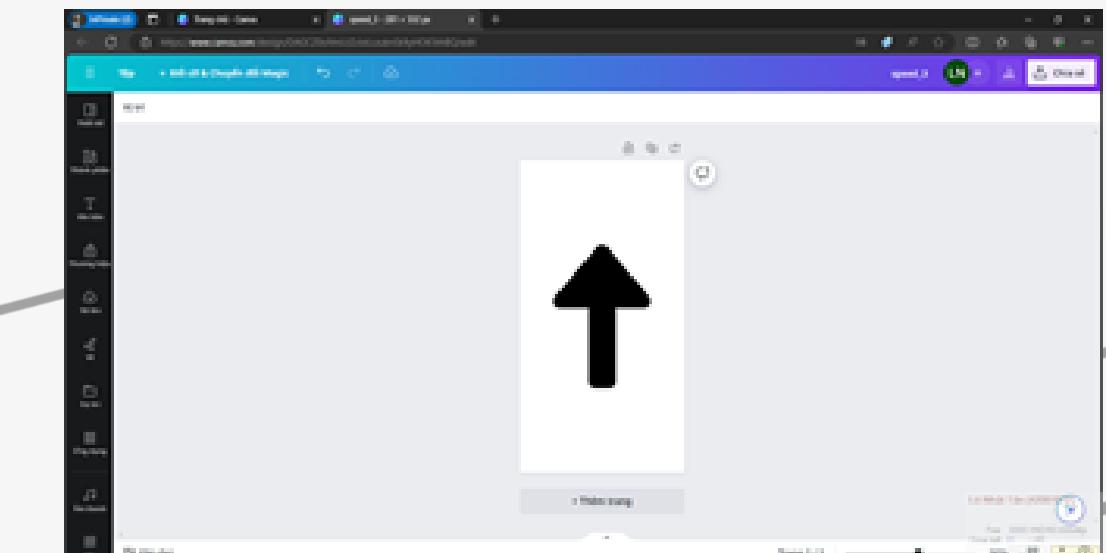
Biển báo speed\_1 đại diện cho tốc độ di chuyển chậm: 0.15 m/s.



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



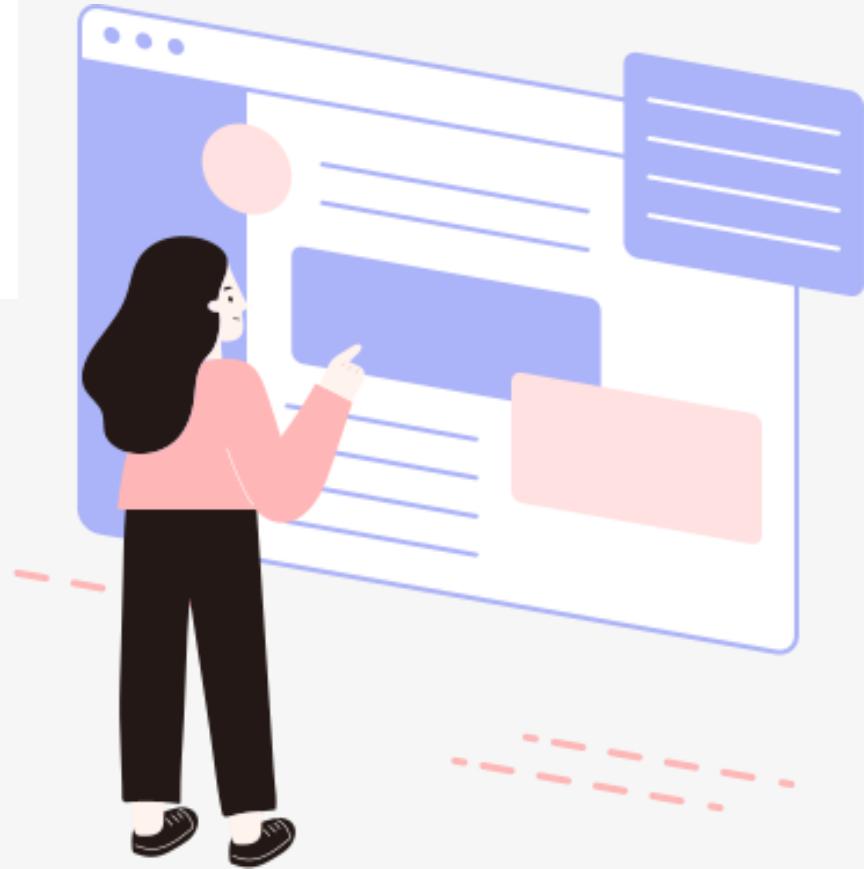
Biển báo speed\_3 đại diện cho tốc độ di chuyển nhanh: 0.25 m/s.



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

## 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`

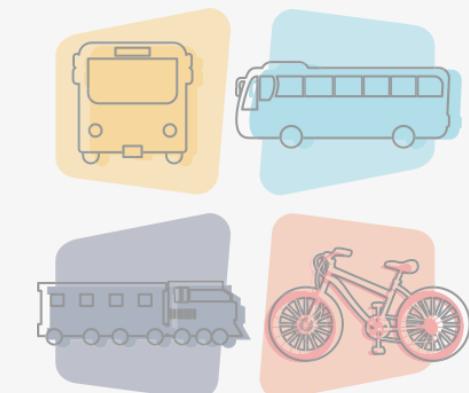
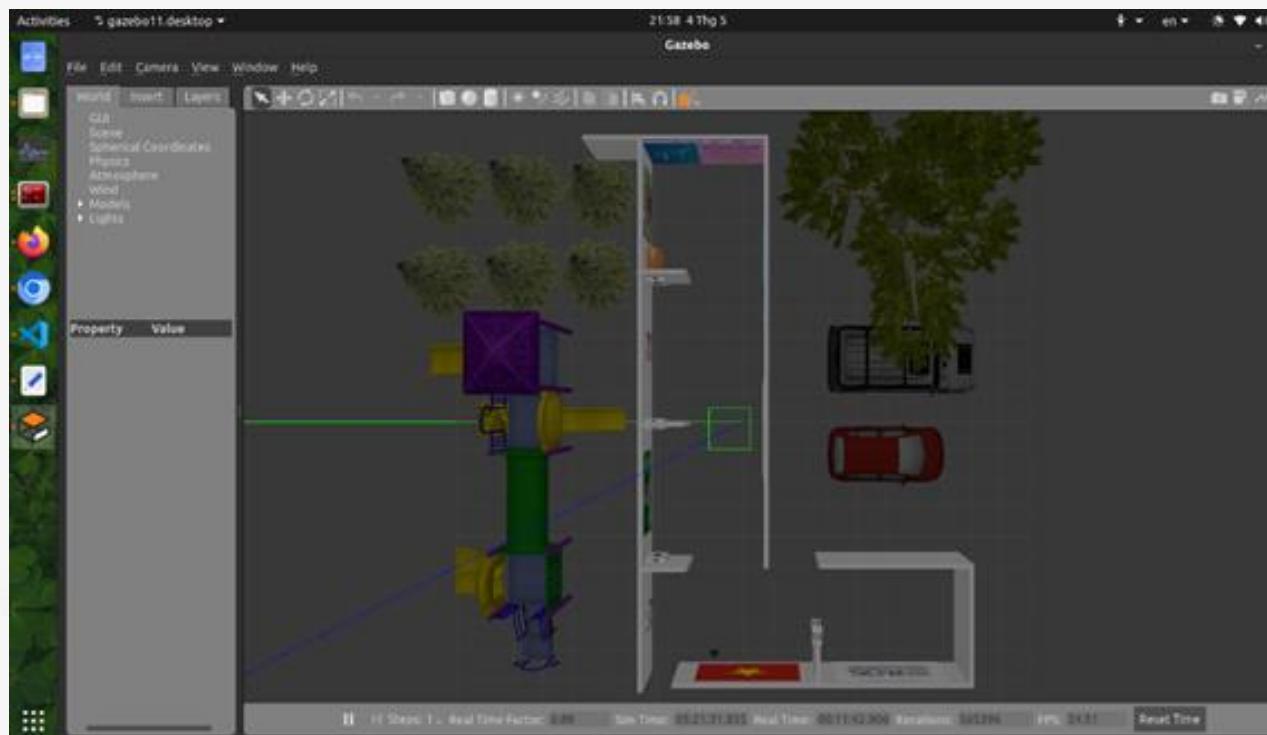


# CHƯƠNG 4. XÂY DỰNG MÔI TRƯỜNG MÔ PHÒNG TRONG GAZEBO



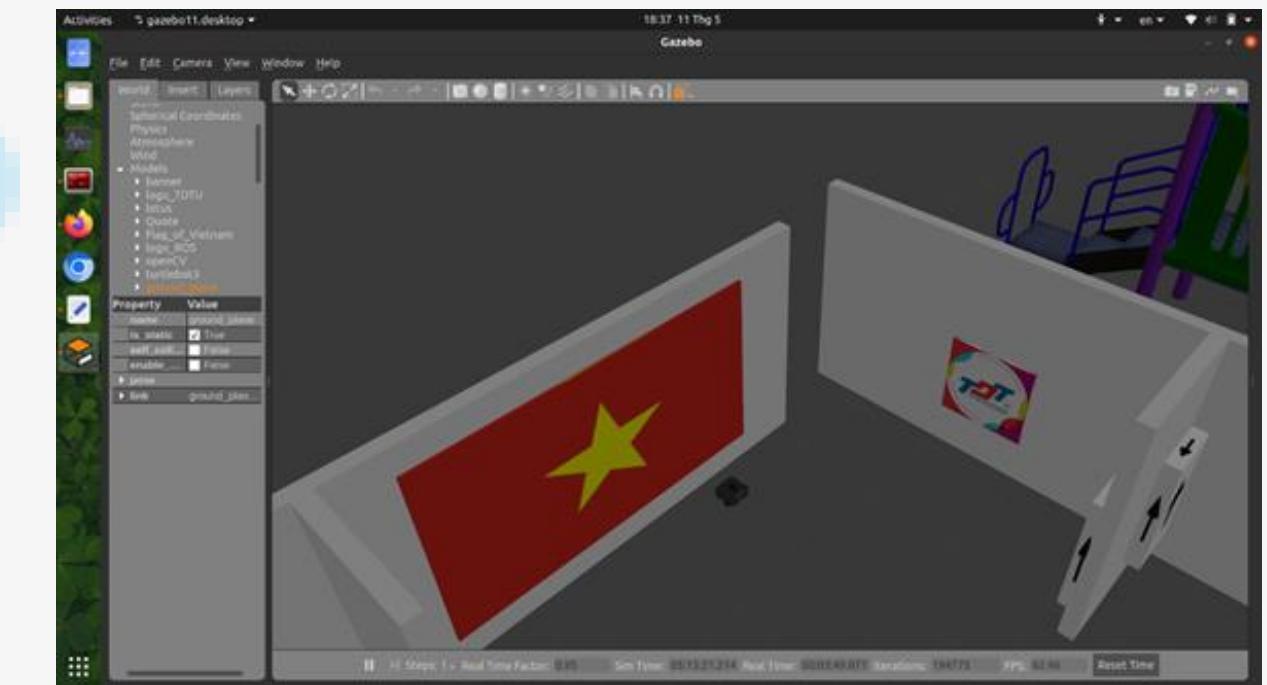
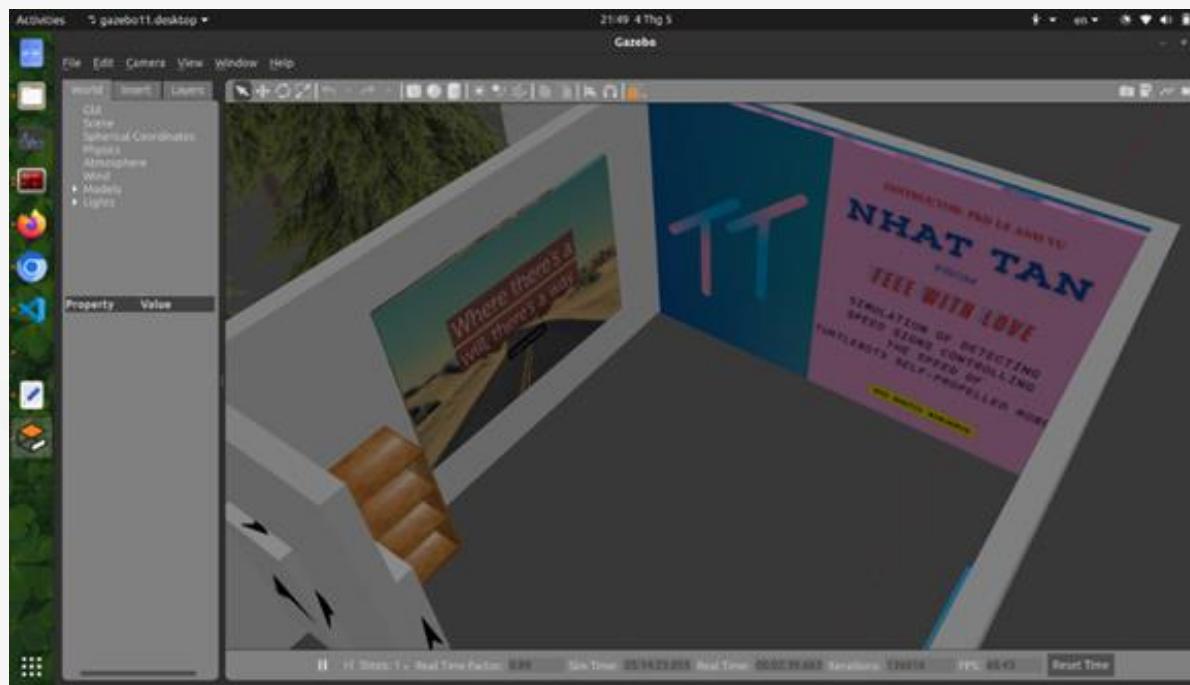
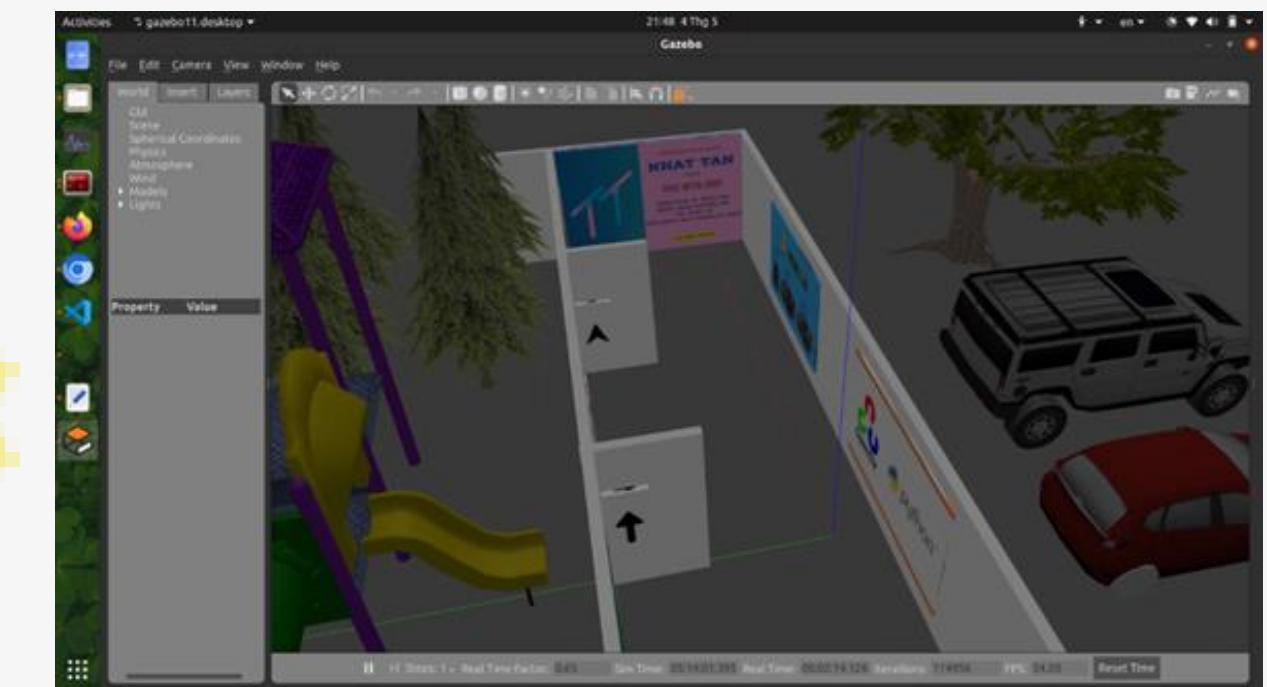
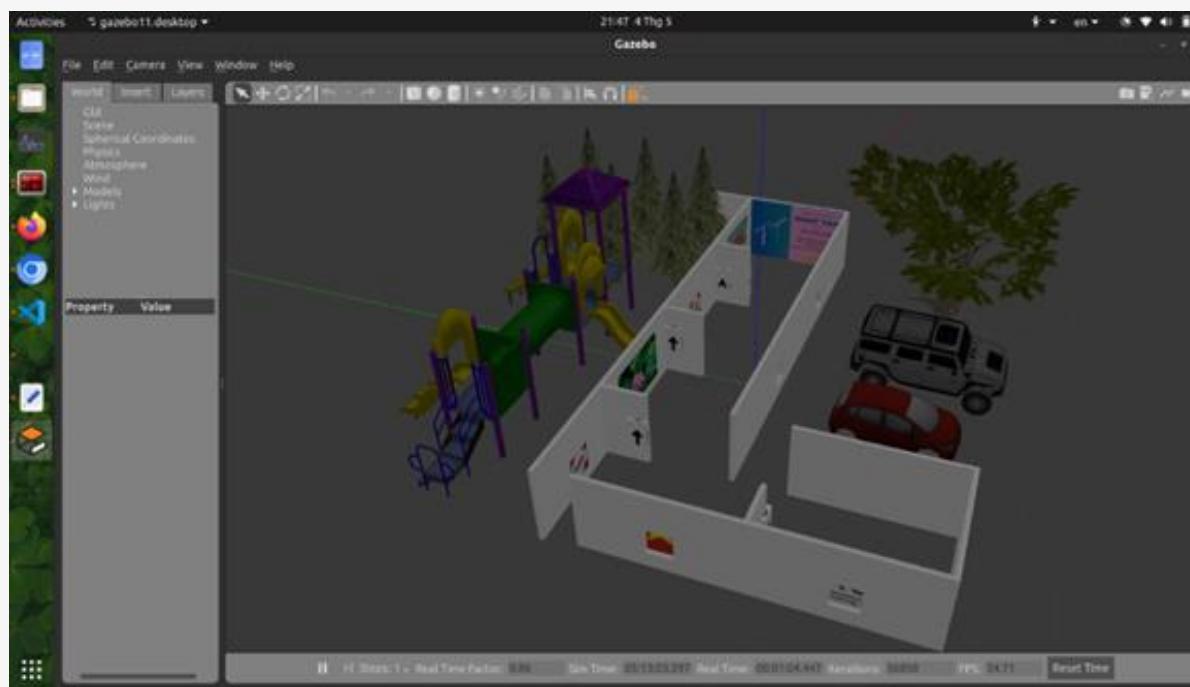
## Xây dựng môi trường mô phỏng

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`



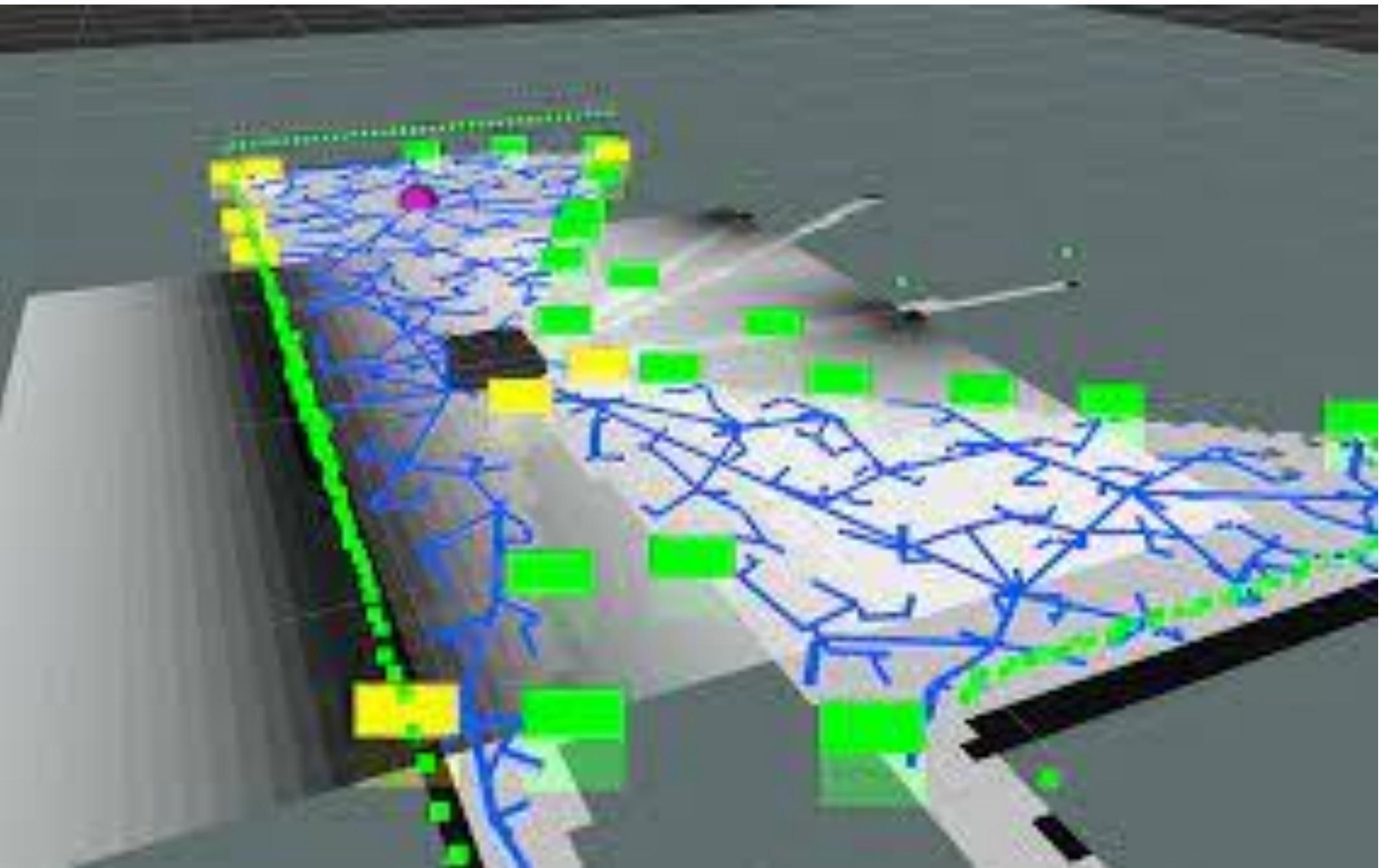
# CHƯƠNG 4. XÂY DỰNG MÔI TRƯỜNG MÔ PHÒNG TRONG GAZEBO

## Xây dựng môi trường mô phỏng

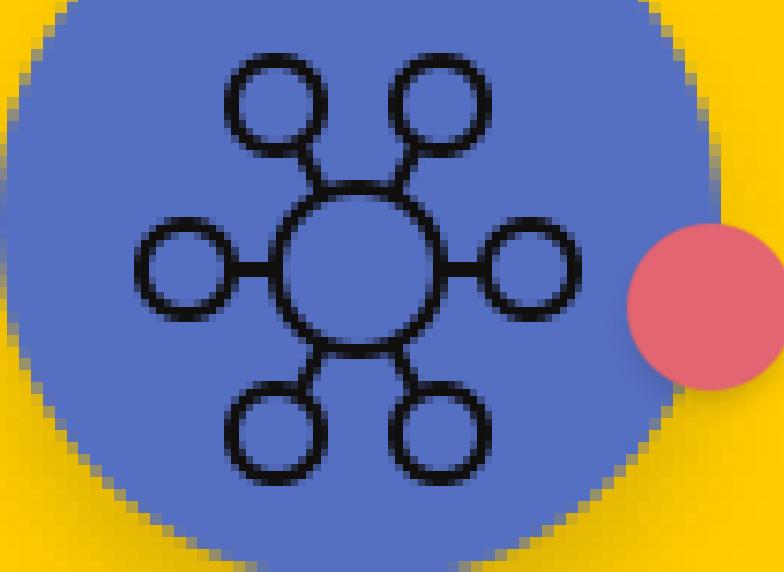


5

# XÂY DỰNG BẢN ĐỒ ĐỊNH TRƯỚC SLAM



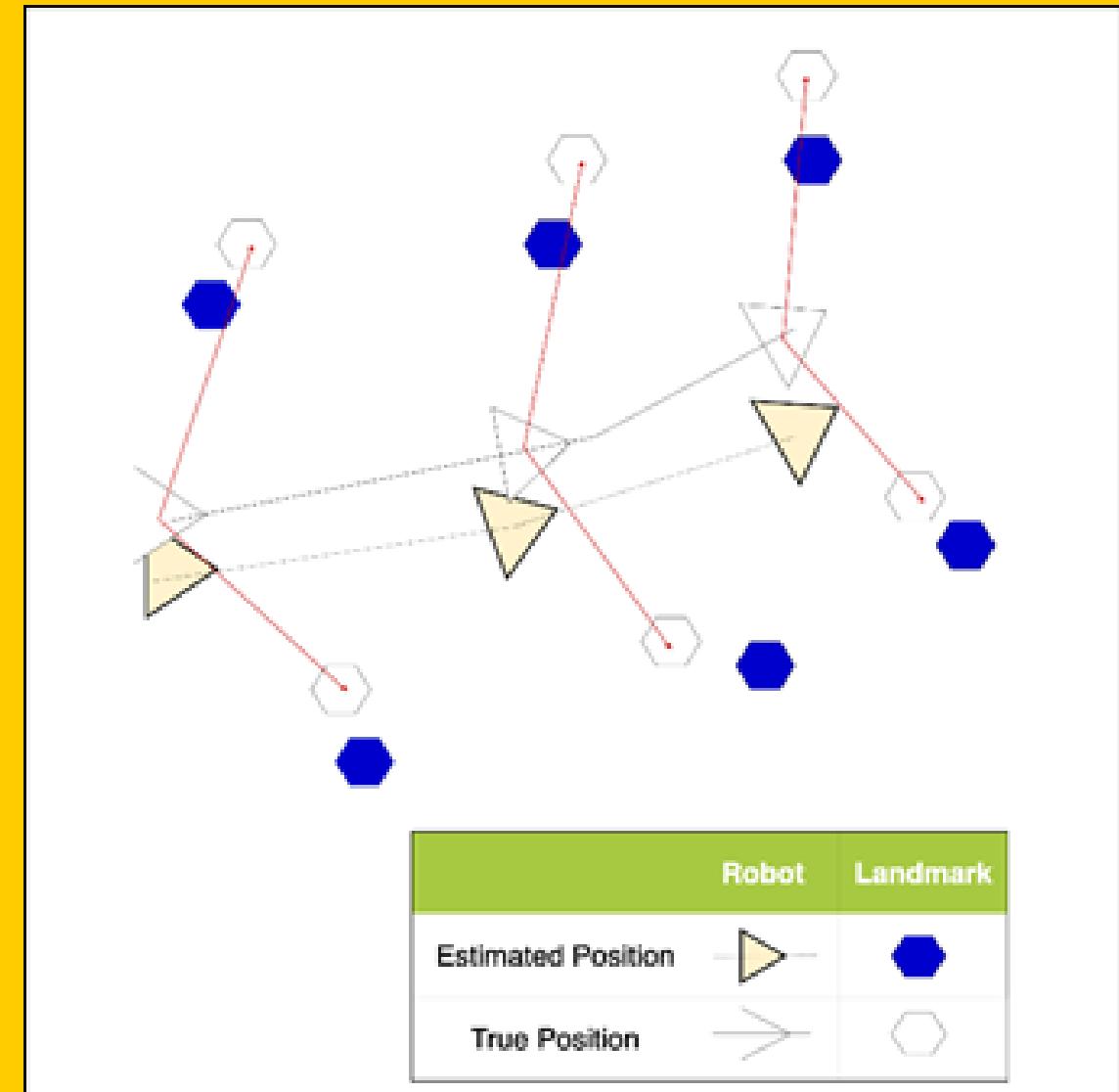
# CHƯƠNG 5. XÂY DỰNG BẢN ĐỒ ĐỊNH TRƯỚC SLAM



## 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



# CHƯƠNG 5. XÂY DỰNG BẢN ĐỒ ĐỊNH TRƯỚC SLAM



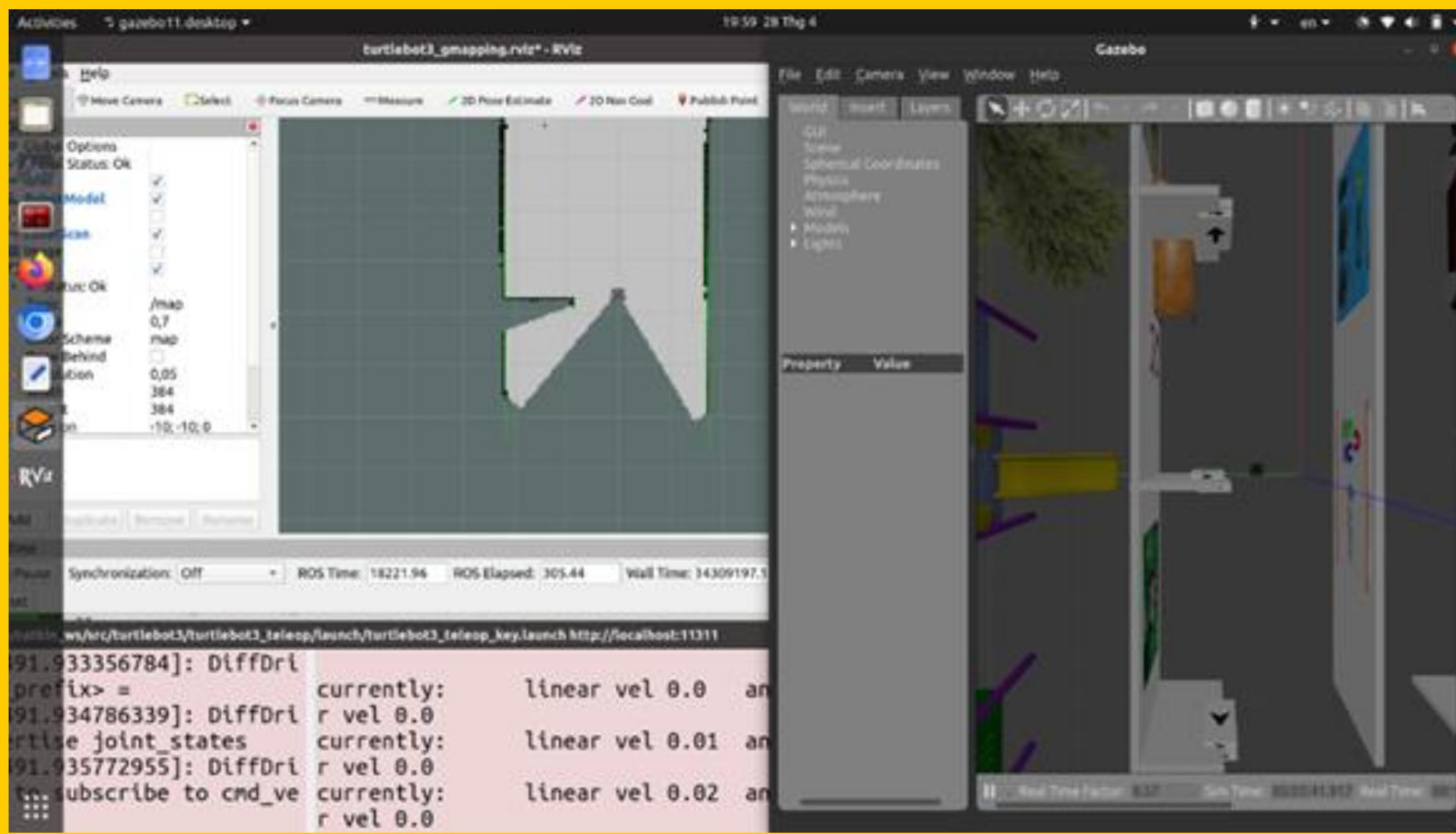
## Thực hiện SLAM

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 già 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**

# CHƯƠNG 5. XÂY DỰNG BẢN ĐỒ ĐỊNH TRƯỚC SLAM

# Thực hiện SLAM

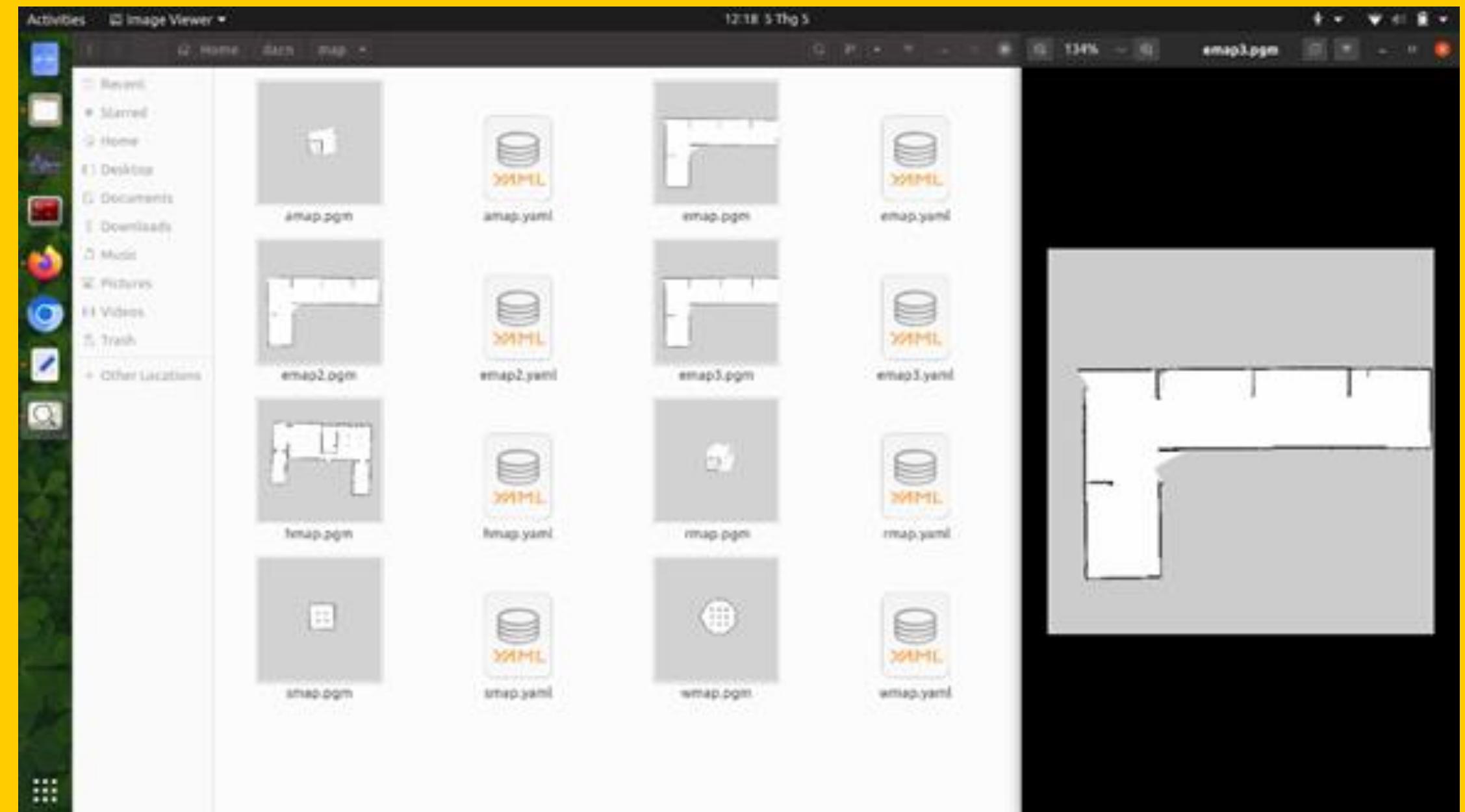


# CHƯƠNG 5. XÂY DỰNG BẢN ĐỒ ĐỊNH TRƯỚC SLAM

# SAVE

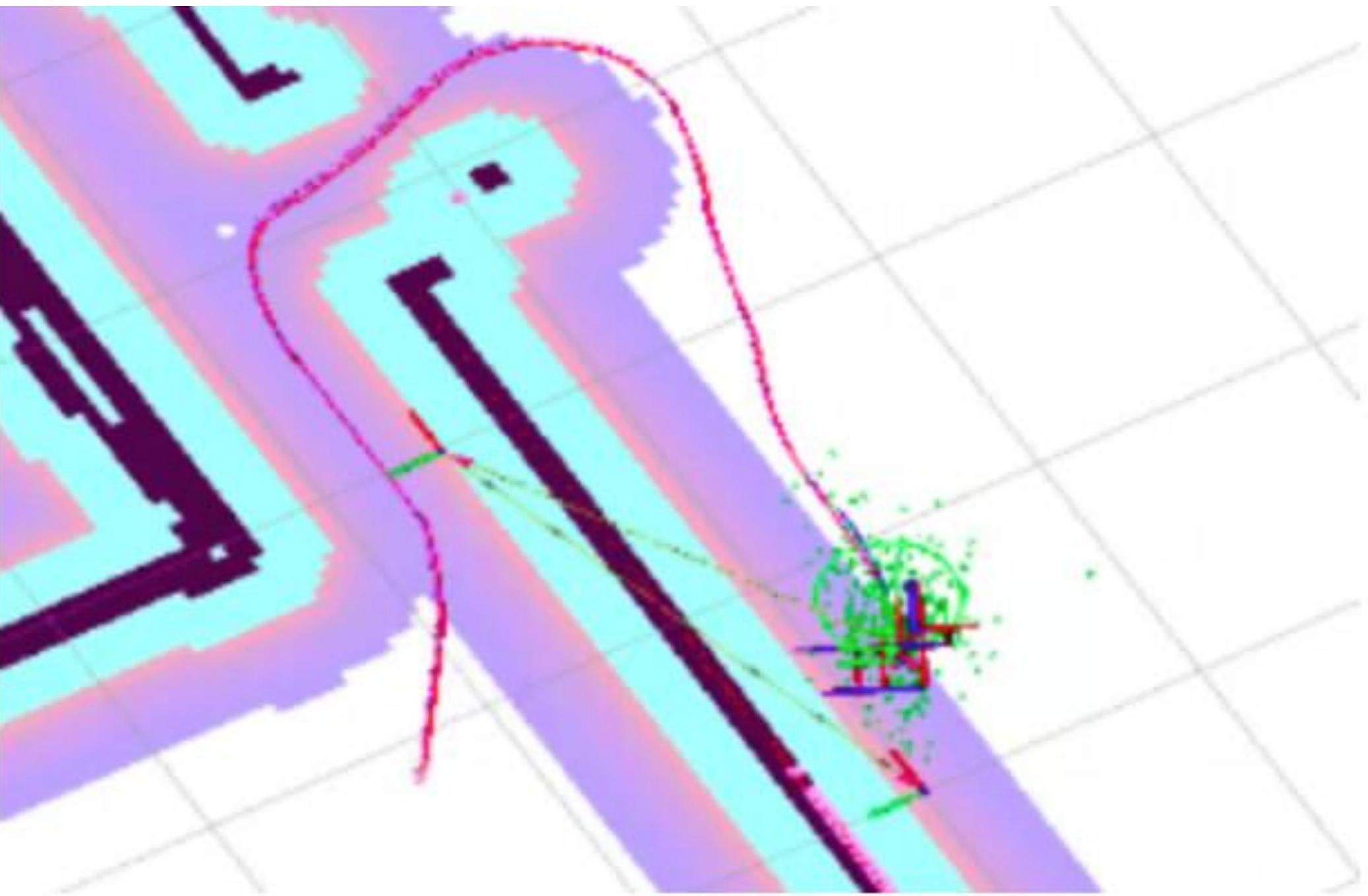
Khi lưu xong, sẽ thấy có 2 file đó là .pgm (lưu trữ bản đồ dưới dạng hình ảnh bitmap) và .yaml (lưu trữ bản đồ dưới dạng dữ liệu cấu trúc để lưu trữ thông tin chi tiết về bản đồ

## Thực hiện SLAM





# TỰ ĐỊNH VỊ BẢN THÂN ACML



# CHƯƠNG 6. TỰ ĐỊNH VỊ BẢN THÂN ACML

YOU ARE  
HERE

## 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)**
- **Cognitive (Nhận thức)**
- **Motivated (Động cơ)**

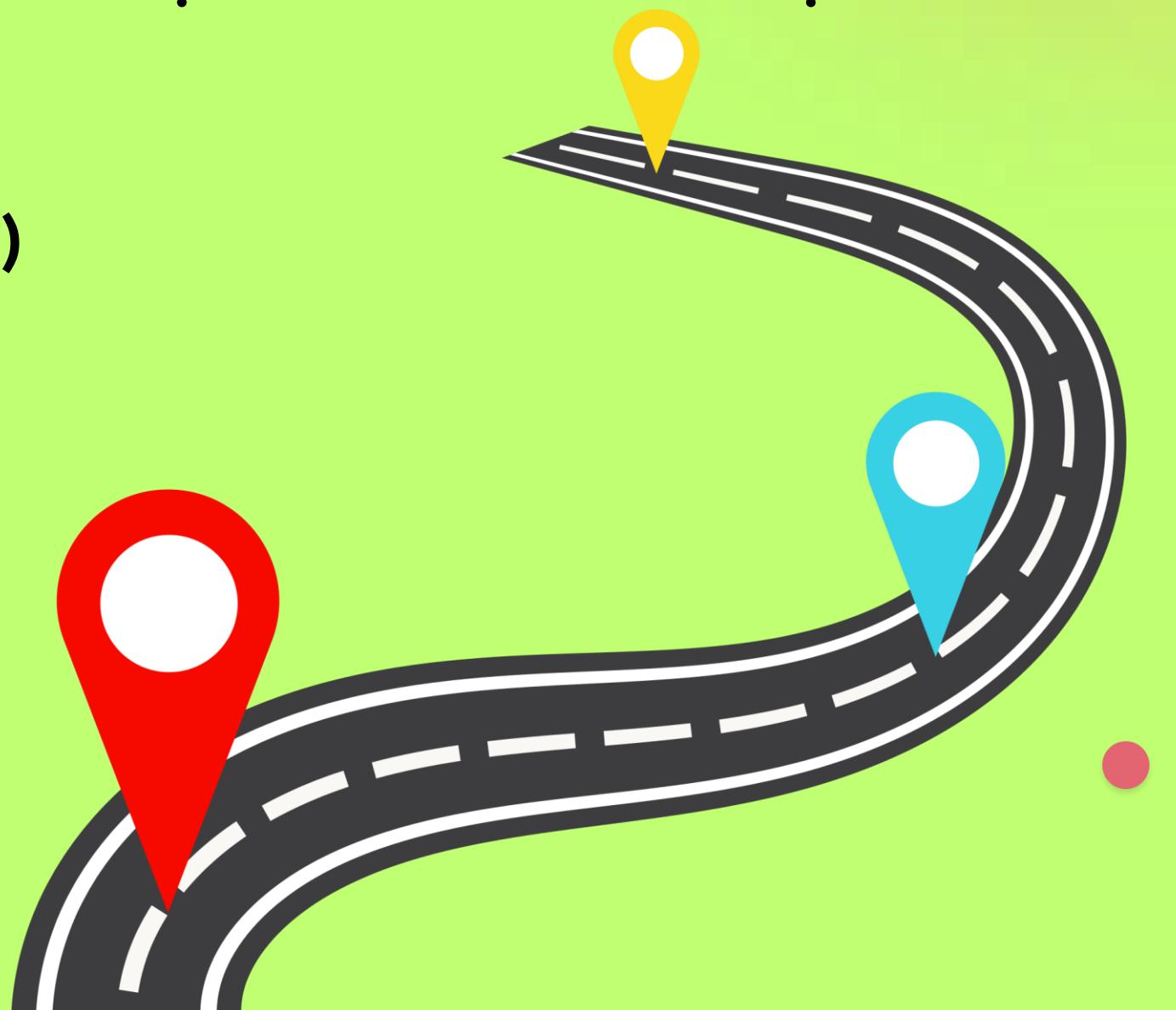


# CHƯƠNG 6. TỰ ĐỊNH VỊ BẢN THÂN ACML

## Các thuật toán tìm đường đi của Turtlebot3

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:

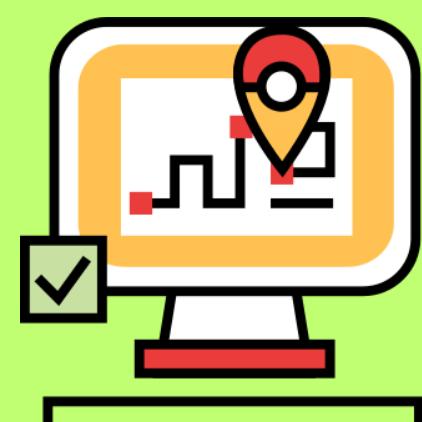
- **Lập kế hoạch đường đi toàn cục (Global Path Planning)**
- **Lập kế hoạch đường đi cục bộ (Local Path Planning)**



# CHƯƠNG 6. TỰ ĐỊNH VỊ BẢN THÂN ACML

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

**Các thuật toán tìm đường đi của Turtlebot3**

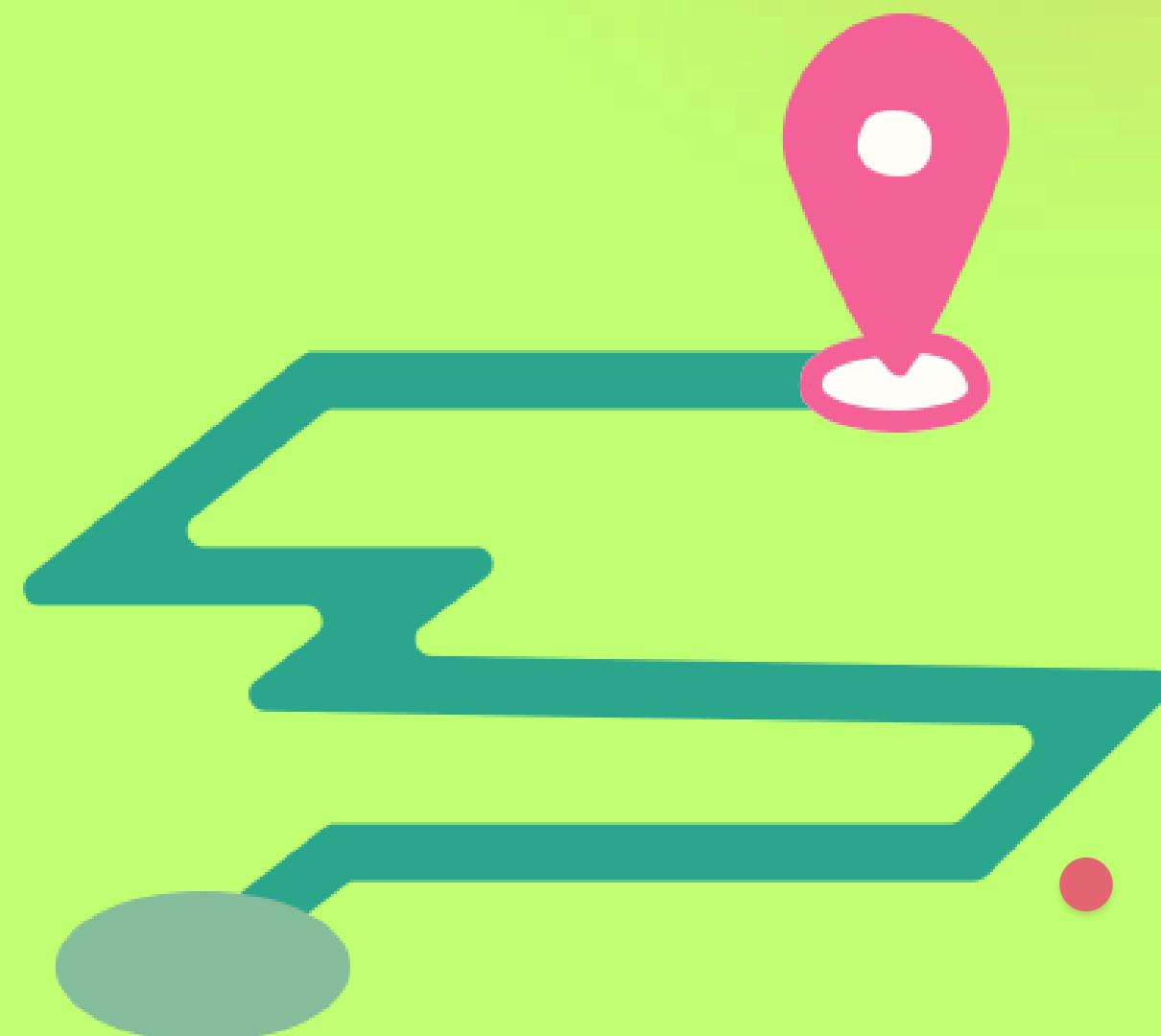


# CHƯƠNG 6. TỰ ĐỊNH VỊ BẢN THÂN ACML



## Các thuật toán tìm đường đi của Turtlebot3

Thuật toán	Độ phức tạp
DWA (Dynamic Window Approach)	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.
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 kê thì độ phức tạp của thuật toán sẽ bị giảm xuống còn $O[(V+E)\log V]$ .



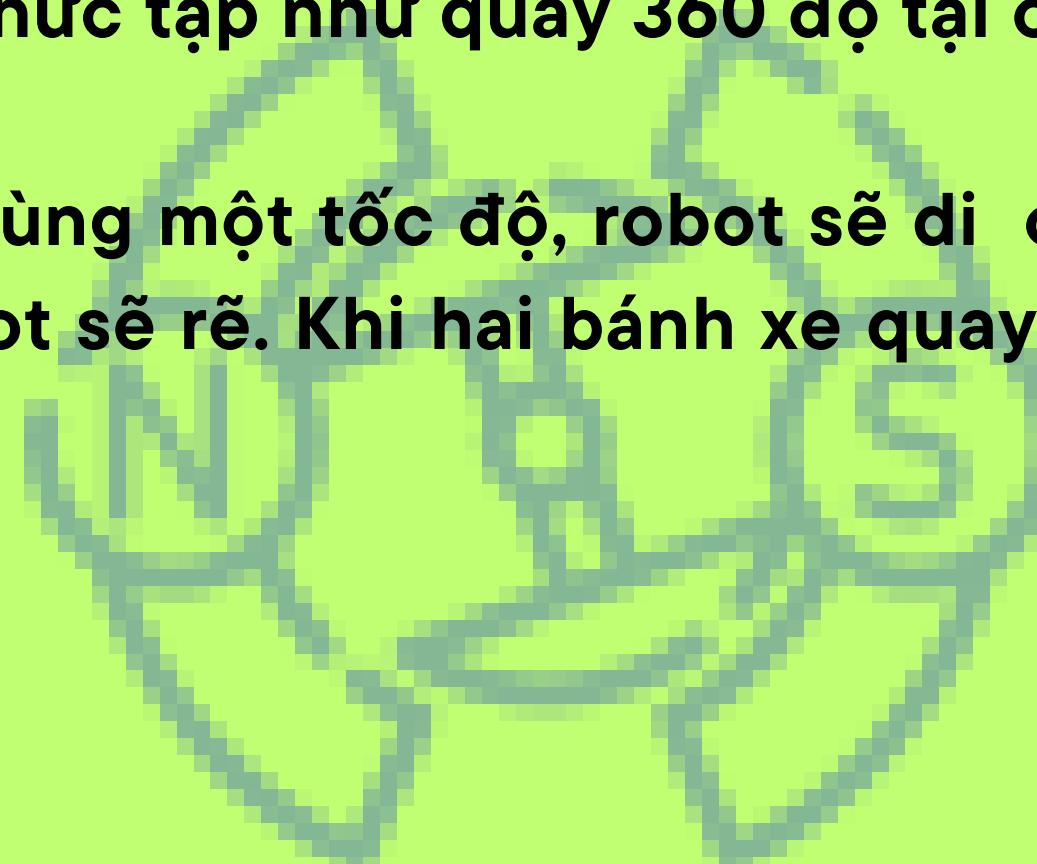
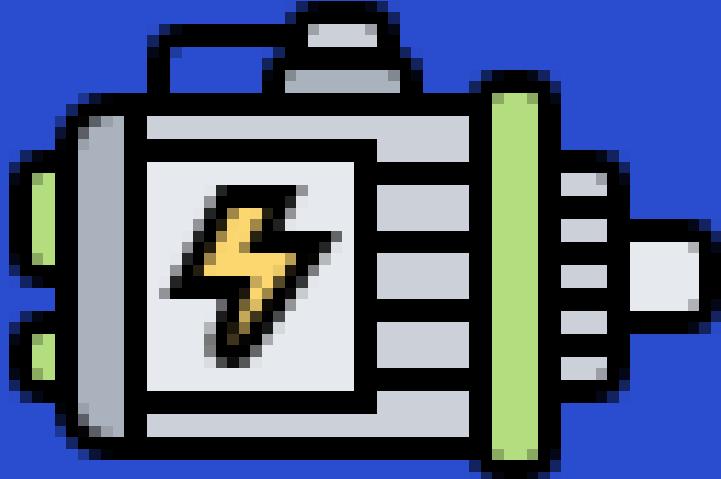
# CHƯƠNG 6. TỰ ĐỊNH VỊ BẢN THÂN ACML

## 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.

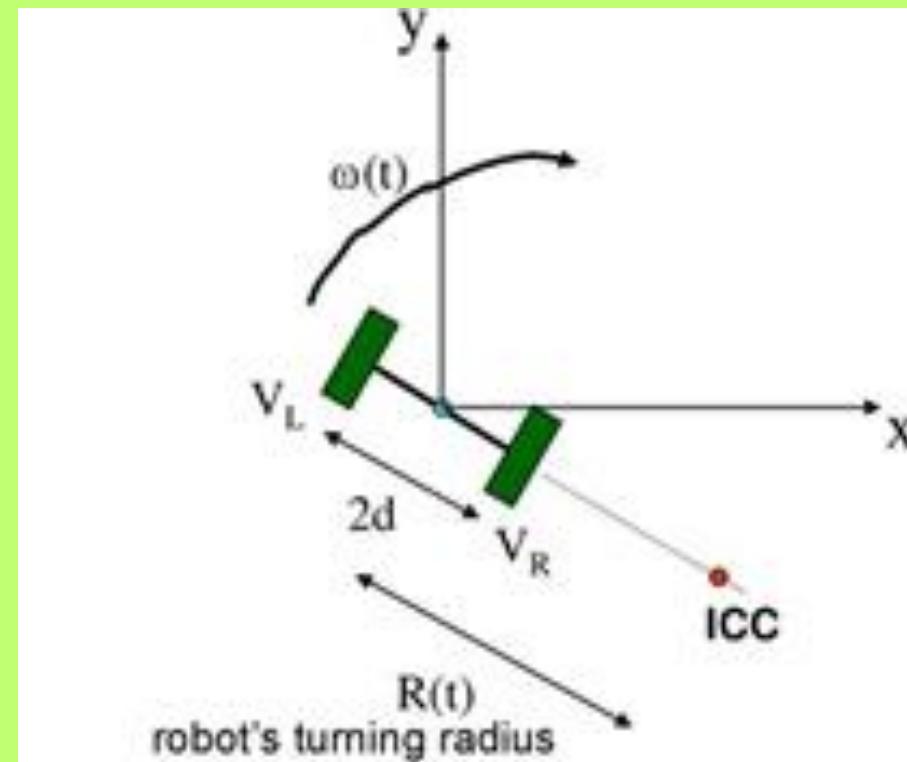
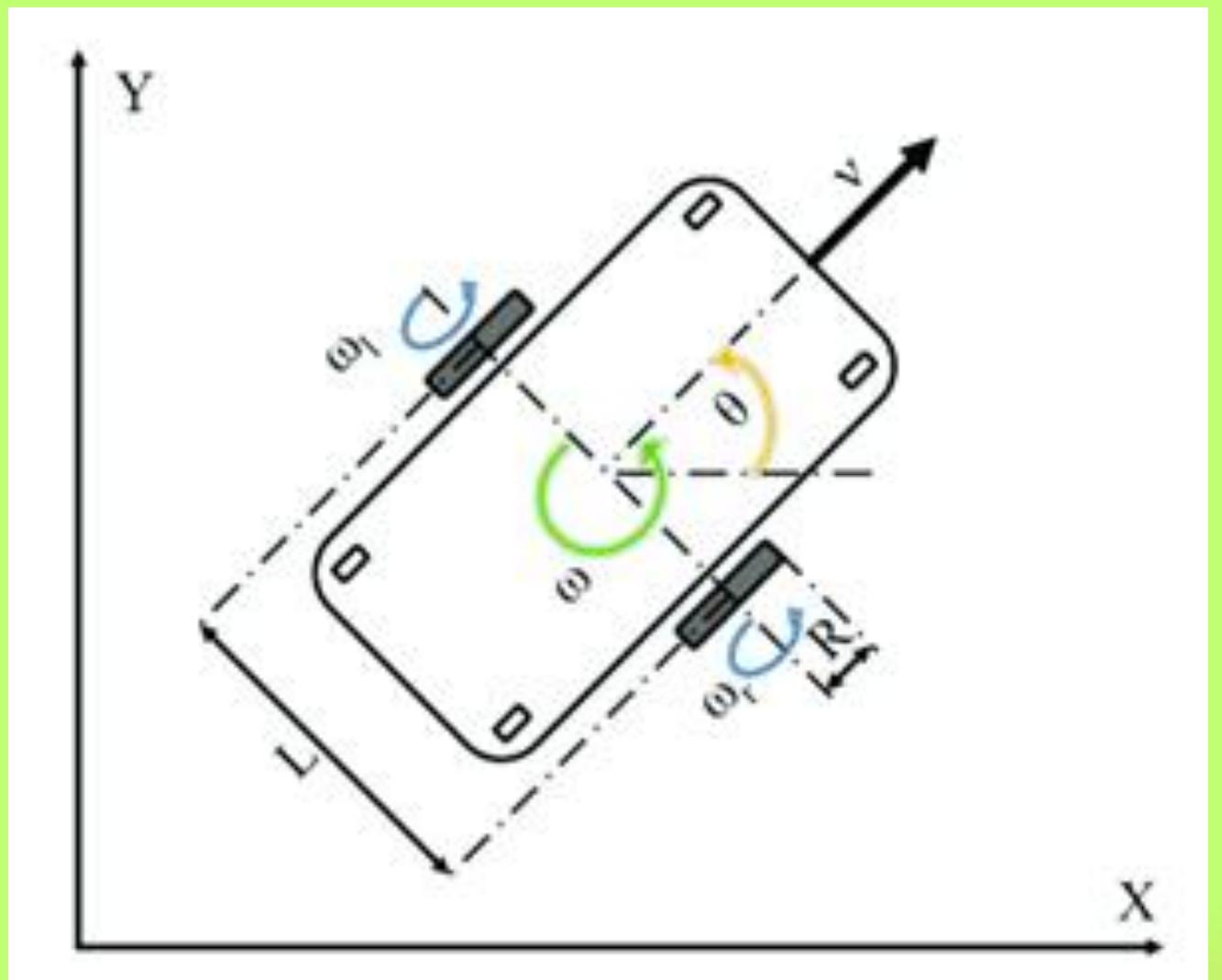
Trong mô hình này, hai bánh xe được đặt song song với nhau và cả 2 đều có thể quay độc lập, cho phép robot thực hiện các chuyển động phức tạp như quay 360 độ tại chỗ.

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



# CHƯƠNG 6. TỰ ĐỊNH VỊ BẢN THÂN ACML

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



$$V_x = V(t) \cos(\theta(t))$$

$$V_y = V(t) \sin(\theta(t))$$

Thus,

$$x(t) = \int V(t) \cos(\theta(t)) dt$$

$$y(t) = \int V(t) \sin(\theta(t)) dt$$

$$\theta(t) = \int \omega(t) dt$$

with

$$\omega = (V_R - V_L) / 2d$$

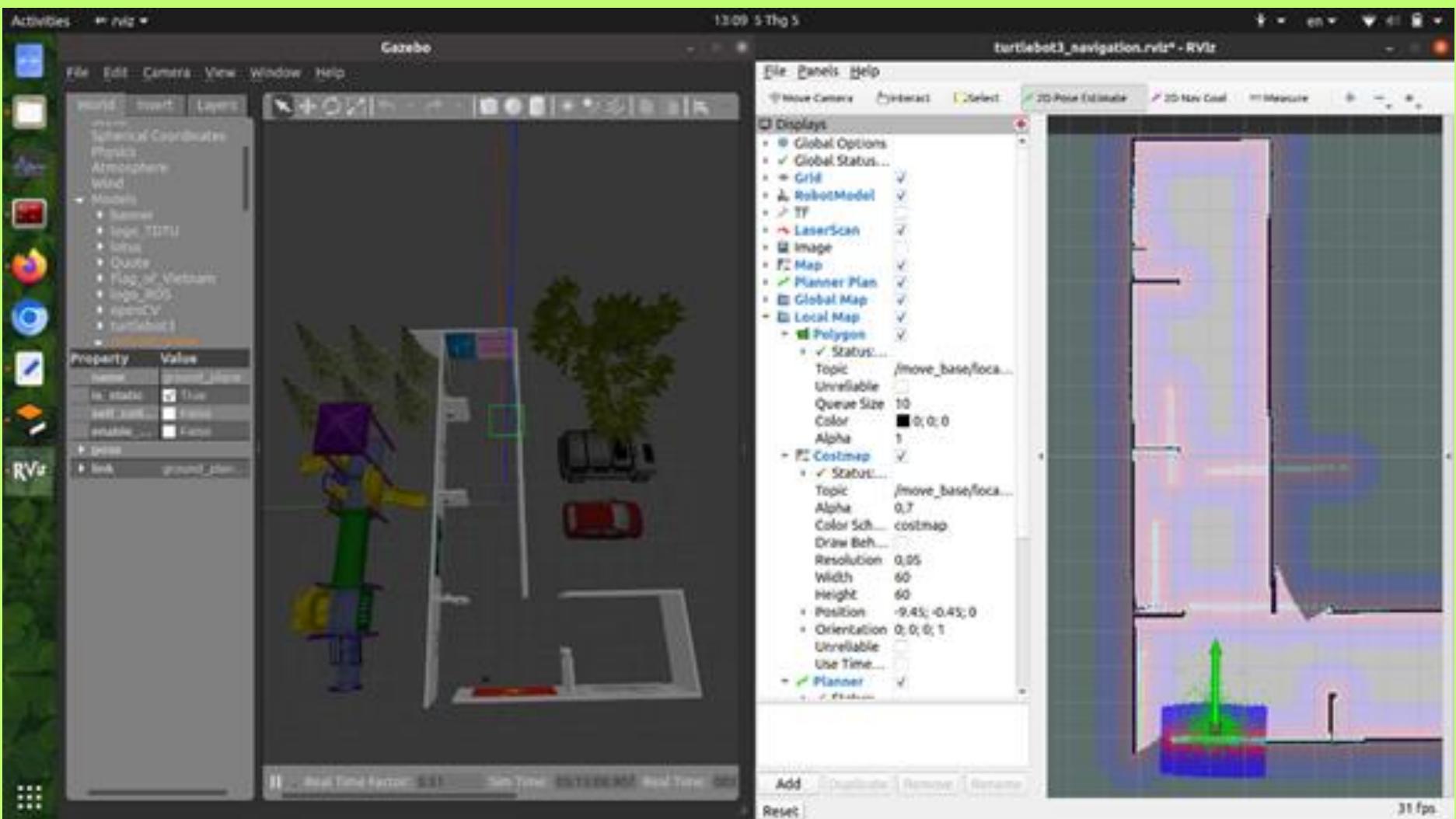
$$R = 2d (V_R + V_L) / (V_R - V_L)$$

$$V = \omega R = (V_R + V_L) / 2$$

# CHƯƠNG 6. TỰ ĐỊNH VỊ BẢN THÂN ACML

## Thực hiện ACML

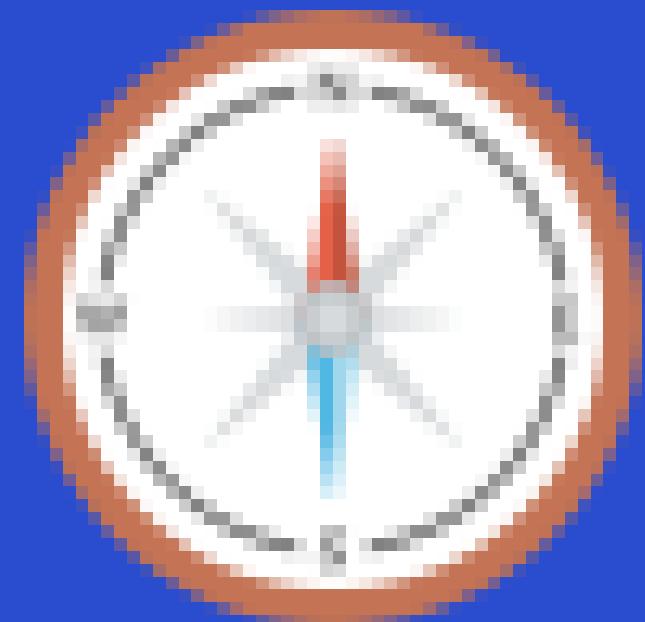
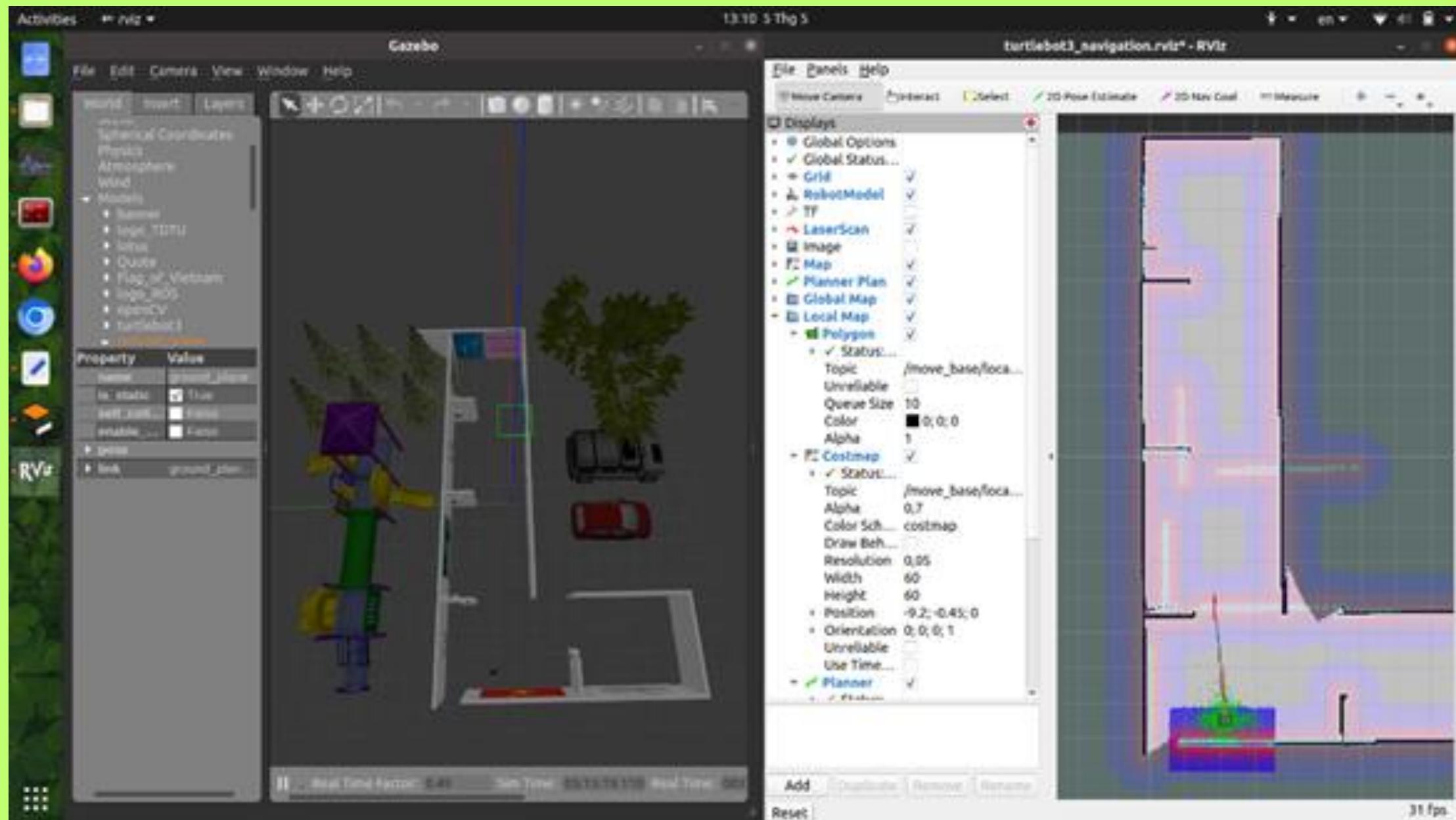
■ Khi khởi chạy, vị trí của Turtlebot3 trong Gazebo và RVIZ sẽ khác nhau. Cần chọn vào nút 2D Pose Estimate trong RVIZ và đánh dấu lại vị trí robot giống với robot bên Gazebo



# CHƯƠNG 6. TỰ ĐỊNH VỊ BẢN THÂN ACML

## Thực hiện ACML

Khi robot đã đúng vị trí, chọn nút 2D Nav Goal để đánh dấu đích để robot di chuyển





# MÔ HÌNH PHÁT HIỆN BIỂN BÁO TỐC ĐỘ



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

## Định nghĩa máy học

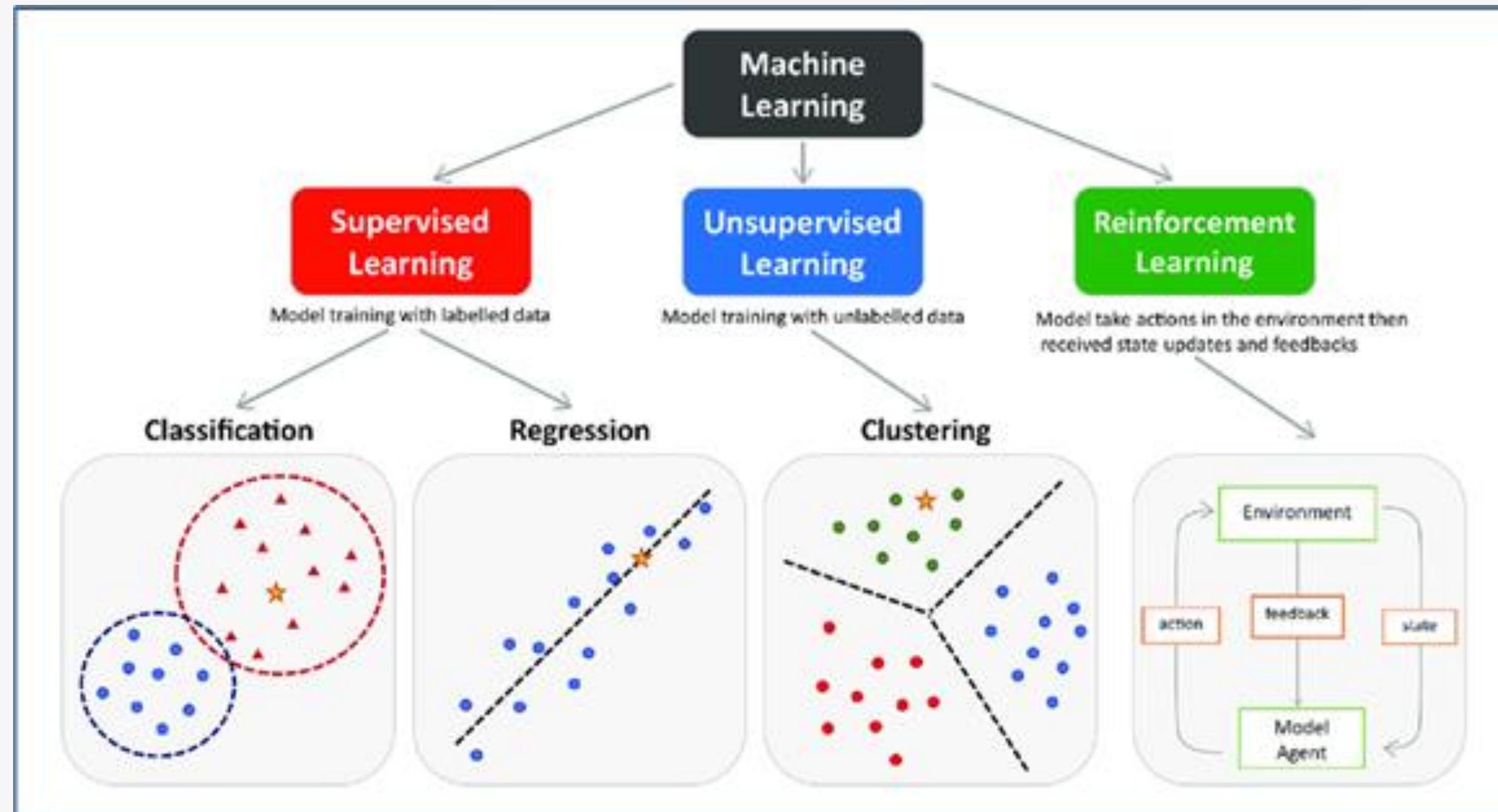
Máy học, hay còn gọi là học máy (machine learning), giúp cho robot có khả năng tự học từ dữ liệu và kinh nghiệm, tự điều chỉnh hành vi dựa trên những thông tin mới. Điều này làm cho robot không chỉ thực hiện các nhiệm vụ cơ học mà còn có thể nhận thức, phân tích thông tin và ra quyết định một cách linh 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.



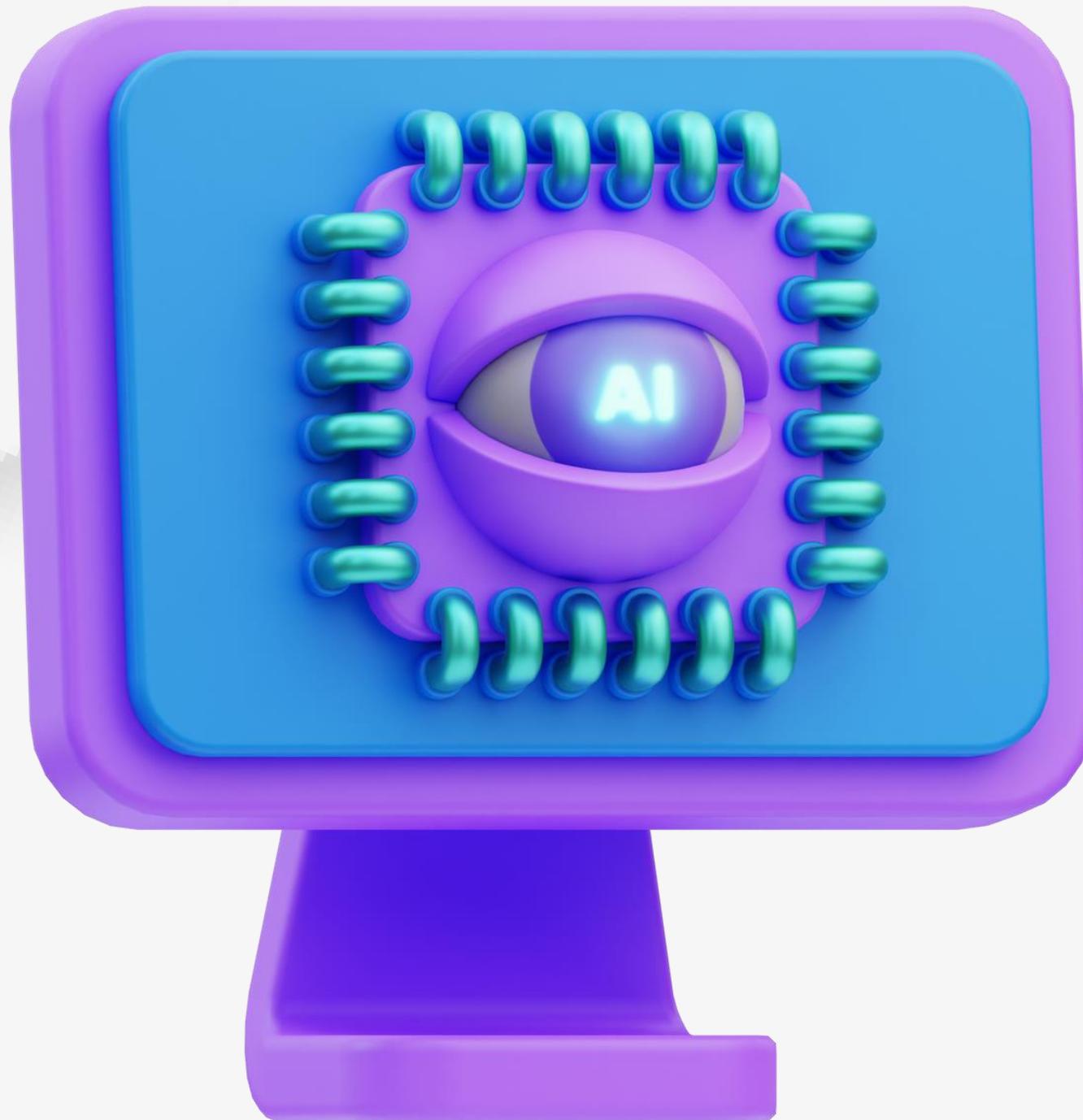
# CHƯƠNG 7. MÔ HÌNH PHÁT HIỆN BIỂN BÁO TỐC ĐỘ

## Phân loại máy học



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

## Định nghĩa Thị giác máy tính

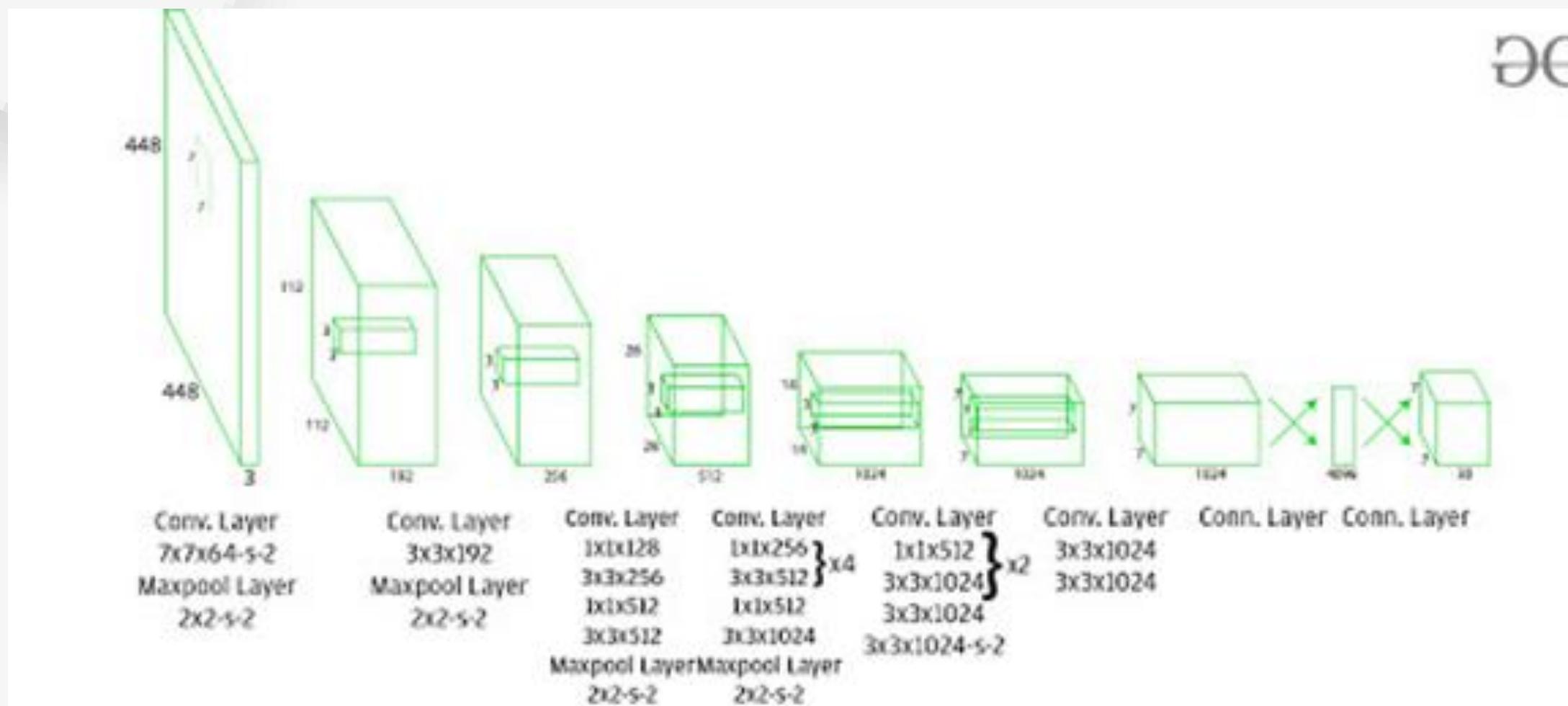


**Thị giác máy tính (Computer Vision)** là một phân nhánh của trí tuệ nhân tạo tập trung vào việc phát triển các phương pháp và công nghệ để máy tính có thể hiểu và phân tích hình ảnh và video một cách tự động. Mục tiêu của thị giác máy tính là làm cho máy tính (và robot) có khả năng "nhìn" và "hiểu" 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.

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

## 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

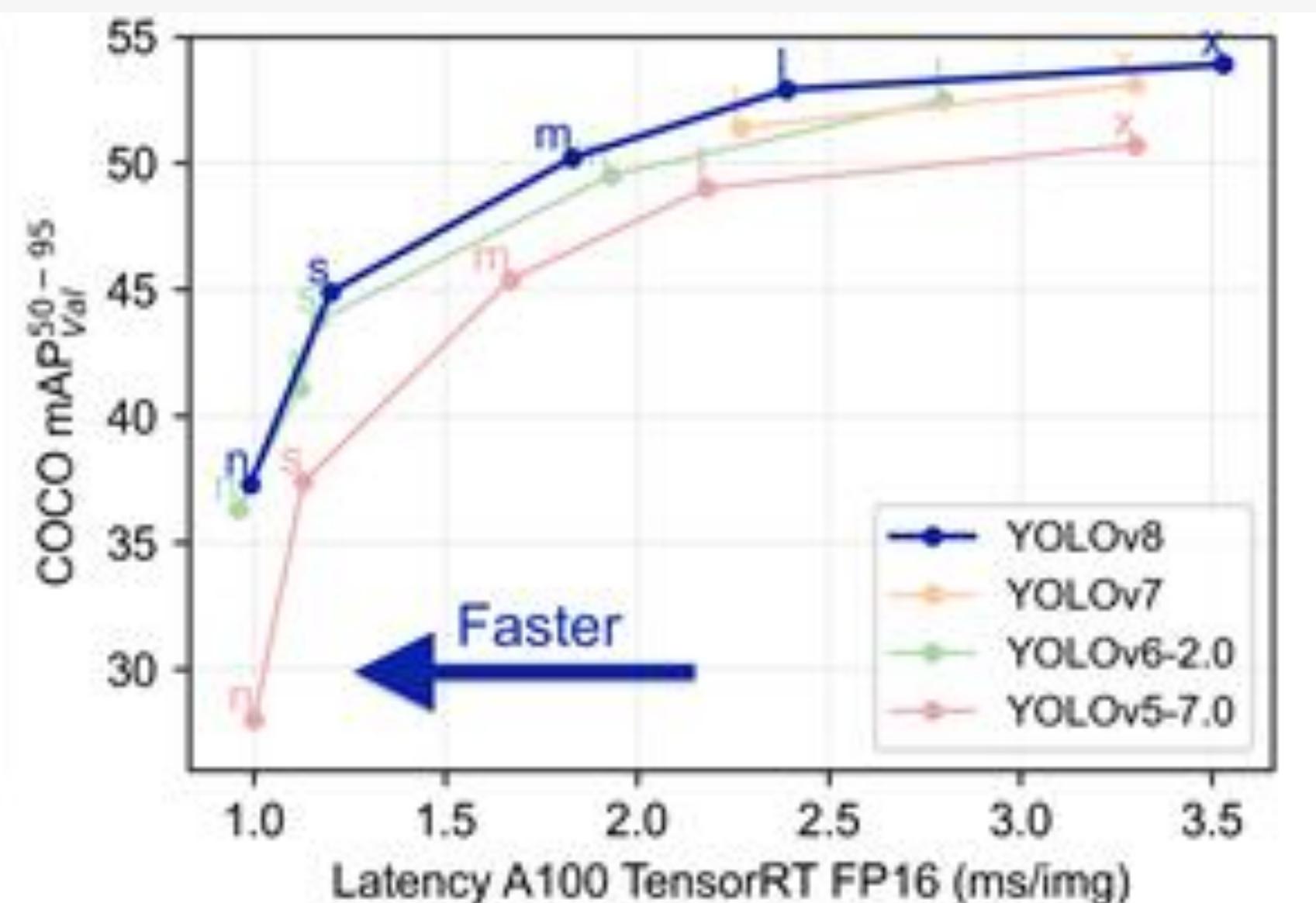
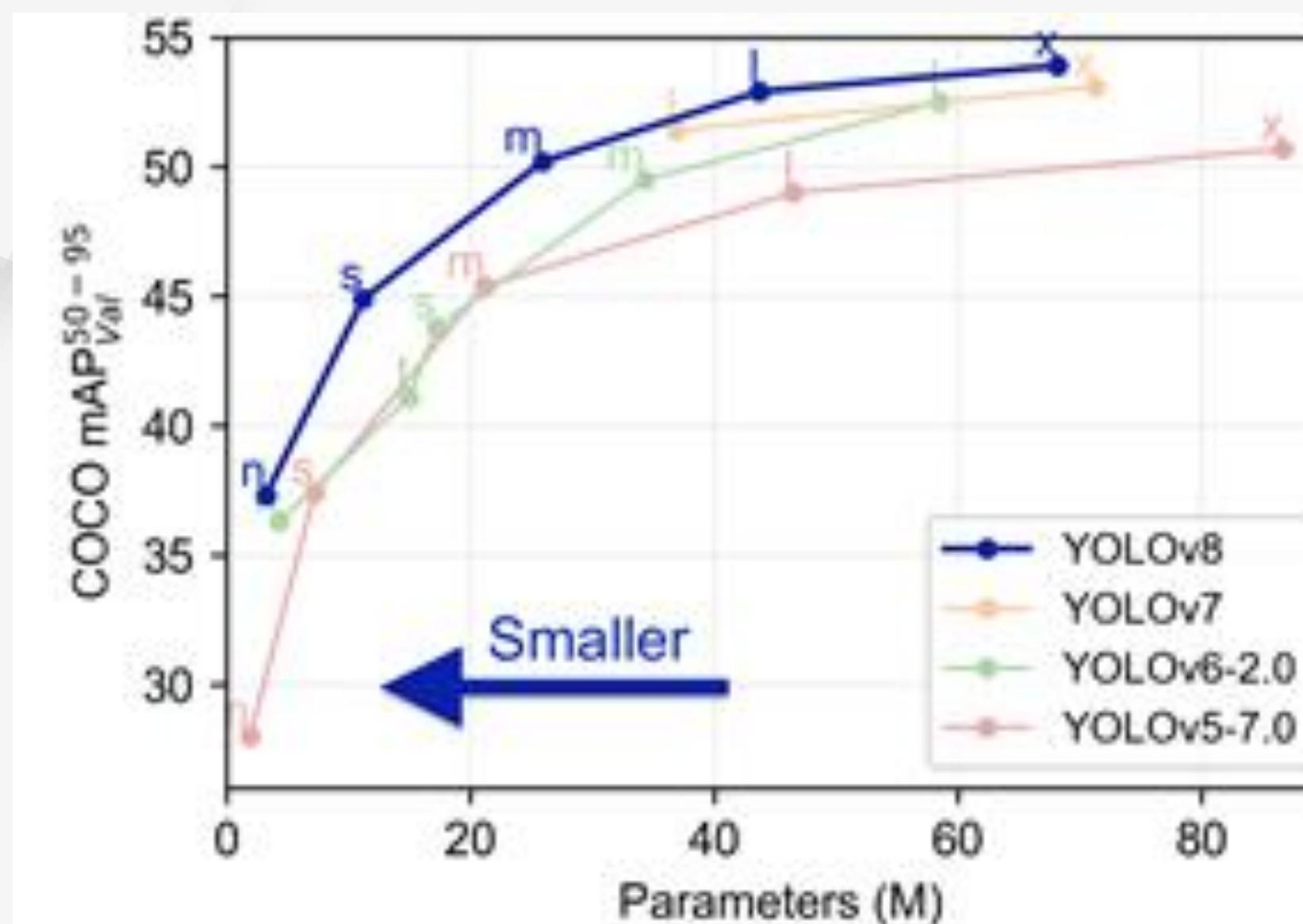


**YOLO sử dụng một mạng nơ-ron tích  
chập (CNN) để  
xử lý hình ảnh, chia hình ảnh thành  
lưới, thực hiện nhiều dự đoán cho  
mỗi ô lưới, lọc  
ra các dự đoán có độ tin cậy thấp, và  
sau đó loại bỏ các hộp chồng chéo  
để tạo ra đầu  
ra cuối cùng.**

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

## Mô hình YOLOv8

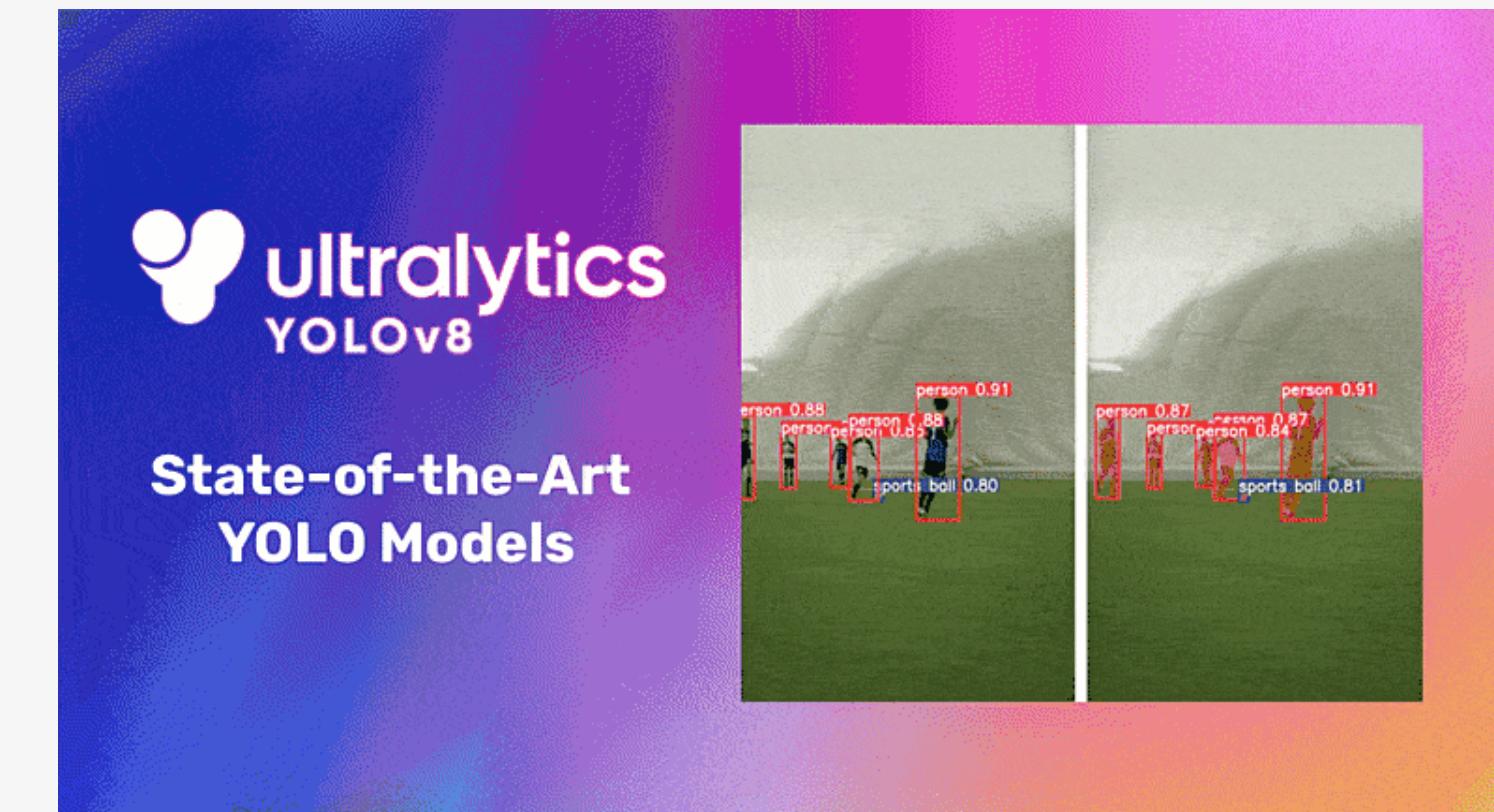
Mô hình YOLOv8 được xây dựng dựa trên các phiên bản YOLO trước đó bằng cách giới thiệu các cải tiến khác nhau về độ chính xác, tốc độ và độ tin cậy



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

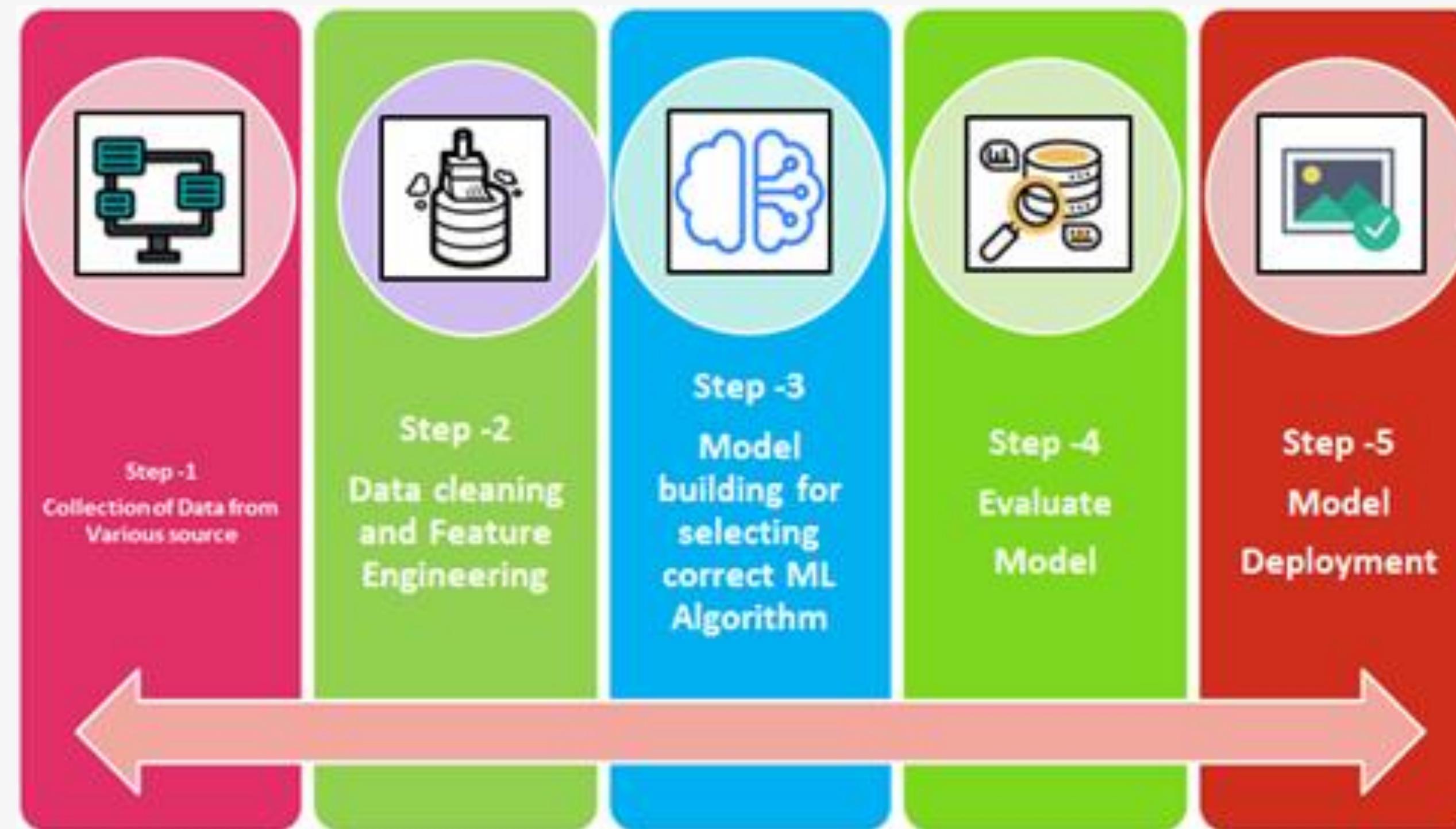
Mô hình	Kích thước (pixels)	mAP val 50-95	Tốc độ CPU ONNX (ms)	Tốc độ A100 TensorRT (ms)	Params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	575.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

## Mô hình YOLOv8



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

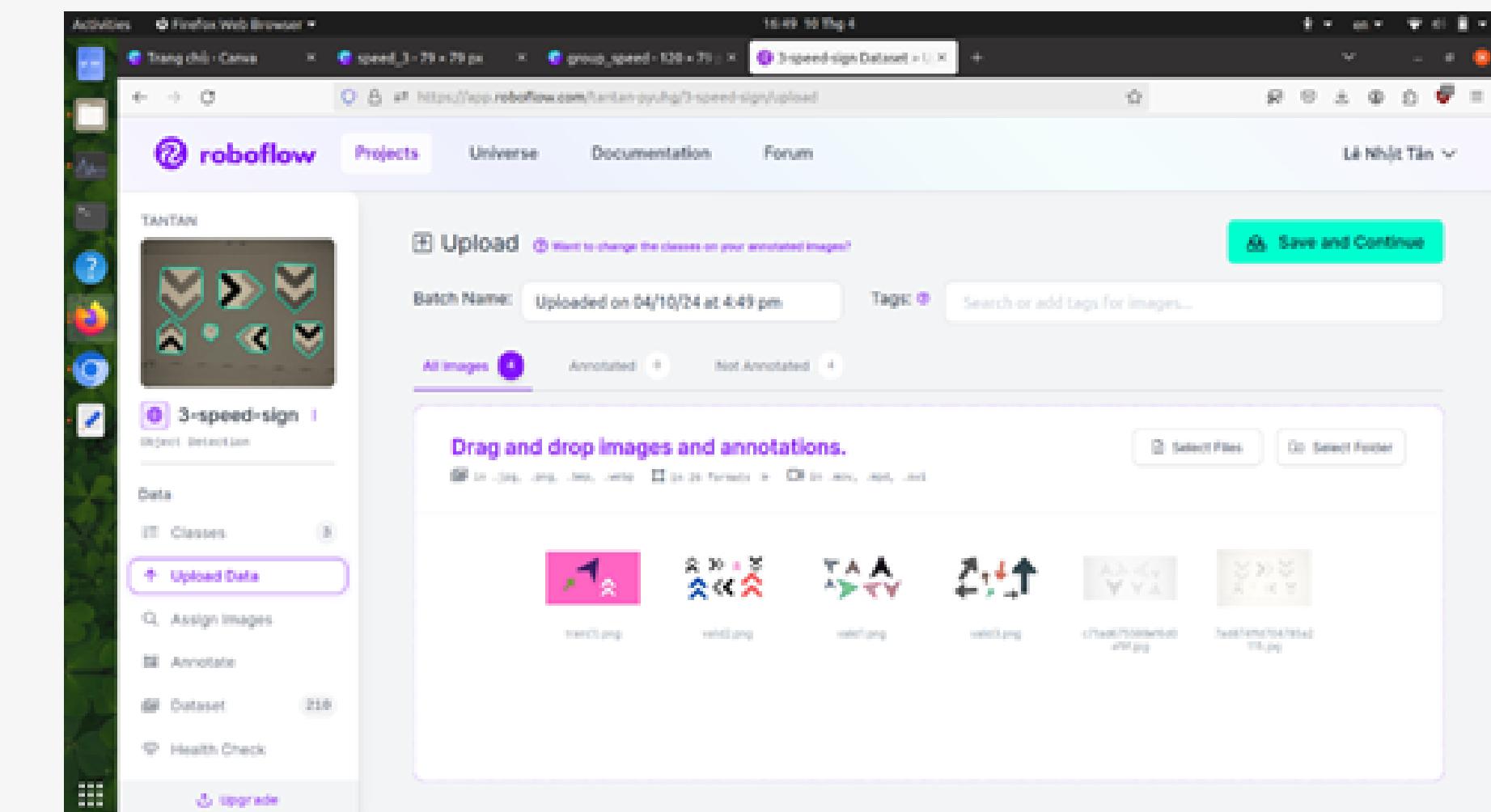
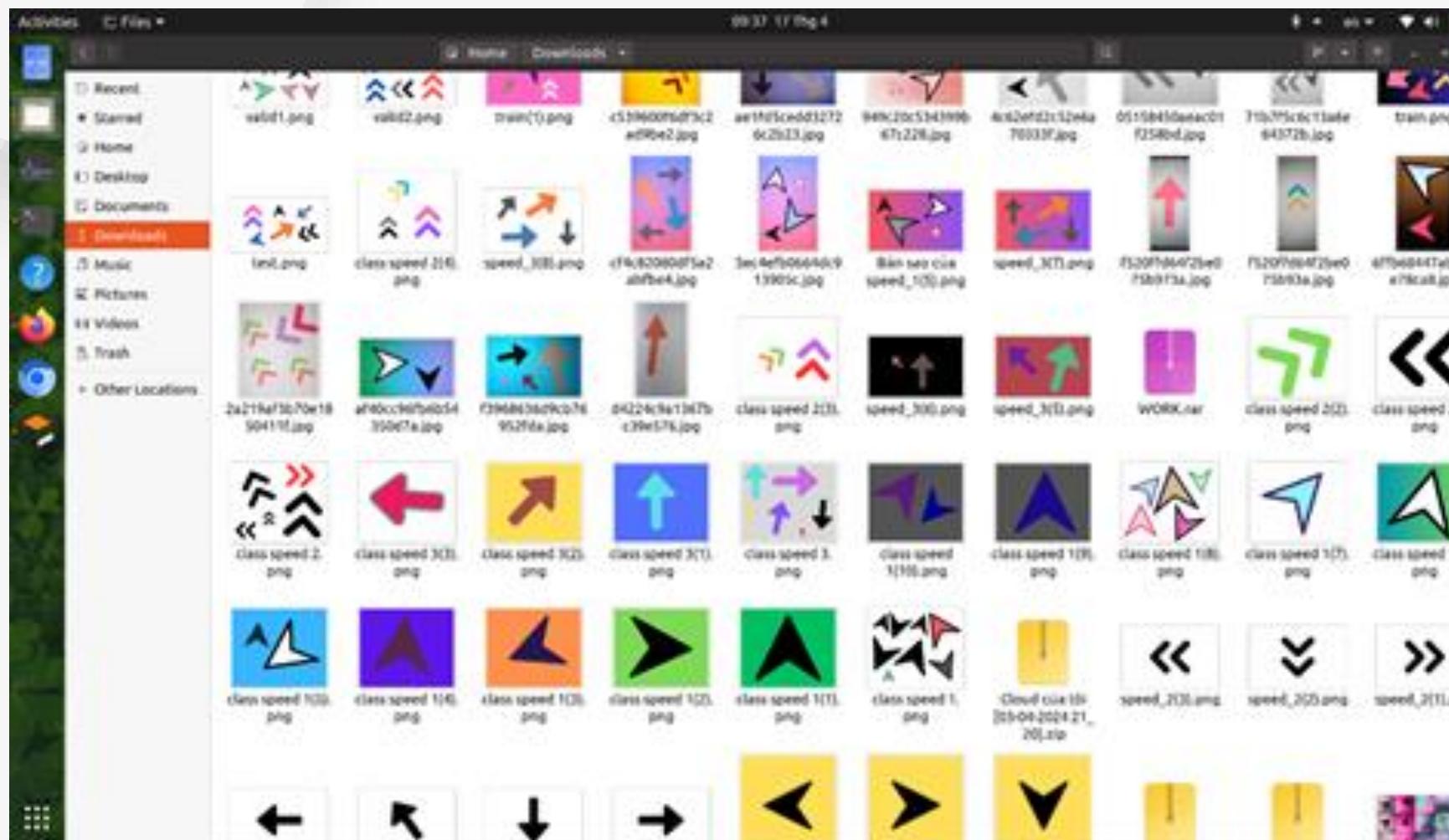
## Quá trình học máy



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

## Đào tạo mô hình phát hiện biển báo

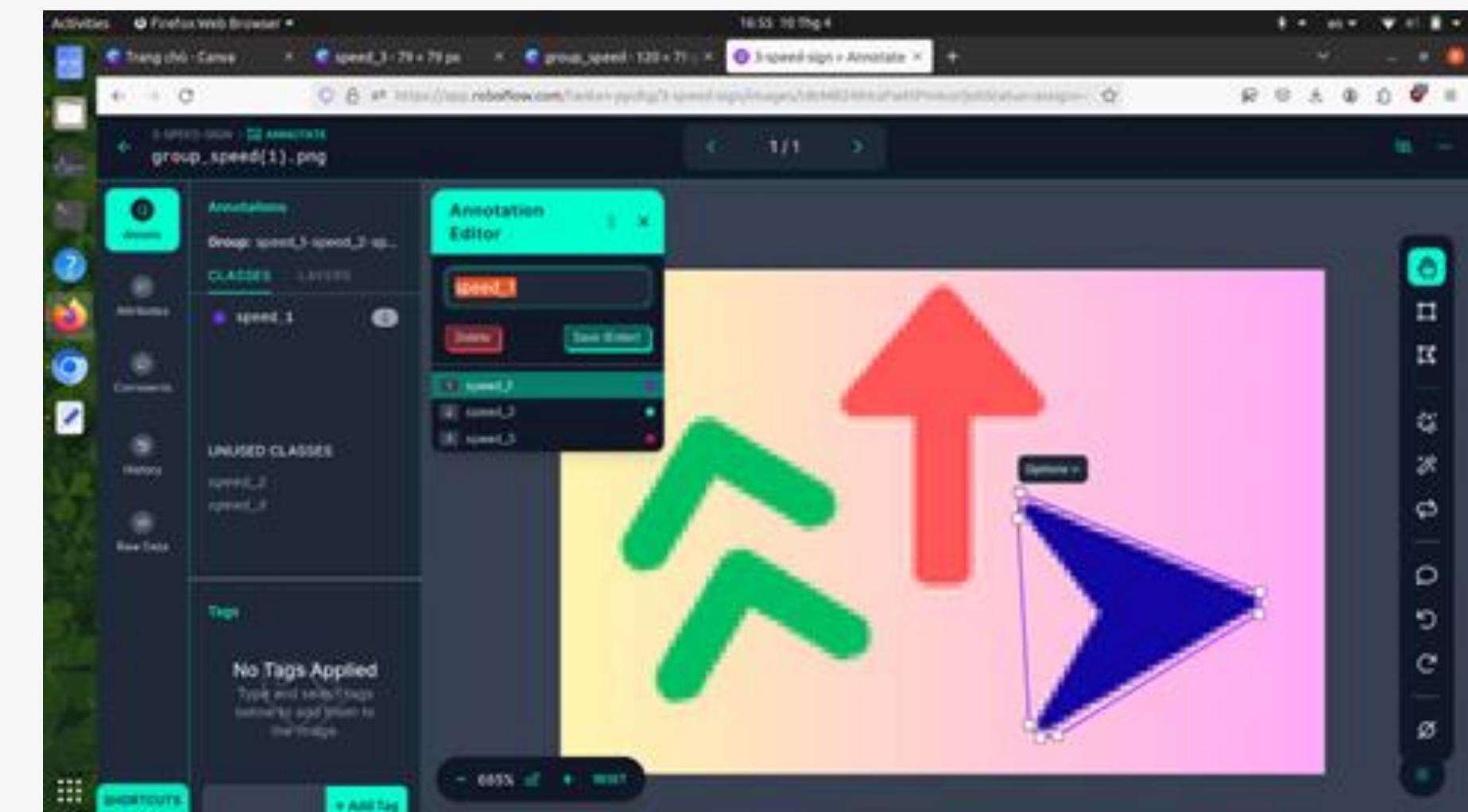
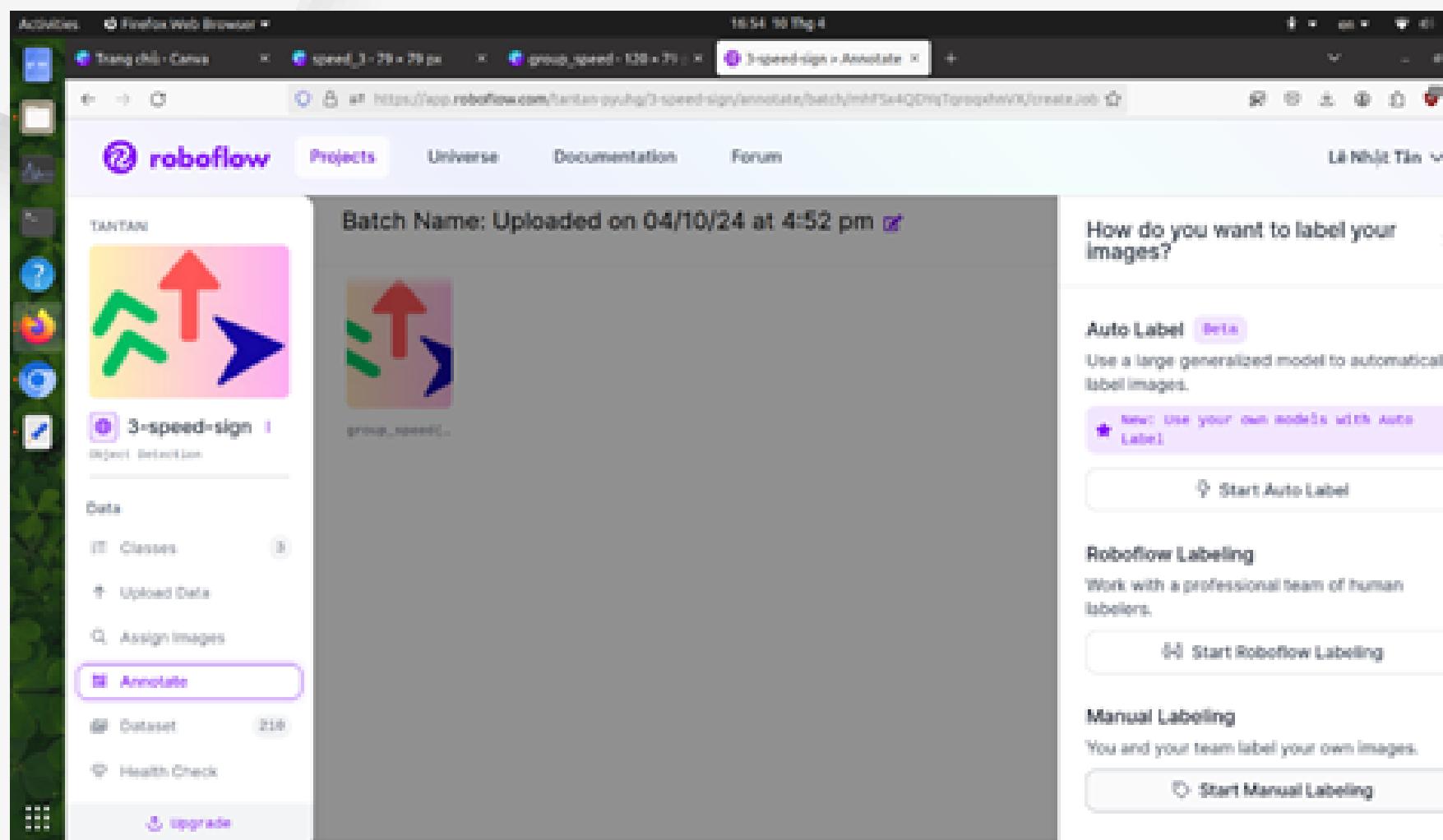
### Chuẩn bị hình ảnh



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

## Đào tạo mô hình phát hiện biển báo

### Dán nhãn hình ảnh

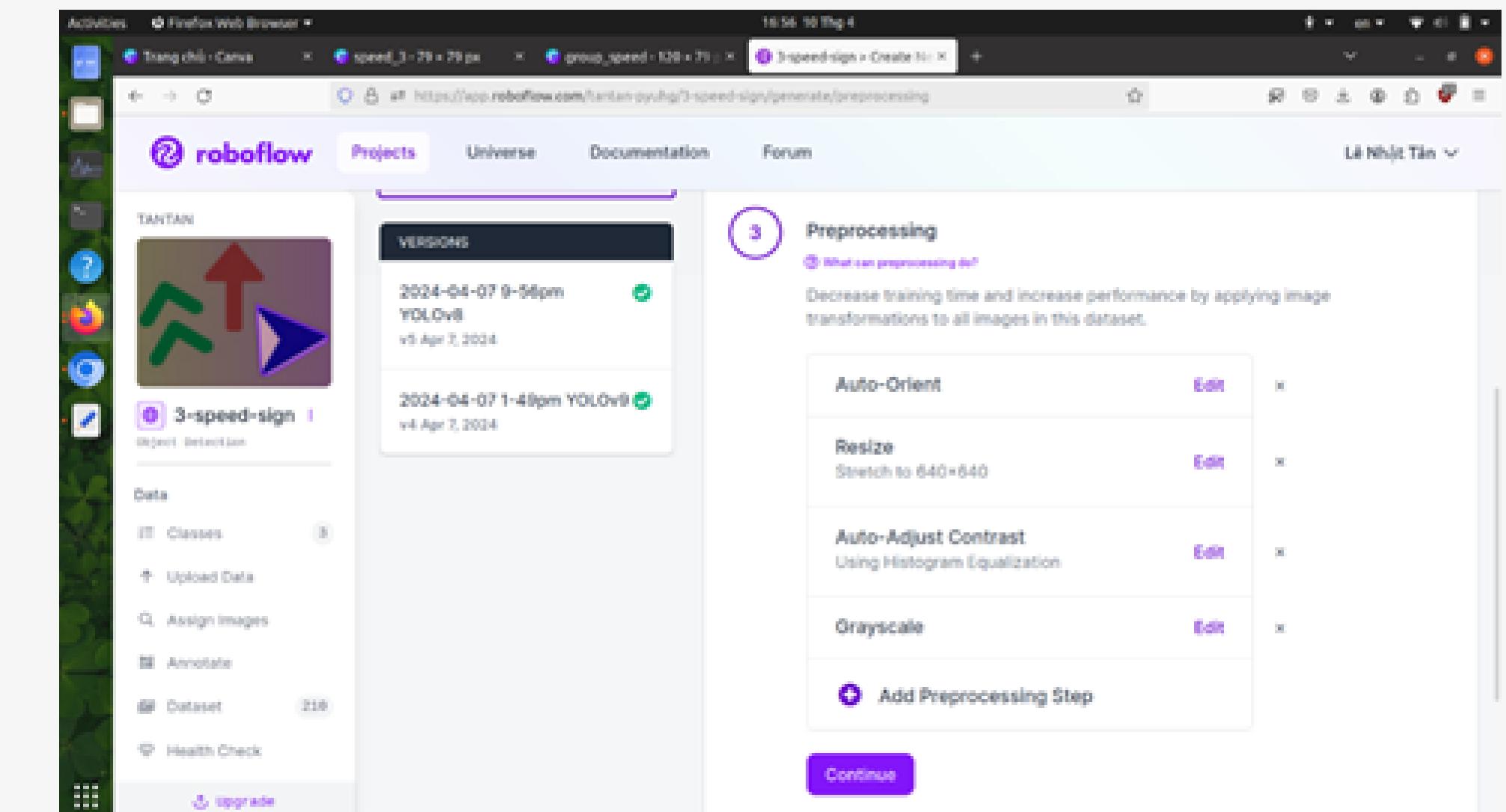


# CHƯƠNG 7. MÔ HÌNH PHÁT HIỆN BIỂN BÁO TỐC ĐỘ

## Đào tạo mô hình phát hiện biển báo

### 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**

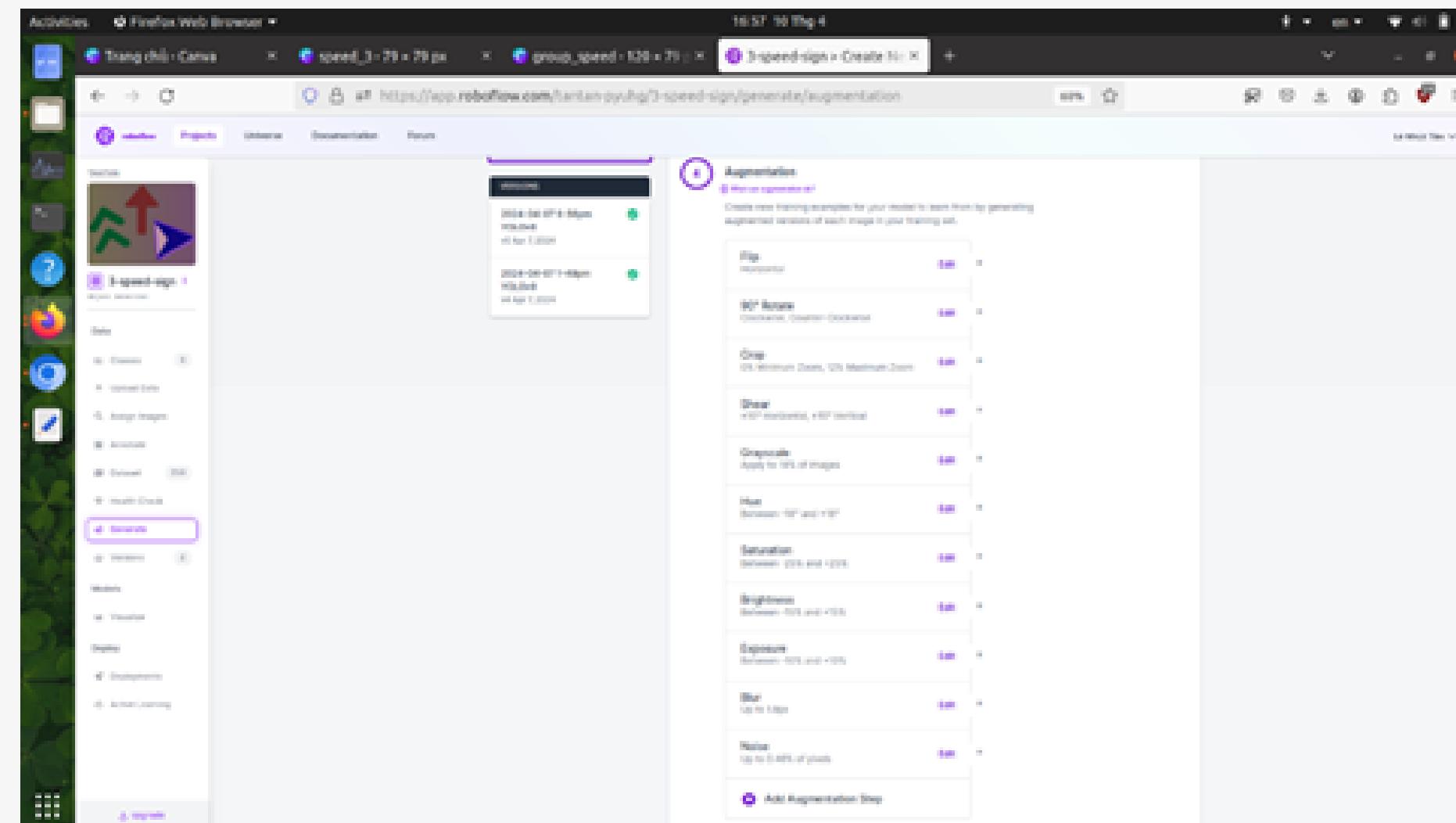


# CHƯƠNG 7. MÔ HÌNH PHÁT HIỆN BIỂN BÁO TỐC ĐỘ

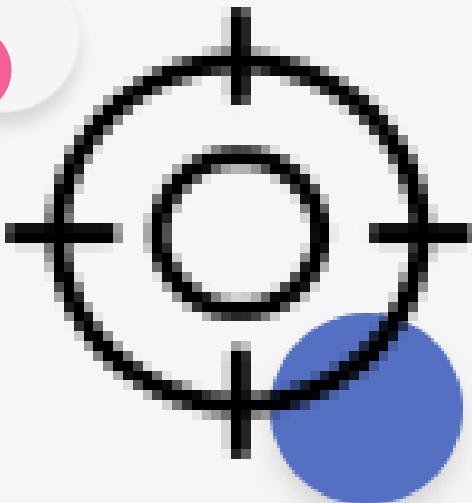
# Đào tạo mô hình phát hiện biển báo

# 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 nhỏ lên dữ liệu.**



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



#YOLO

## Đào tạo mô hình phát hiện biển báo

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

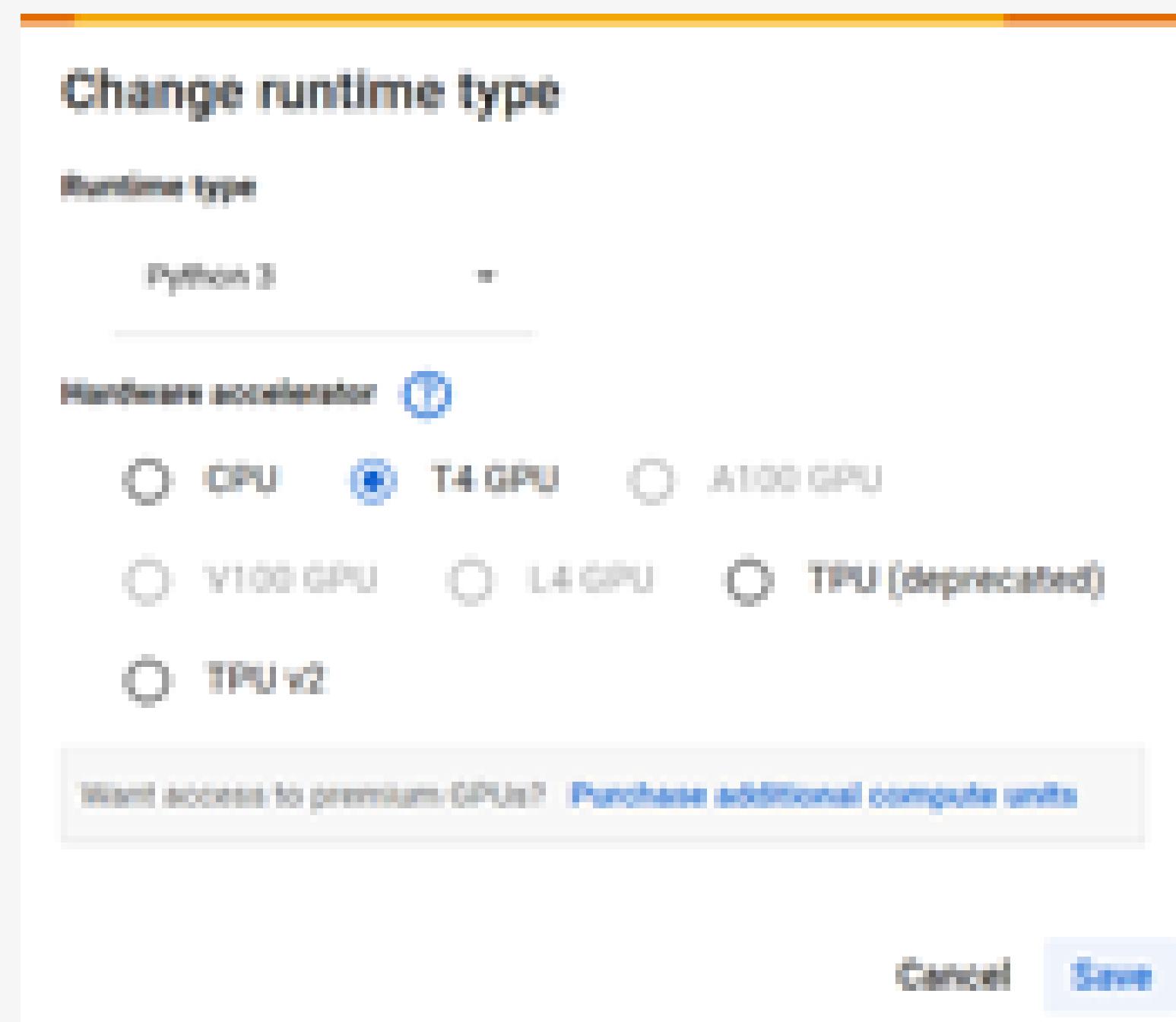


```
File Edit View Insert Runtime Tools Help cannot save changes
Code Text Copy to Drive
Install YOLOv8
YOLOv8 is still under heavy development. Breaking changes are being introduced almost weekly. We strive to make our YOLOv8 notebooks work with the latest version of the library. Last tests took place on 03.01.2024 with version YOLOv8.0.196.
If you notice that our notebook behaves incorrectly - especially if you experience errors that prevent you from going through the tutorial - don't hesitate! Let us know and open an issue on the Roboflow Notebooks repository.
YOLOv8 can be installed in two ways - from the source and via pip. This is because it is the first iteration of YOLO to have an official package.

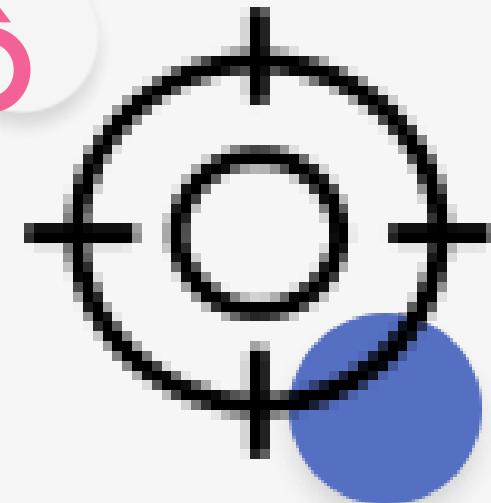
Pip install method (recommended)
!pip install ultralytics==8.0.196
from IPython import display
display.clear_output()
import ultralytics
ultralytics.checks()

Ultralytics YOLOv8.0.196 Python-3.10.12 torch-2.2.1+cu131 CUDA:0 (Tesla T4, 15GBRAM)
Setup complete (2 CPUs, 12.7 GB RAM, 29.678.2 GB disk)

from ultralytics import YOLO
from IPython.display import display, Image
Executing (In 42): <cell line 3> in system() in _system_compat() in _run_command() in _monitor_process() in _poll_process()
```



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

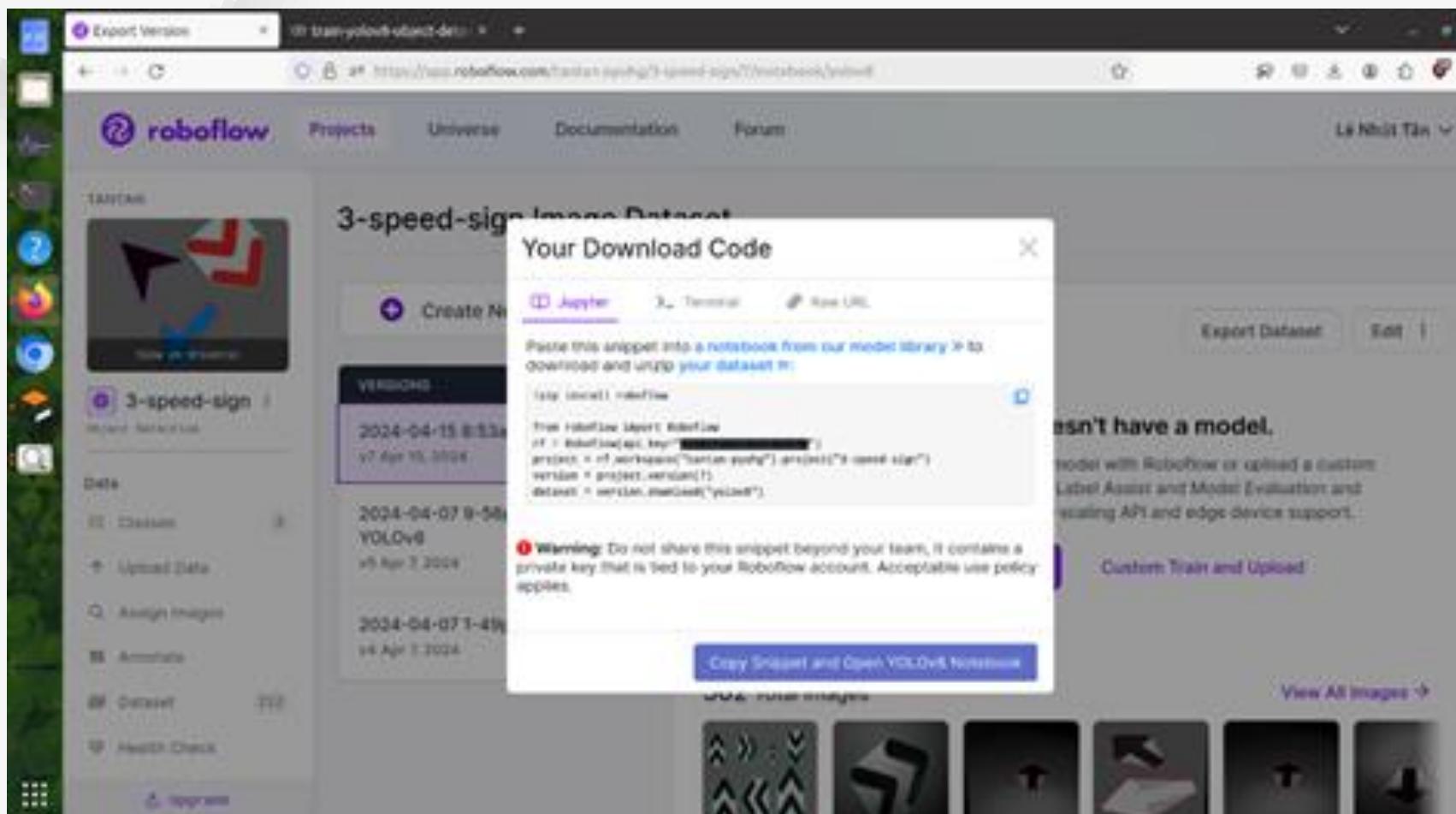


#YOLO

## Đào tạo mô hình phát hiện biển báo

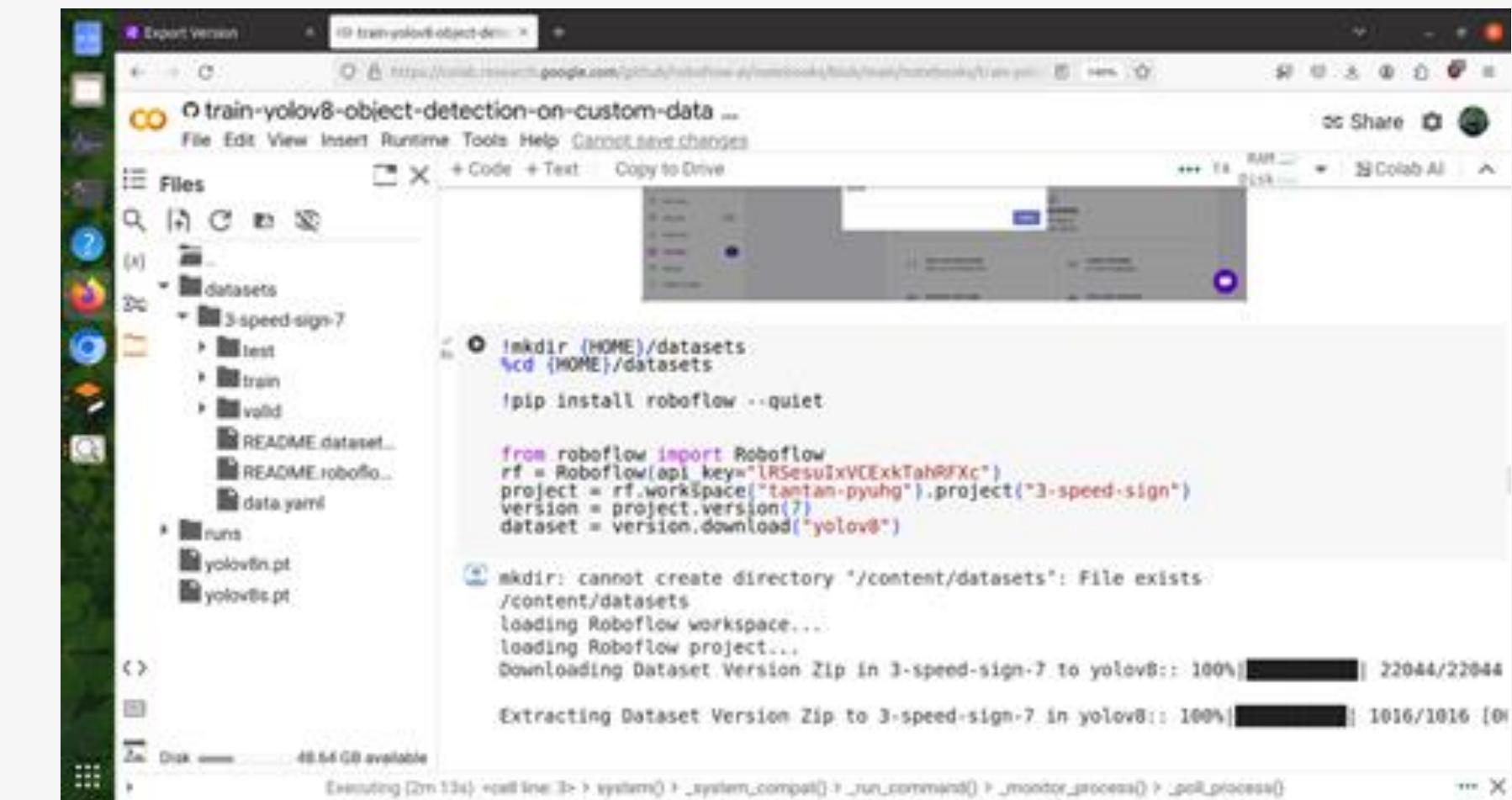
Thực hiện đào tạo hình ảnh trên nền tảng Google Colab

Sao chép API để tải về tệp hình ảnh đã dán nhãn từ Roboflow



```
from roboflow import Roboflow
rf = Roboflow(api_key="LR5esuIxVCEExkTahRFXc")
project = rf.workspace().get_project("3-speed-sign")
version = project.get_version(1)
dataset = version.get_dataset("yolov8")
```

Tải về tệp hình ảnh đã dán nhãn trên Google Colab



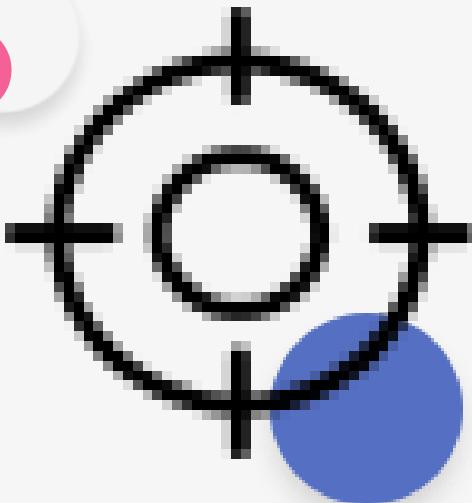
```
!mkdir {HOME}/datasets
%cd {HOME}/datasets
!pip install roboflow --quiet

from roboflow import Roboflow
rf = Roboflow(api_key="LR5esuIxVCEExkTahRFXc")
project = rf.workspace().get_project("3-speed-sign")
version = project.get_version(1)
dataset = version.download("yolov8")

mkdir: cannot create directory "/content/datasets": File exists
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in 3-speed-sign-7 to yolov8:: 100% [██████████] 22044/22044

Extracting Dataset Version Zip to 3-speed-sign-7 in yolov8:: 100% [██████████] 1016/1016 [0]
```

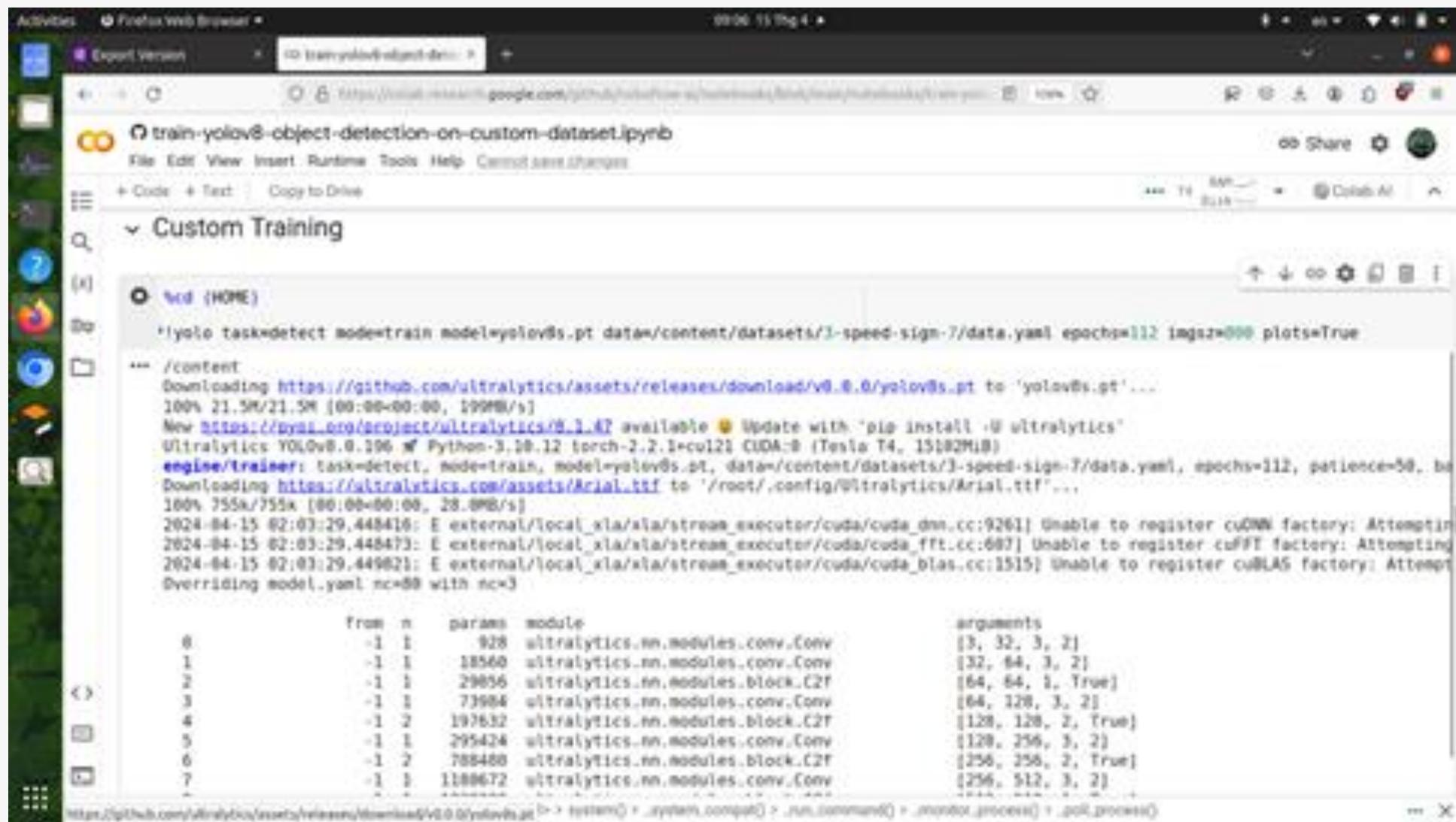
# CHƯƠNG 7. MÔ HÌNH PHÁT HIỆN BIỂN BÁO TỐC ĐỘ



#YOLO

## Đào tạo mô hình phát hiện biển báo

Thực hiện đào tạo hình ảnh với model yolov8s với số lần duyệt qua dataset (epochs) là 112 để đảm bảo độ chính xác của mô hình



```
!yolov8 task=detect mode=train model=yolov8s.pt data=/content/datasets/3-speed-sign-7/data.yaml epochs=112 imgsz=960 plots=True
```

The screenshot shows a Jupyter Notebook interface with the title "train-yolov8-object-detection-on-custom-dataset.ipynb". The code cell contains the command to train a YOLOv8s model on a custom dataset of speed signs. The output of the command is displayed below, showing the download of the model file and the training process starting on a Tesla T4 GPU.

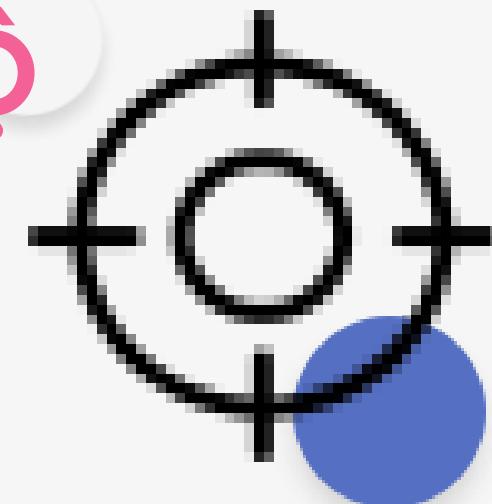
```
!yolov8 task=detect mode=train model=yolov8s.pt data=/content/datasets/3-speed-sign-7/data.yaml epochs=112 imgsz=960 plots=True
```

```
*** /content
Downloading https://github.com/ultralytics/assets/releases/download/v8.0.0/yolov8s.pt to 'yolov8s.pt'...
100% 21.5M/21.5M [00:00<00:00, 299MB/s]
New alias://pyai_aito/project/ultralytics@8.1.42 available ⓘ Update with "pip install -U ultralytics"
Ultralytics YOLOv8.0.396 M Python 3.10.12 torch 2.2.1+cu121 CUDA 9 (Tesla T4, 15GBM)
engine/trainer: task=detect, mode=train, model=yolov8s.pt, data=/content/datasets/3-speed-sign-7/data.yaml, epochs=112, patience=50, ba
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100% 755k/755k [00:00<00:00, 28.0MB/s]
2024-04-15 02:03:29,448416: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attemptin
2024-04-15 02:03:29,448473: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:687] Unable to register cuFFT factory: Attemptin
2024-04-15 02:03:29,449821: E external/local_xla/xla/stream_executor/cuda/cuda_bias.cc:1515] Unable to register cuBLAS factory: Attemptin
Overriding model.yaml nc=88 with nc=3
```

from	n	params	module	arguments
0	-1	928	ultralytics.nn.modules.conv.Conv	[3, 32, 3, 2]
1	-1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
2	-1	29656	ultralytics.nn.modules.block.C2f	[64, 64, 1, True]
3	-1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
4	-1	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
5	-1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
6	-1	788480	ultralytics.nn.modules.block.C2f	[256, 256, 2, True]
7	-1	1188672	ultralytics.nn.modules.conv.Conv	[256, 512, 3, 2]

https://github.com/ultralytics/assets/releases/download/v8.0.0/yolov8s.pt > tensor() > \_system.comput0 > run\_command() > monitor\_process() > poll\_process()

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



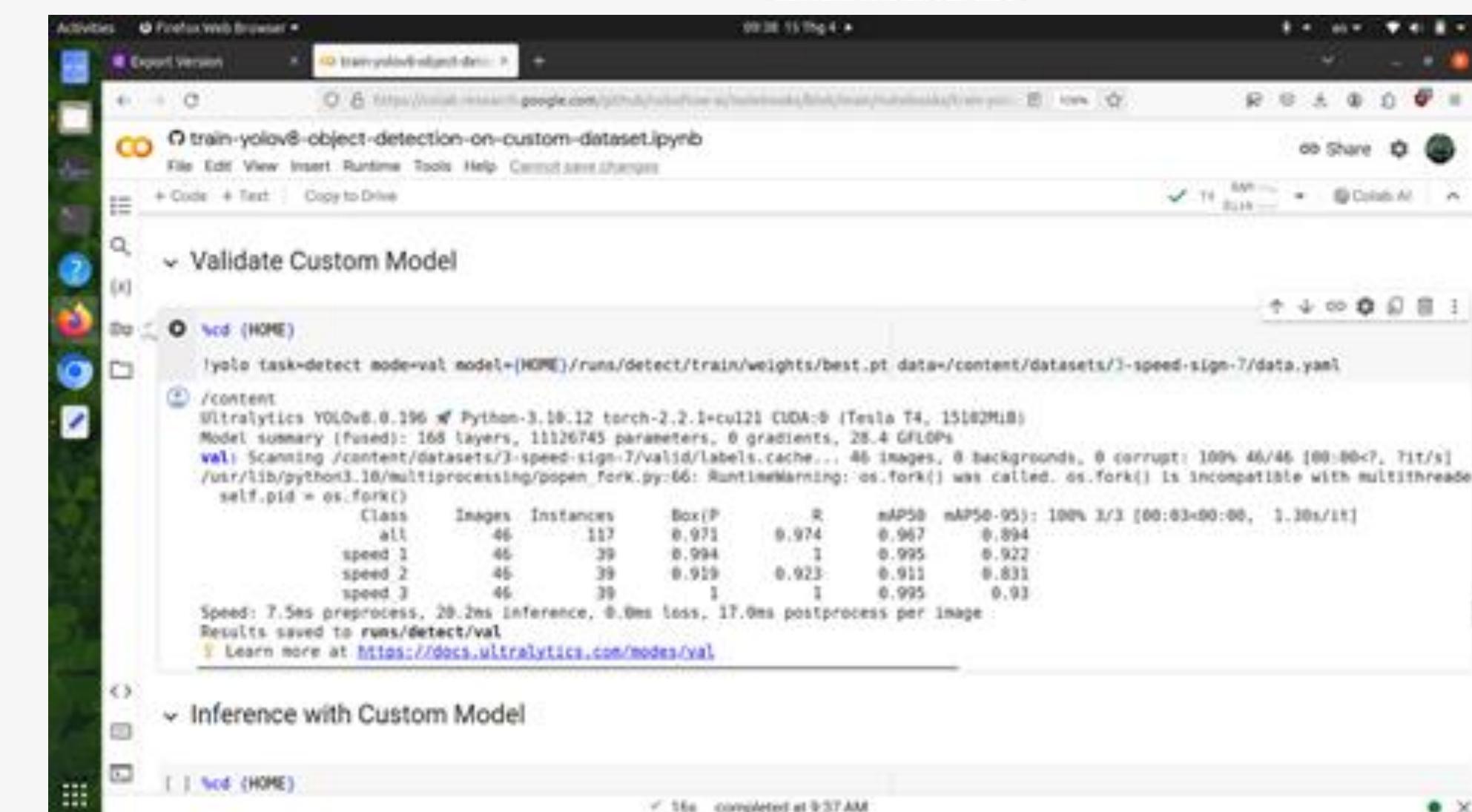
#YOLO

## Đào tạo mô hình phát hiện biển báo

Kiểm tra các tệp mới được tạo sau khi thực hiện đào tạo hình ảnh trên Google Colab

```
ls -l [8] !ls (HOME)/runs/detect/train/
```

[x]	args.yaml	PR_curve.png	train_batch2858.jpg
	confusion_matrix_normalized.png	R_curve.png	train_batch2.jpg
	confusion_matrix.png	results.csv	val_batch0_labels.jpg
	events.out.tfevents.1713146612.701f3d8640fe.1486.0	results.png	val_batch0_pred.jpg
	F1_curve.png	train_batch0.jpg	val_batch1_labels.jpg
	labels_correlogram.jpg	train_batch1.jpg	val_batch1_pred.jpg
	labels.jpg	train_batch2856.jpg	weights
	P_curve.png	train_batch2857.jpg	



The screenshot shows a Google Colab notebook titled "train-yolov8-object-detection-on-custom-dataset.ipynb". The notebook displays validation results for a custom model. The validation summary table is as follows:

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	0.833
speed 3	46	39	1	1	0.995	0.93

Below the table, the text reads: "Speed: 7.5ms preprocess, 20.2ms inference, 0.0ms loss, 17.0ms postprocess per image. Results saved to runs/detect/val. Learn more at <https://docs.ultralytics.com/modes/val>.

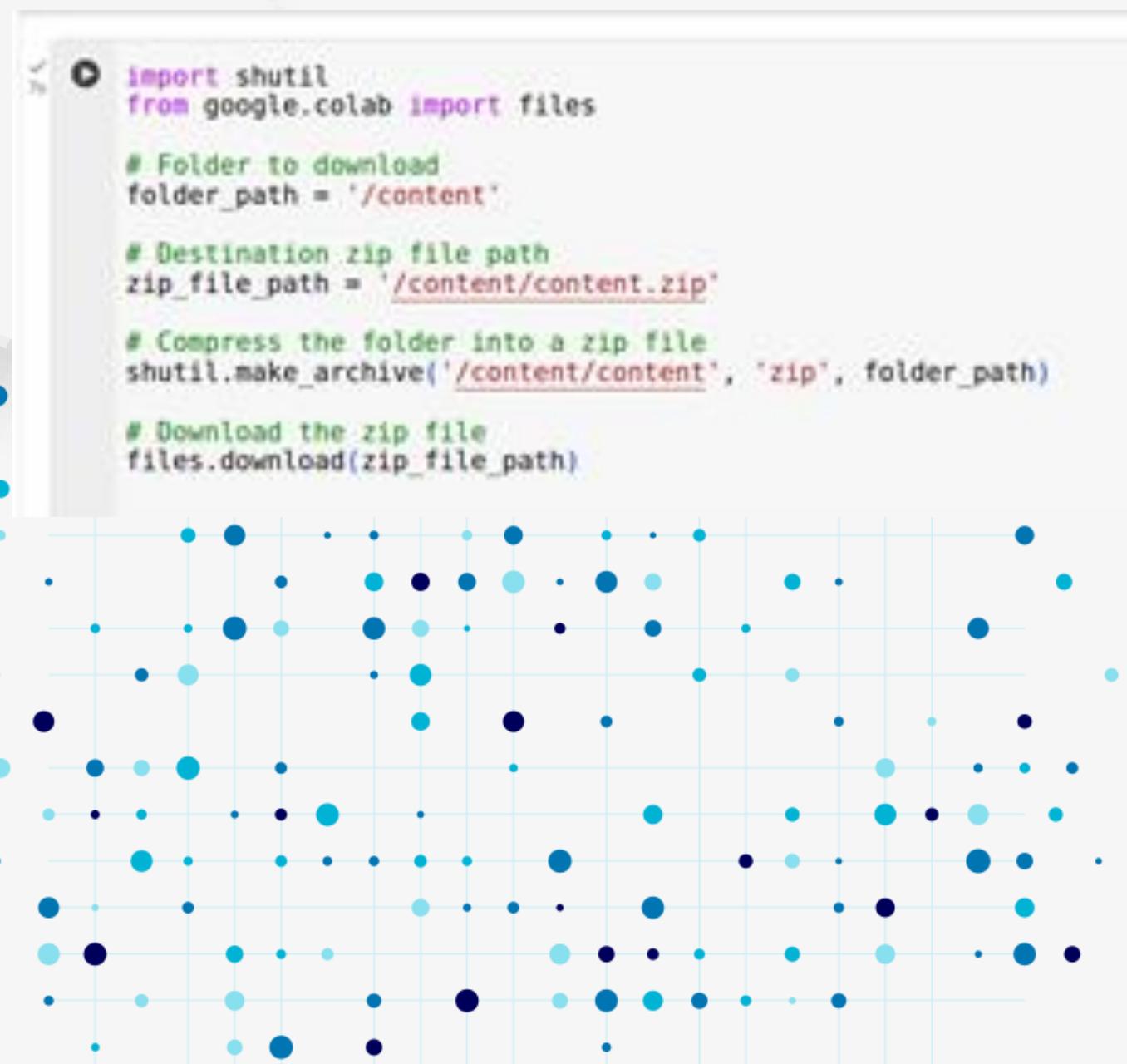
Sau khi đào tạo, cần thực hiện thẩm định (validation). Mục đích của việc này là để đánh giá mức độ tổng quát của mô hình, tức là khả năng của mô hình khi áp dụng vào dữ liệu mà nó chưa từng thấy. Điều này giúp ngăn chặn hiện tượng overfitting, khi mô hình quá khớp với dữ liệu huấn luyện và không thể tổng quát hóa tốt trên dữ liệu mới.

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

#YOLO

## Đào tạo mô hình phát hiện biển báo

Chạy thêm lệnh sau để tải về thư mục content, là toàn bộ dữ liệu liên quan đến dữ liệu đào tạo đã tạo trên ổ đĩa ảo.



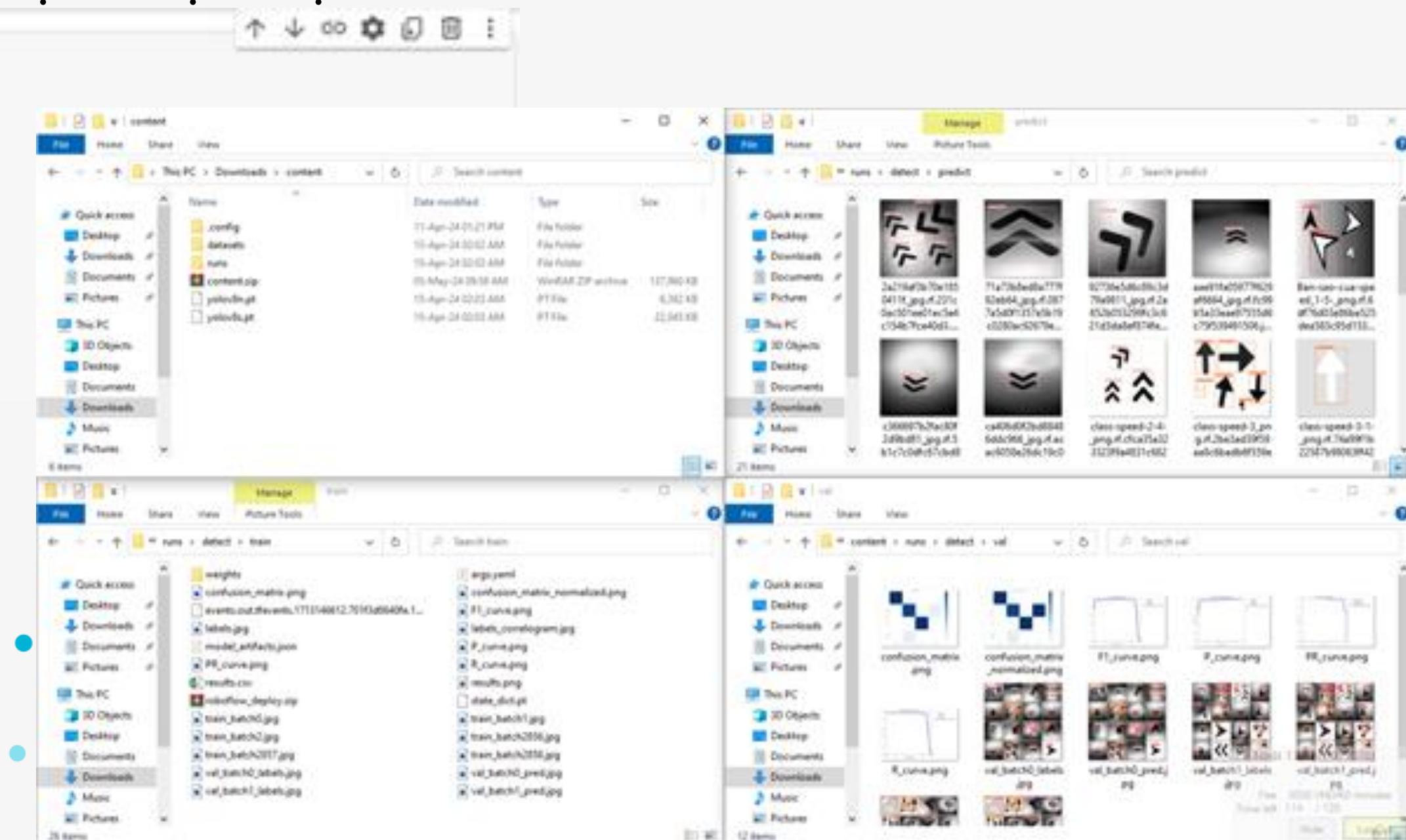
```
import shutil
from google.colab import files

# 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)
```



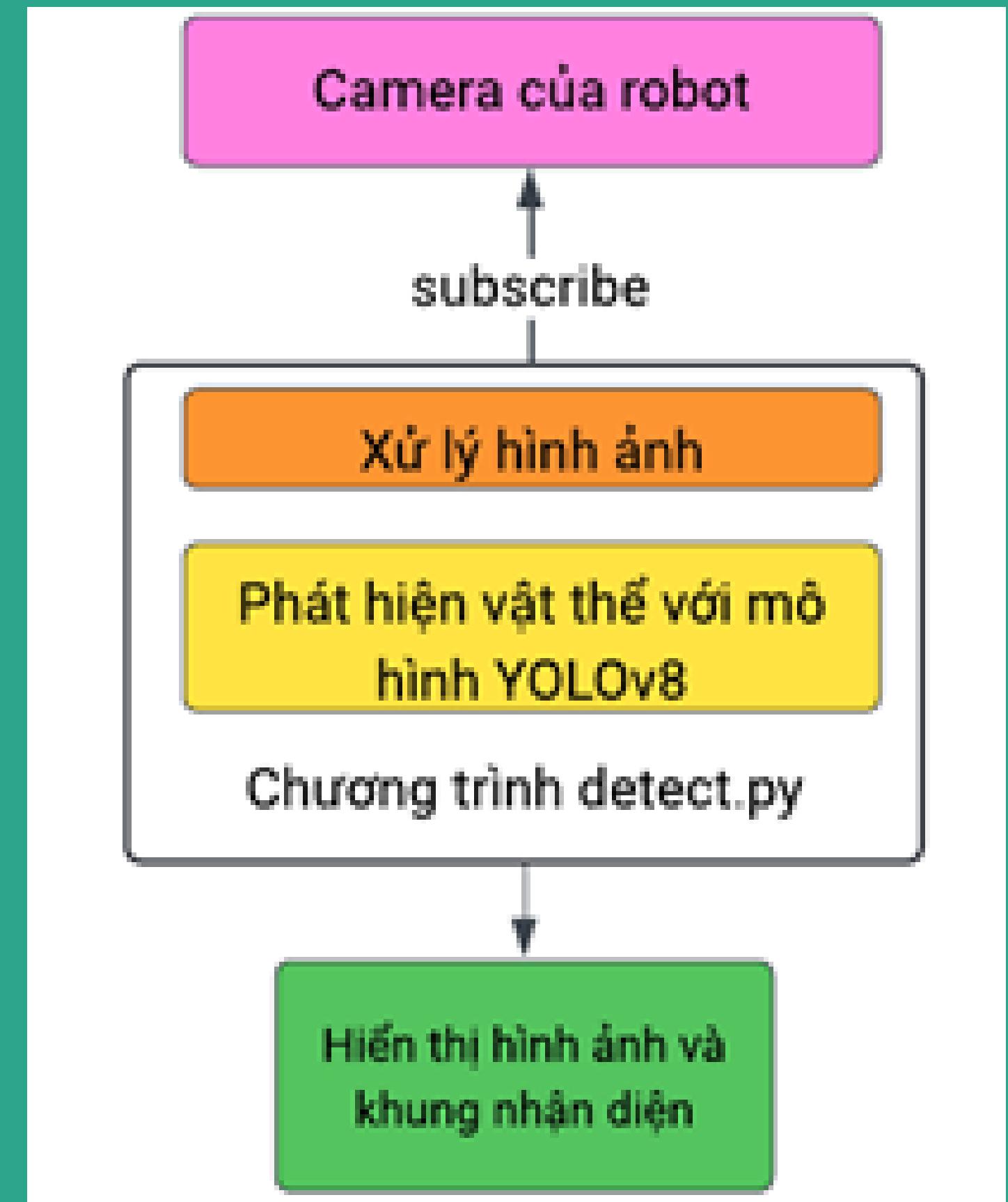
8

# THỰC THI PHẦN MỀM



# CHƯƠNG 8. THỰC THI PHẦN MỀM

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



# CHƯƠNG 8. THỰC THI PHẦN MỀM

## 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.

```
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:48627/)
Subscribers:
* /gazebo (http://localhost:40539/)
```

## BEFORE

## AFTER

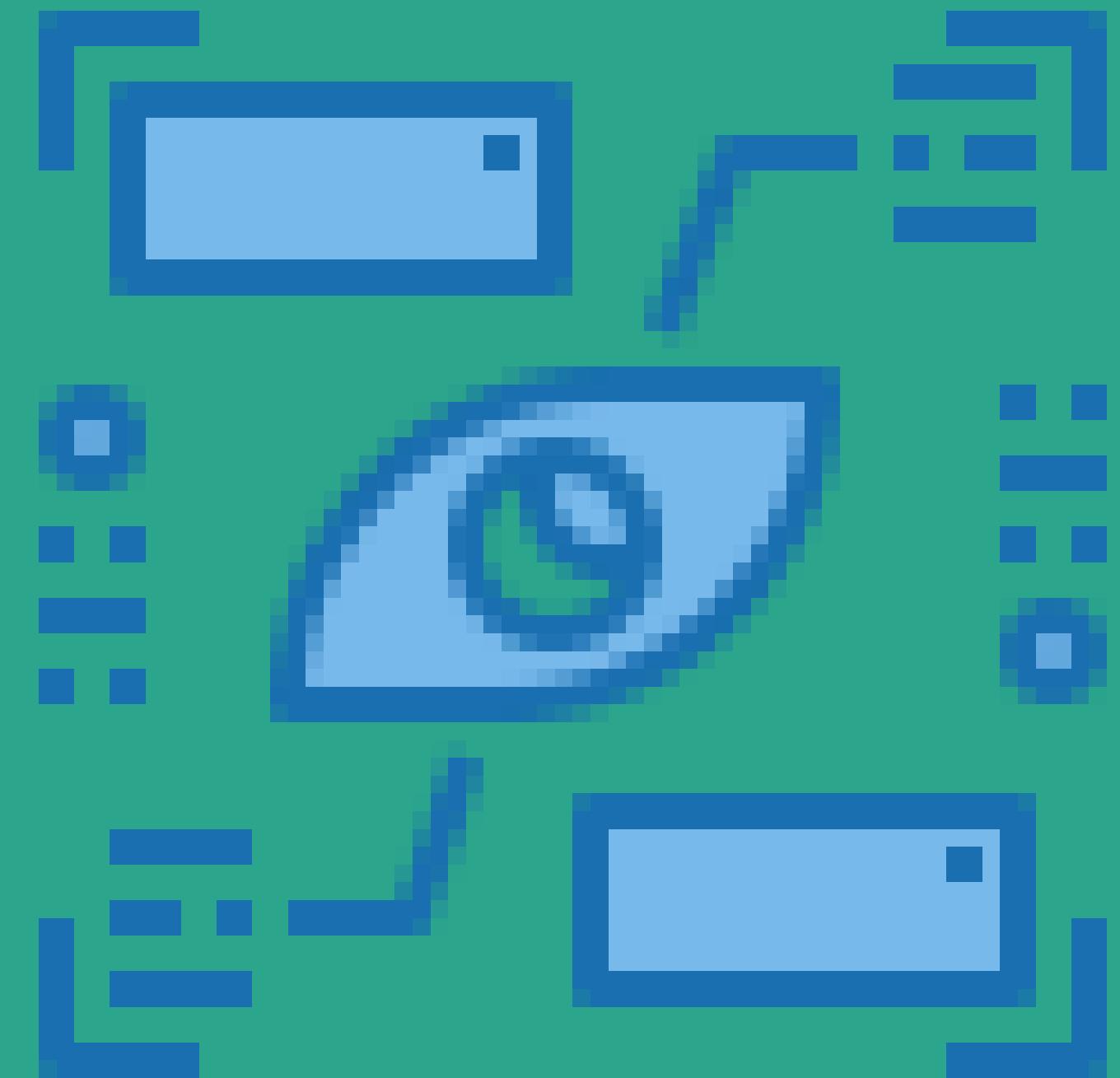
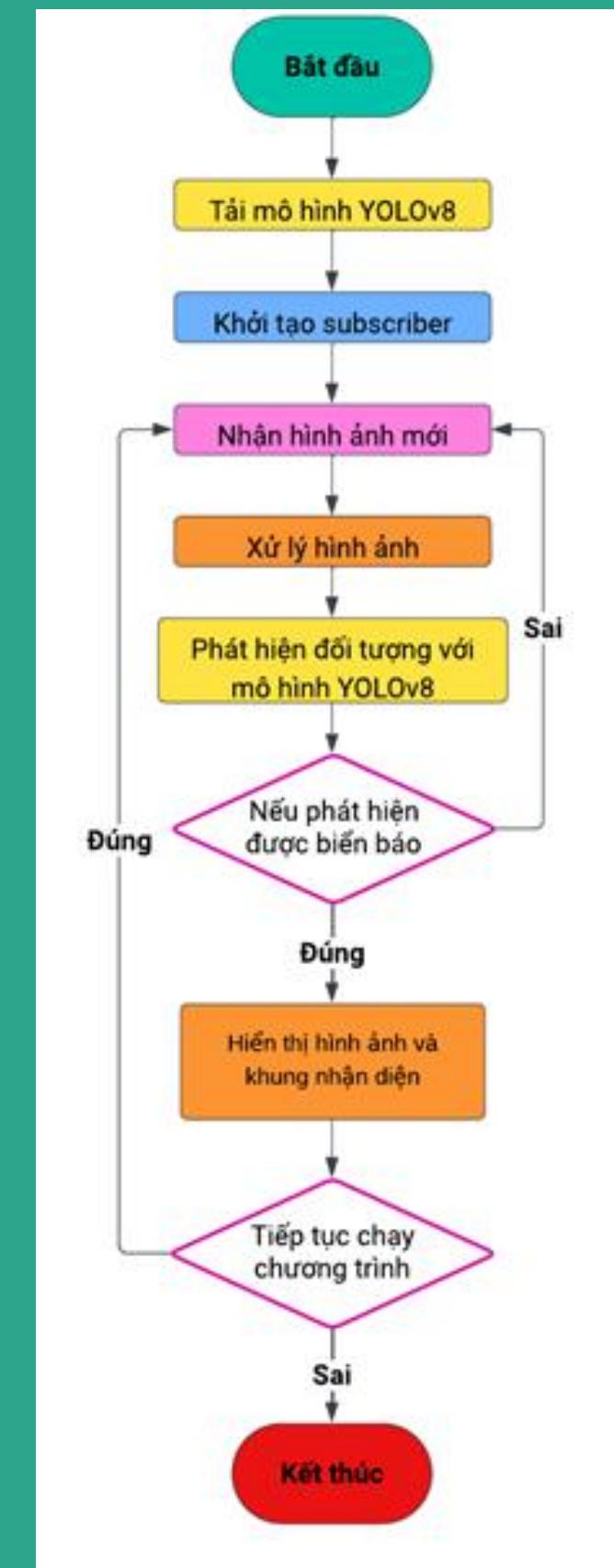


```
thesun@thesun-lap:~/dacn$ rostopic info /camera/rgb/i
mage_raw
Type: sensor_msgs/Image
Publishers:
* /gazebo (http://localhost:40539/)
Subscribers:
* /image_subscriber_6885_1714910861985 (http://local
host:38803/)
```

```
thesun@thesun-lap:~/dacn$ rostopic info /cmd_vel
Type: geometry_msgs/Twist
Publishers:
* /move_base (http://localhost:48627/)
Subscribers:
* /gazebo (http://localhost:40539/)
```

# CHƯƠNG 8. THỰC THI PHẦN MỀM

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



# CHƯƠNG 8. THỰC THI PHẦN MỀM

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



## Mô tả hoạt động

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

```
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

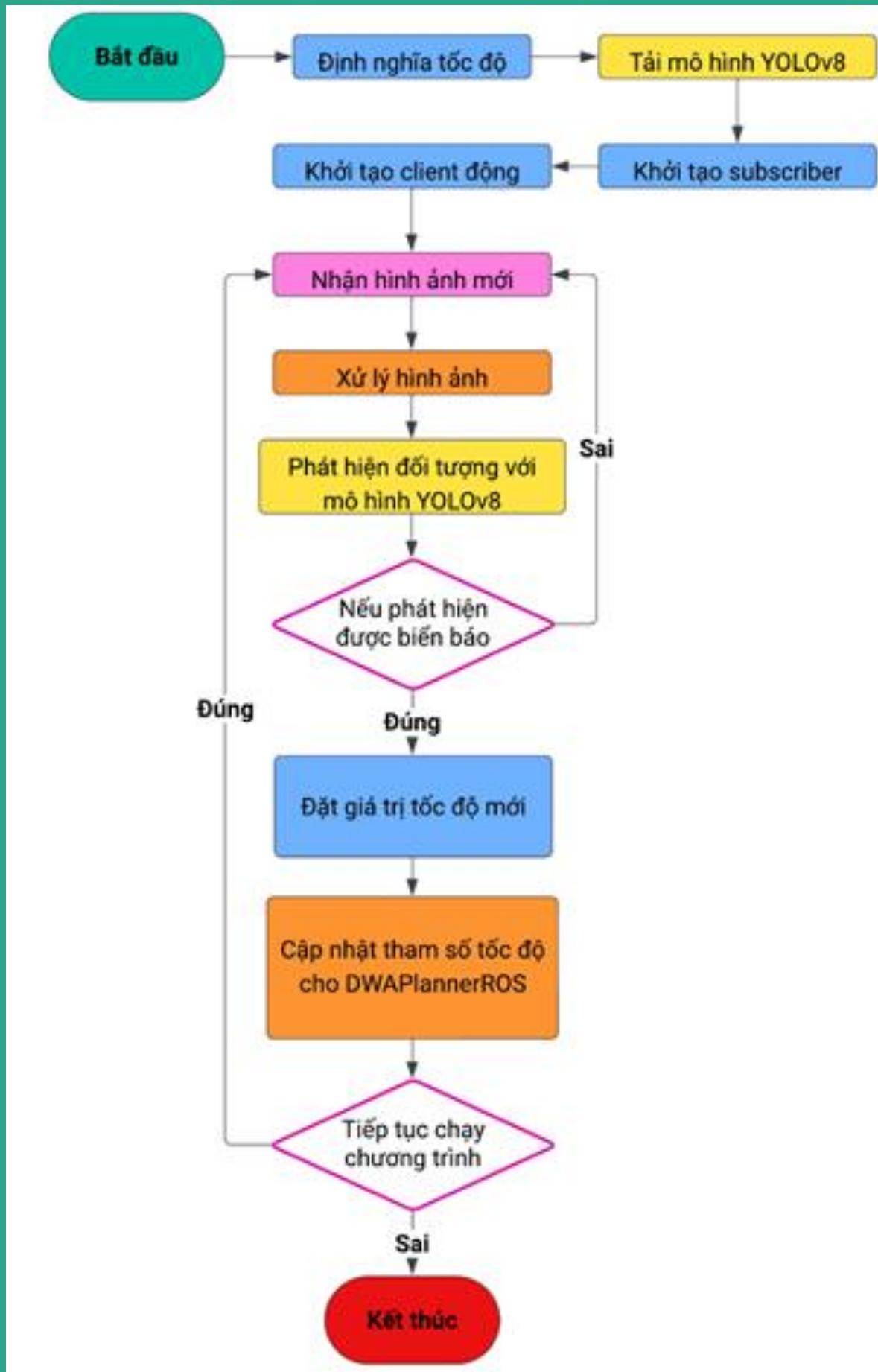
Publishers:
* /move_base (http://localhost:40627/)

Subscribers:
* /gazebo (http://localhost:40539/)

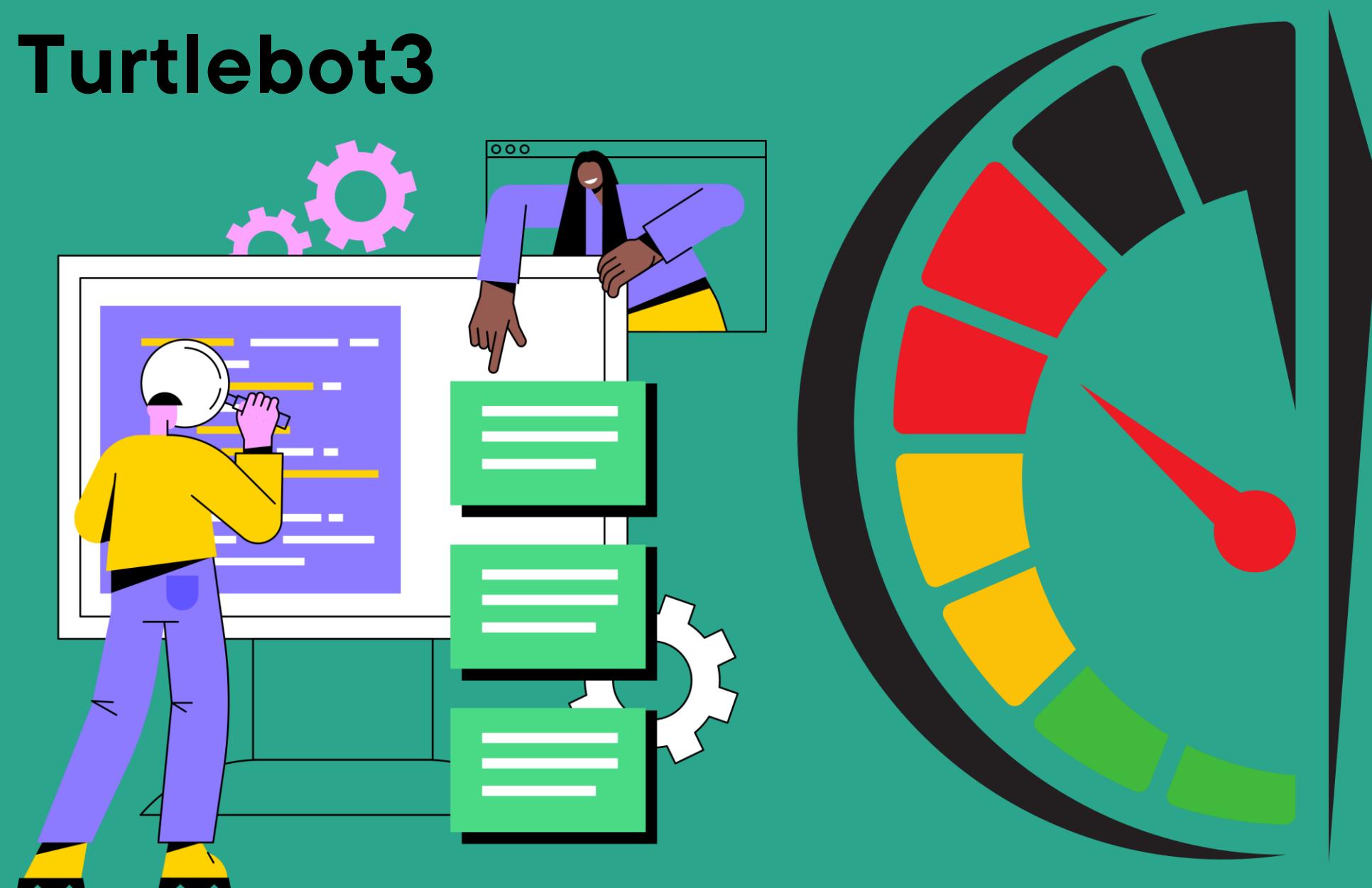
thesun@thesun-lap:~/dacn$
```

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 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 biển báo tốc độ mới (giữ tốc độ trước đó cho đến khi phát hiện biển báo mới).

# CHƯƠNG 8. THỰC THI PHẦN MỀM



Lưu đồ thuật toán chương trình  
phát hiện biển báo tốc độ điều  
khiển tốc độ robot tự hành  
**Turtlebot3**



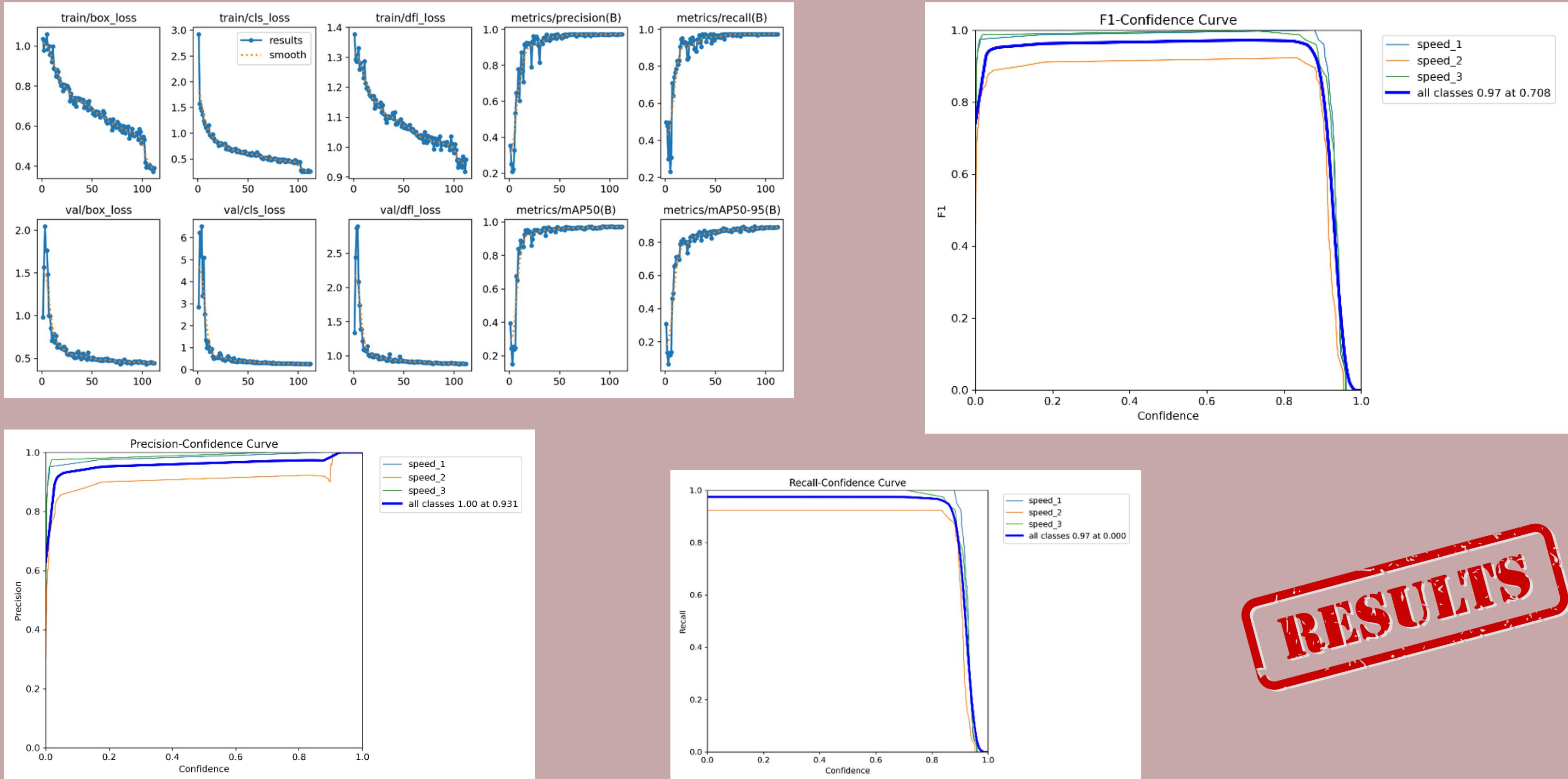
9

# KẾT QUẢ MÔ PHÒNG



# CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG

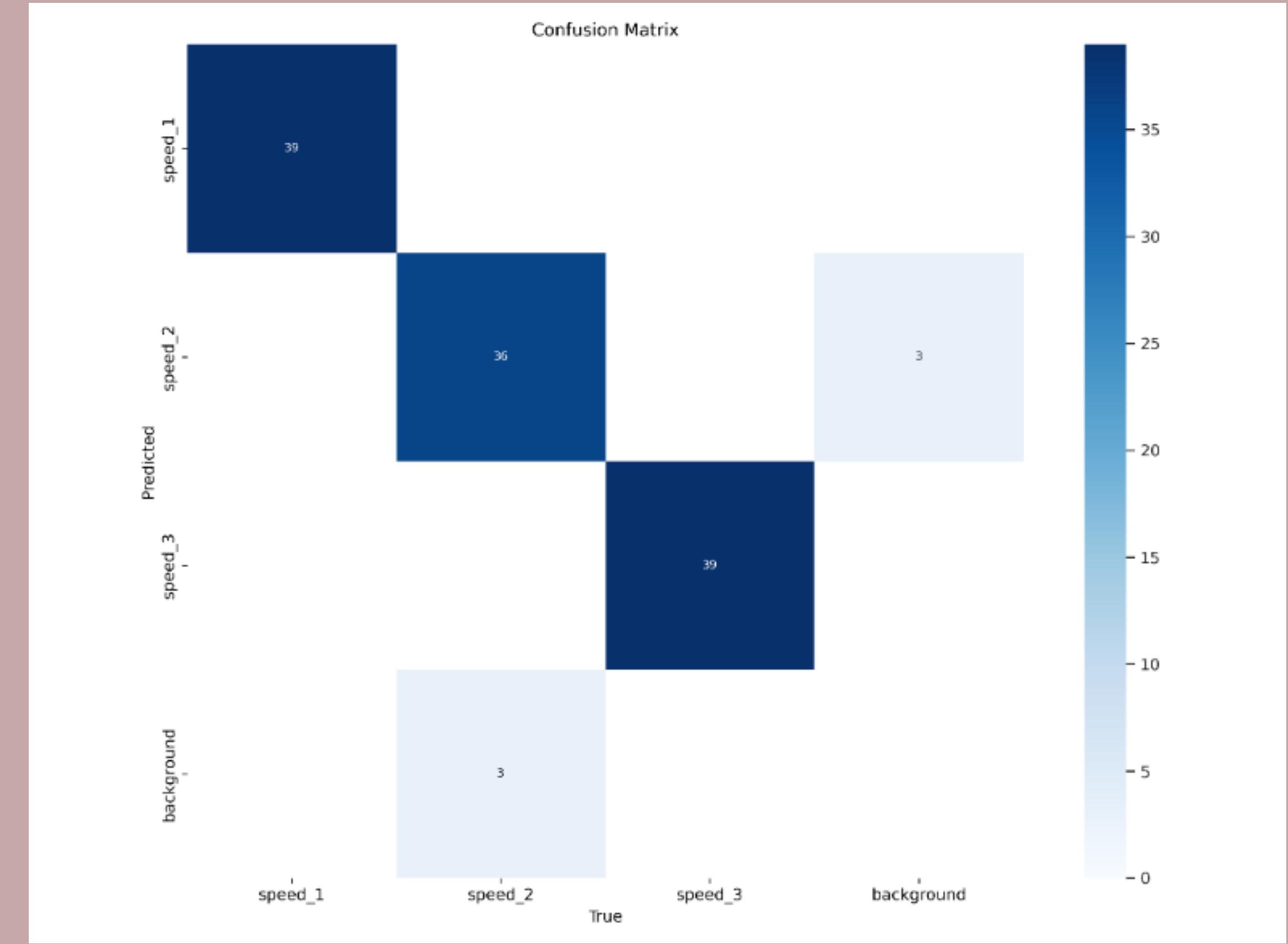
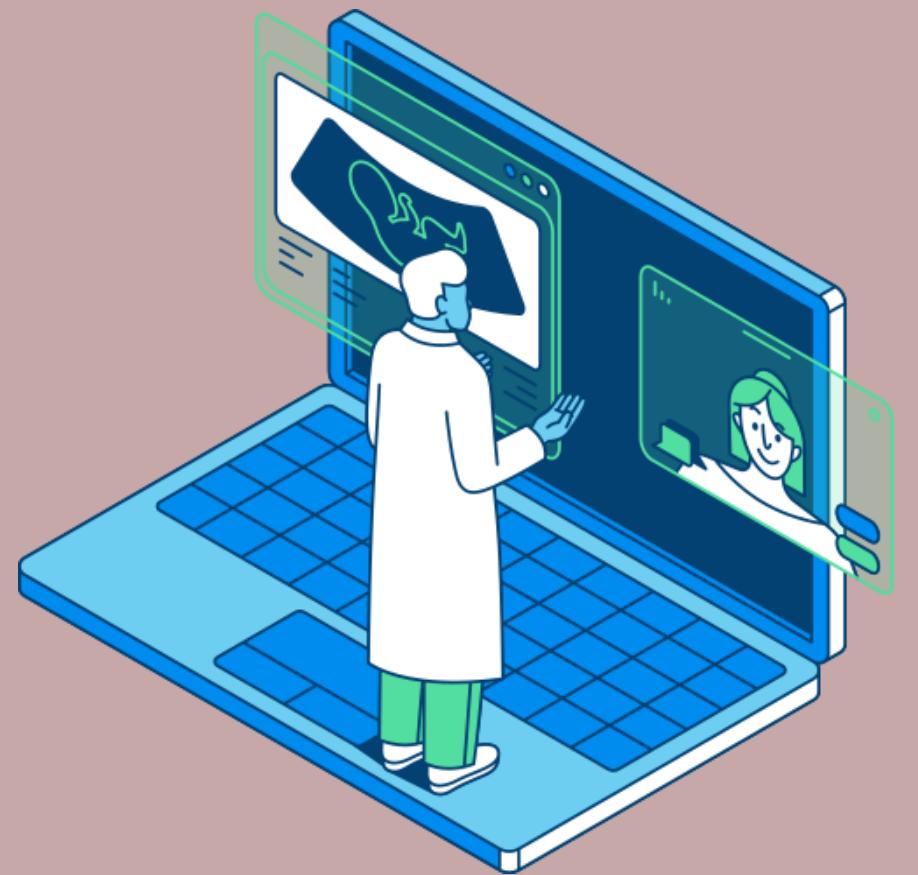
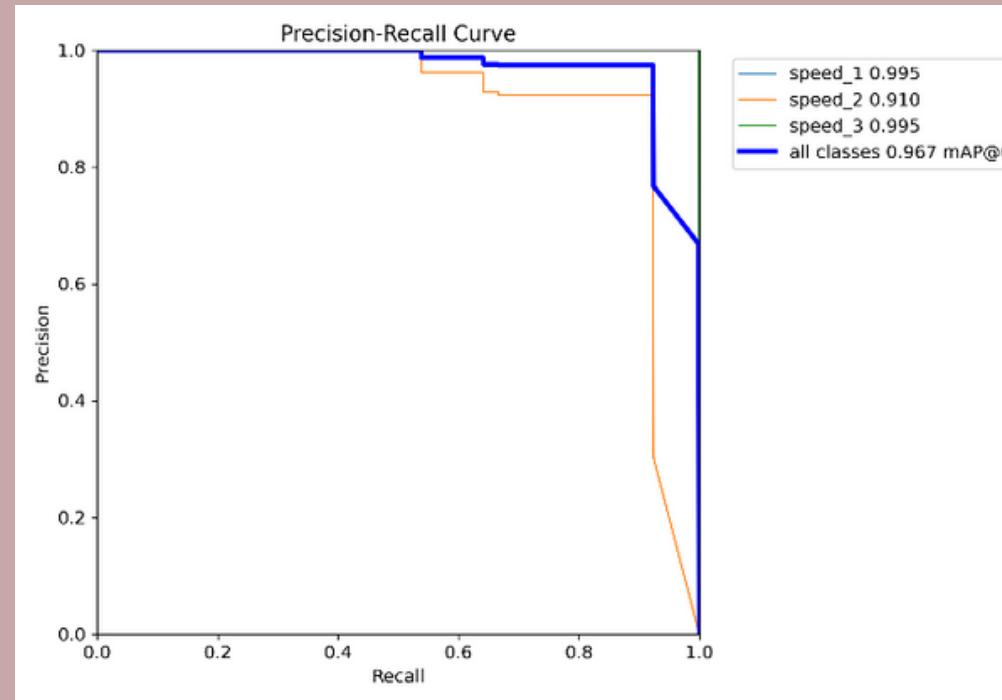
## Mô hình phát hiện biển báo sau khi đào tạo



**RESULTS**

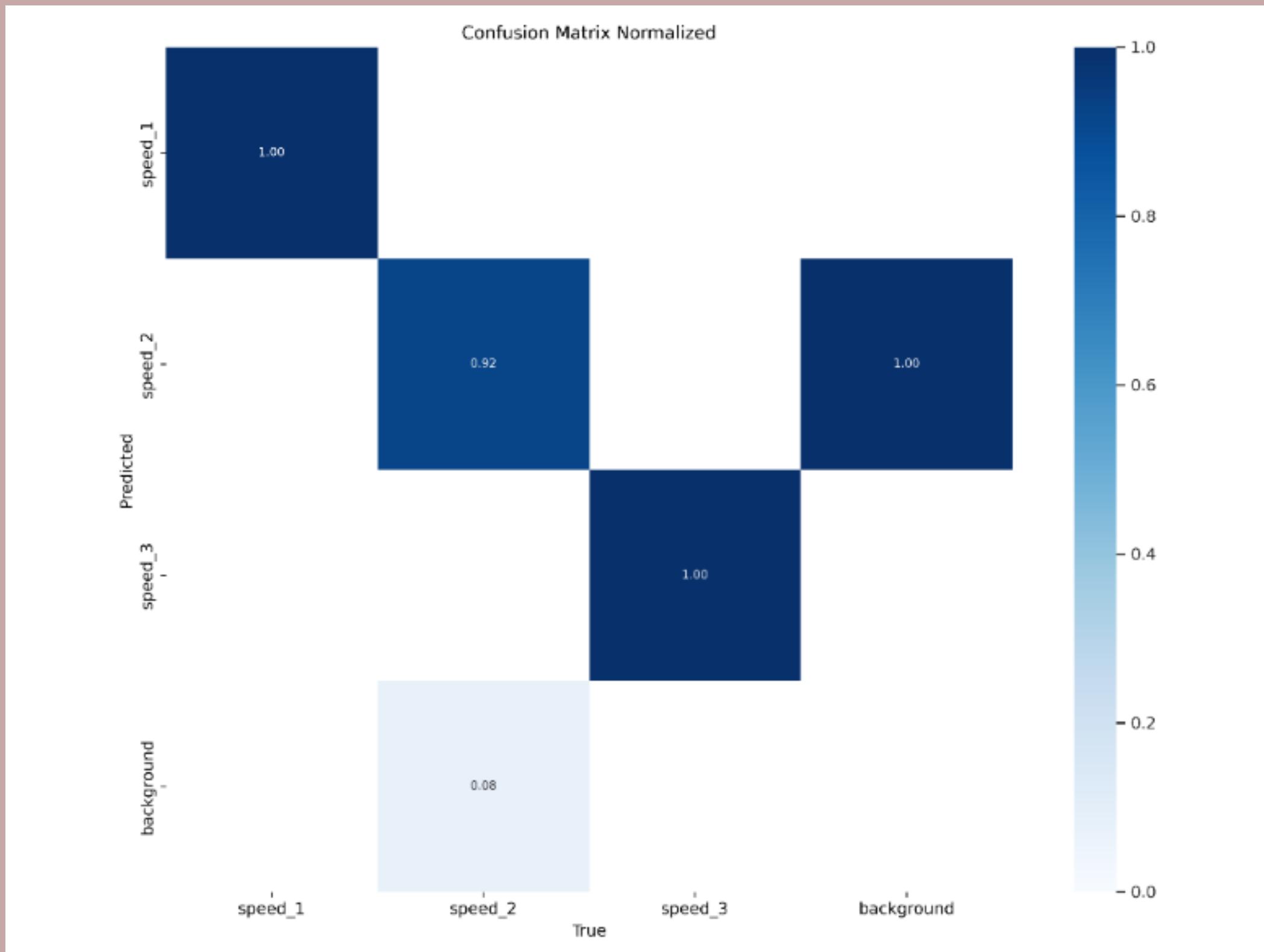
# CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG

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



# CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG

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



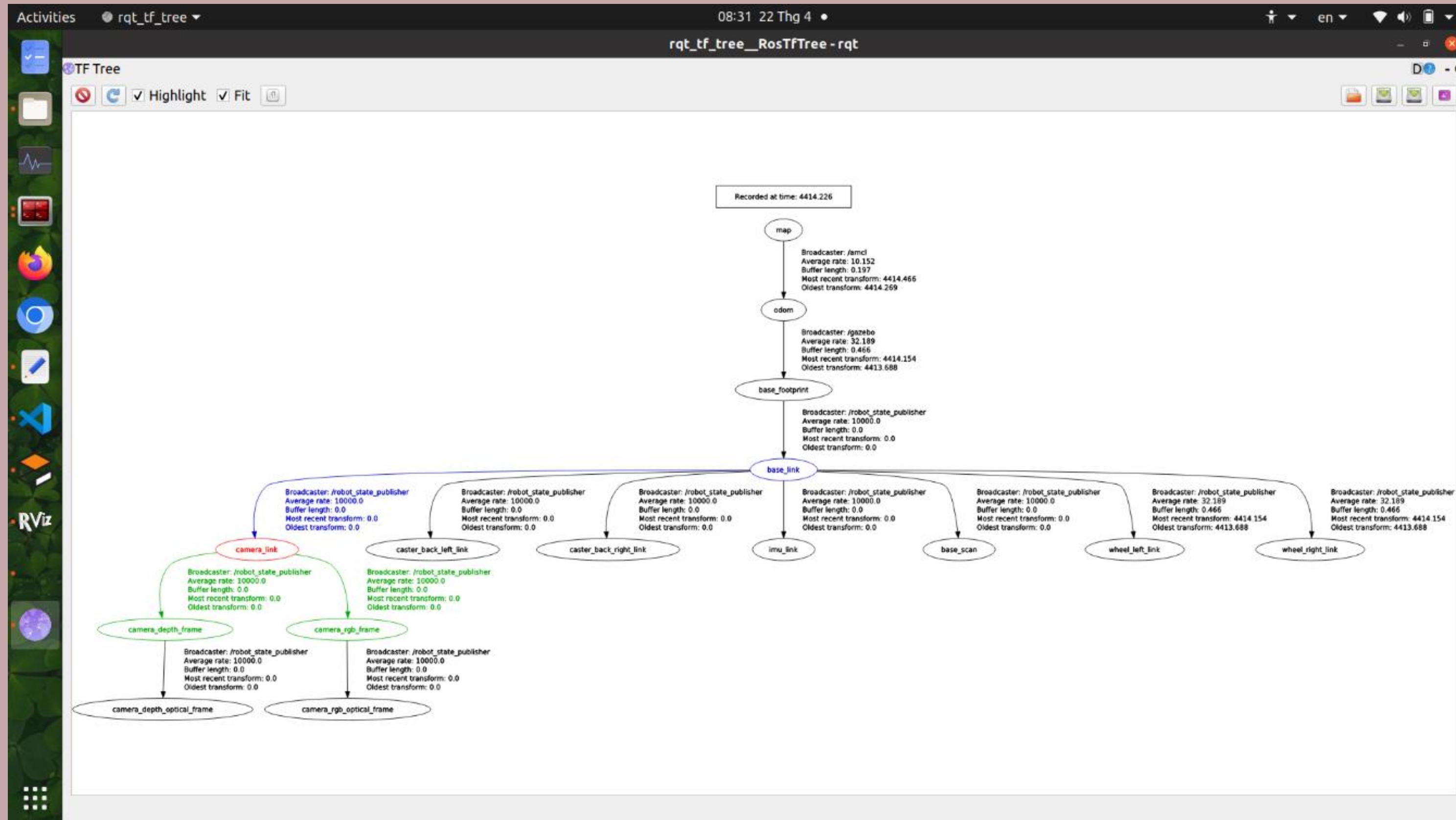
Mô hình phát hiện biển báo sau khi được đào tạo đã cho ra các chỉ số kết quả như sau:

- 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%.



# CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG

## Cây TF hoàn chỉnh của chương trình mô phỏng



# CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG

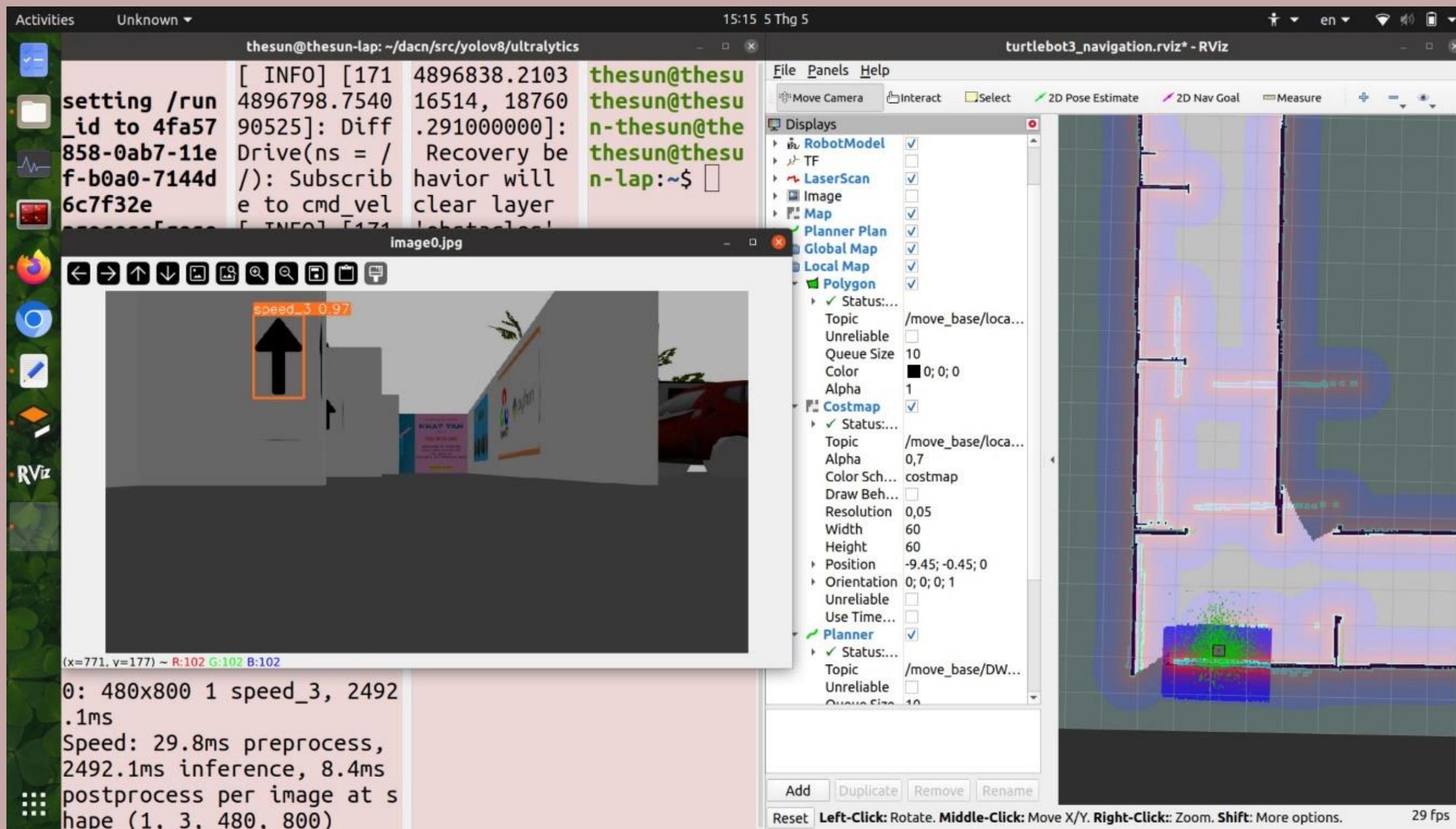
## Các topic được sử dụng

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/compressed/parameter_updates
/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
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
 imu
/initialpose
/joint_states
/map
/map_metadata
/map_updates
/move_base/DWAPlannerROS/cost_cloud
/move_base/DWAPlannerROS/global_plan
/move_base/DWAPlannerROS/local_plan
/move_base/DWAPlannerROS/parameter_descriptions
/move_base/DWAPlannerROS/parameter_updates
/move_base/DWAPlannerROS/trajectory_cloud
/move_base/NavfnROS/plan
/move_base/cancel
/move_base/current_goal
/move_base/feedback
/move_base/global_costmap/costmap
/move_base/global_costmap/costmap_updates
/move_base/global_costmap/footprint
/move_base/global_costmap/inflation_layer/parameter_descriptions
/move_base/global_costmap/inflation_layer/parameter_updates
/move_base/global_costmap/obstacle_layer/parameter_updates
/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
```

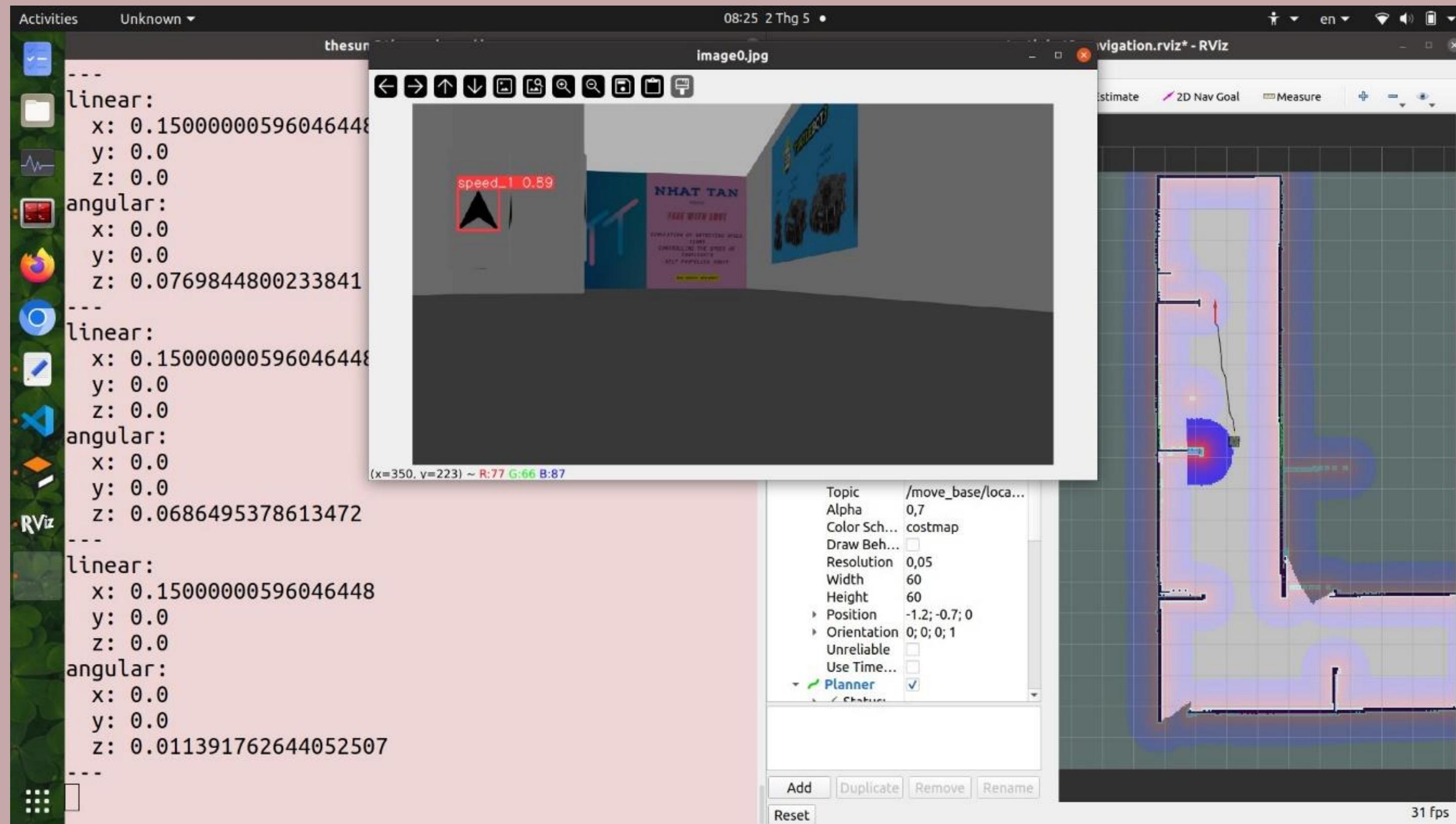
# CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG

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



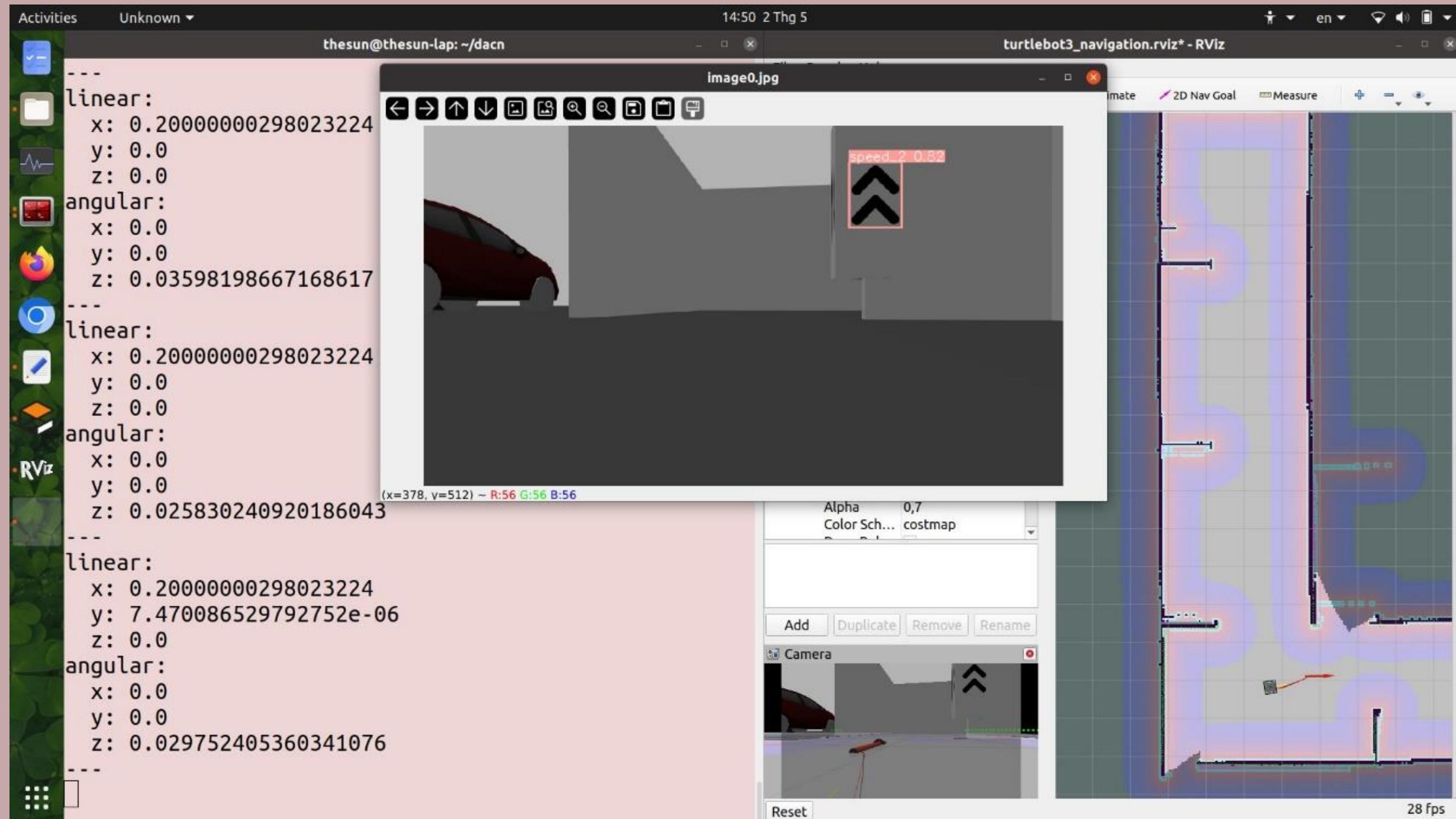
# CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG

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



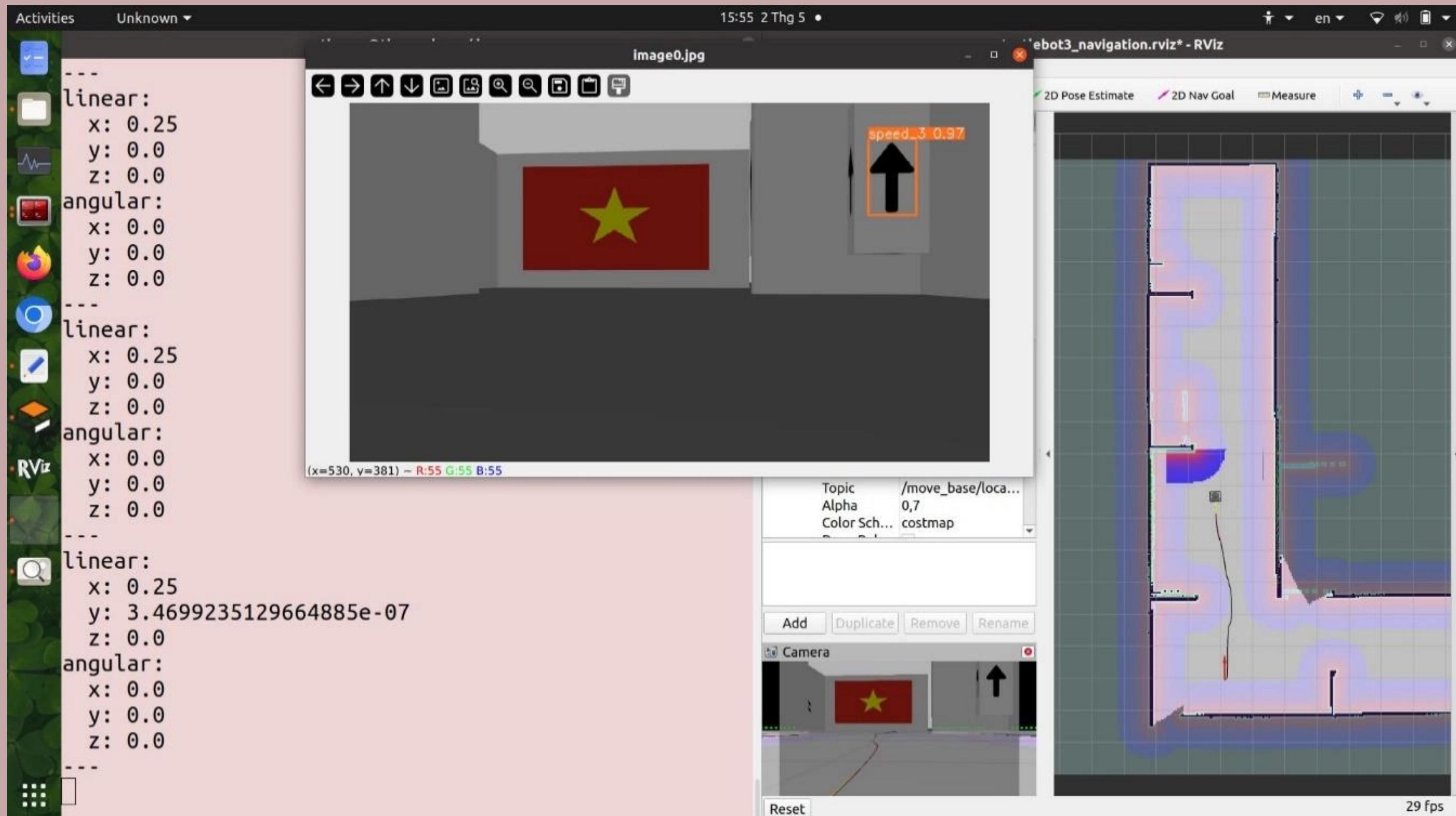
# CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG

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



# CHƯƠNG 9. KẾT QUẢ MÔ PHỎNG

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



10

# KẾT LUẬN



# CHƯƠNG 10. KẾT LUẬN

## 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 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.



# CHƯƠNG 10. KẾT LUẬN



## 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.



# CHƯƠNG 10. KẾT LUẬN

## 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.



# CHƯƠNG 10. KẾT LUẬN

## 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.



# CHƯƠNG 10. KẾT LUẬN



## 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

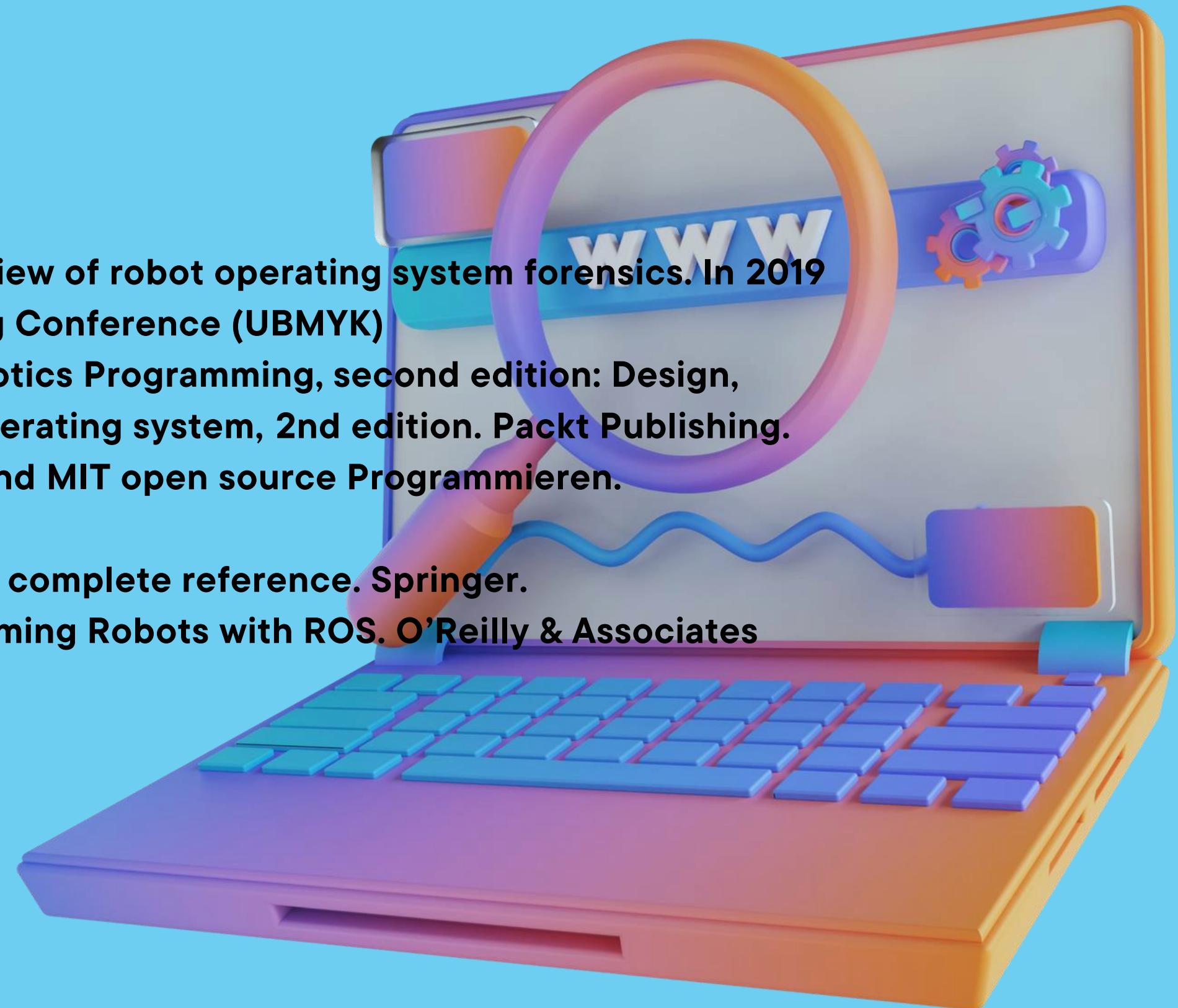
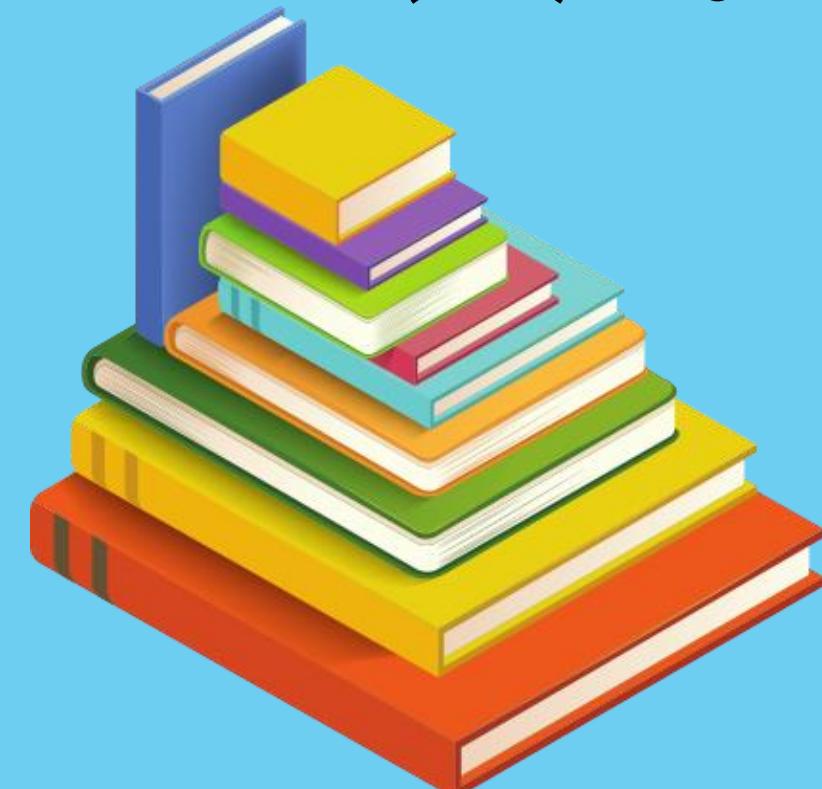
Basheer, M. M., & Varol, A. (2019, November). An overview of robot operating system forensics. In 2019 1st International Informatics and Software Engineering Conference (UBMYK)

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.



# Em xin cảm ơn Thầy đã lắng nghe!

