ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ VIỄN THÔNG
BỘ MÔN MÁY TÍNH – HỆ THỐNG NHÚNG

NGUYỄN GIANG

Đề tài khóa luận tốt nghiệp:

Nghiên cứu và triển khai hệ thống hỗ trợ tránh va chạm phía sau xe dùng cảm biến Lidar

Chuyên ngành Máy Tính - Hệ Thống Nhúng

TP. Hồ Chí Minh, tháng 7 năm 2025

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ - VIỄN THÔNG
BỘ MÔN MÁY TÍNH - HỆ THỐNG NHÚNG

NGUYỄN GIANG
21200284

Đề tài:

NGHIÊN CỨU VÀ TRIỂN KHAI HỆ THỐNG HỖ TRỢ TRÁNH VA CHẠM PHÍA SAU XE DÙNG CẢM BIẾN LIDAR

RESEARCH AND DEPLOYMENT OF A LIDAR-BASED REAR COLLISION AVOIDANCE ASSIST SYSTEM

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN
NGÀNH KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG
CHUYÊN NGÀNH MÁY TÍNH - HỆ THỐNG NHÚNG

NGƯỜI HƯỚNG DẪN KHOA HỌC

ThS. Hoàng Anh Tuấn

TP. Hồ Chí Minh, tháng 7 năm 2025

Lời cảm ơn

Hành trình hoàn thành khóa luận tốt nghiệp là một chặng đường đầy thử thách nhưng cũng vô cùng ý nghĩa, đánh dấu sự khép lại của bốn năm học tập và rèn luyện. Để có được thành quả ngày hôm nay, em đã nhận được sự giúp đỡ và động viên quý báu. Nhân dịp này, em xin được bày tỏ lòng biết ơn chân thành.

Em xin trân trọng cảm ơn thầy ThS. Hoàng Anh Tuấn đã tạo điều kiện và chấp thuận để em có thể thực hiện và phát triển đề tài khóa luận. Sự góp ý của thầy trong quá trình đó đã giúp em hoàn thiện hơn nghiên cứu của mình.

Em cũng xin trân trọng tri ân các thầy cô trong Khoa Điện tử - Viễn thông và bộ môn Máy tính và Hệ thống nhúng. Em xin cảm ơn sự dìu dắt và tận tình của các thầy cô đã giúp đỡ, hỗ trợ trong học tập cho em trong suốt bốn năm qua.

Gửi lời cảm ơn sâu sắc đến Ba Mẹ, người đã nuôi con lớn và trao cho con cơ hội học tập đến bây giờ. Cảm ơn sự hi sinh thầm lặng của Ba Mẹ. Cảm ơn đến gia đình.

Gửi lời cảm ơn đến anh chị khóa trên, các em khóa dưới đã hỗ trợ. Và cảm ơn chân thành đến những người bạn đại học đã giúp đỡ bản thân trong bốn năm qua.

Trân trọng cảm ơn mọi người!

TÓM TẮT KHÓA LUẬN (TIẾNG VIỆT)

Nội dung xoay quanh (2 phần):

- Cách Triển khai mô hình SSD-MobileNet đã được huấn luyện với tập dữ dữ liệu tùy biến để nhận diện vật thể phía sau trên JetRacer
- Mô phỏng hệ thống cảnh báo va chạm phía sau (RCW) hoạt động theo thời gian thực trên JetRacer, sử dụng camera, lidar 2D và xử lý tín hiệu nhận diện để đưa ra cảnh báo.

Khóa luận được chia làm 4 chương:

Chương 1	Trình bày bối cảnh, lý do chọn đề tài, mục tiêu và phạm vi nghiên cứu hệ thống cảnh báo va chạm phía sau.
Chương 2	Tổng hợp kiến thức cơ bản về cảnh báo va chạm, mô hình SSD-MobileNet, dữ liệu PascalVOC, phần cứng và phần mềm sử dụng.
Chương 3	Thiết lập môi trường, thu thập dữ liệu, huấn luyện mô hình, triển khai và tích hợp hệ thống cảnh báo va chạm trên JetRacer.
Chương 4	Trình bày kết quả thực nghiệm, đánh giá hiệu quả hệ thống và đề xuất hướng phát triển trong tương lai.

Những kiến thức và kết quả đã đạt được sau quá trình thực hiện:

- Kiến thức thu được:
    - Hiểu rõ về hệ thống cảnh báo va chạm phía sau trong các ứng dụng robot tự hành.
    - Hiểu cơ bản kiến thức về nhận dạng vật thể bằng học sâu, đặc biệt là mô hình SSD-MobileNet.
    - Hiểu quy trình xử lý dữ liệu theo định dạng PascalVOC, từ thu thập, gán nhãn đến huấn luyện.
    - Biết các sử dụng các ROS package, node, topic, và message trong hệ thống.
- Kết quả đạt được:

- Xây dựng và triển khai thành công hệ thống cảnh báo va chạm phía sau hoạt động theo thời gian thực trên JetRacer.
- Huấn luyện mô hình SSD-MobileNet đạt độ chính xác ổn định trong việc phát hiện vật thể ở vùng phía sau xe.
- Hệ thống được kiểm nghiệm thực tế, cho kết quả khả quan và có thể mở rộng cho các ứng dụng thực tiễn khác.

TÓM TẮT KHÓA LUẬN (TIẾNG ANH)

THE ABSTRACT OF THE GRADUATION THESIS

Content Overview (Two Main Parts):

- How to deploy a custom-trained SSD-MobileNet model for rear-view object detection on the JetRacer.
- Simulation of a real-time Rear Collision Warning (RCW) system on JetRacer, using camera and 2D LiDAR fusion, plus signal processing to generate alerts

Thesis Structure (4 Chapters):

Chapter 1	Presents the background, the rationale for selecting the topic, the objectives, and the scope of the research on the rear collision warning system
Chapter 2	Provides an overview of fundamental knowledge related to collision warning, the SSD-MobileNet model, Pascal VOC dataset, and the hardware and software used
Chapter 3	describes the environment setup, data collection, model training, and the deployment and integration of the collision warning system on JetRacer
Chapter 4	presents the experimental results, evaluates the system's performance, and proposes future development directions.

Knowledge and results achieved following the implementation process:

- Knowledge gained:
    - Gained a clear understanding of rear collision warning systems in autonomous robotic applications.
    - Acquired fundamental knowledge of deep learning-based object detection, specifically the SSD-MobileNet model.
    - Learned the entire data processing pipeline in Pascal VOC format, from collection and annotation to model training.
    - Became proficient in using ROS packages, nodes, topics, and message types within the system.
- Results achieved:

- Successfully built and deployed a real-time rear collision warning system on JetRacer.
- Trained the SSD-MobileNet model to achieve stable accuracy in detecting objects in the vehicle’s rear zone.
- Conducted practical testing of the system, obtaining promising results with potential for extension to other real-world applications.

MỤC LỤC

DANH SÁCH VIẾT CHỮ TẮT

Từ viết tắt	Ý nghĩa đầy đủ
RCW	Rear Collision Warning (Cảnh báo va chạm phía sau)
ADAS	Advanced driver-assistance system (Hệ thống hỗ trợ lái xe tiên tiến)
ROS	The Robot Operating System (Hệ điều hành robot)
CUDA	Compute Unified Device Architecture (Kiến trúc thiết bị tính toán hợp nhất)
cuDNN	CUDA Deep Neural Network (Thư viện nguyên thủy được tăng tốc bằng GPU dành cho mạng nơ-ron sâu)
TensorRT	Một bộ công cụ phần mềm cho suy luận học sâu hiệu suất cao trên GPU NVIDIA
SSD	Single shot multibox detector (một phương pháp học sâu để phát hiện đối tượng, được biết đến với tốc độ và độ chính xác)
TTC	Time to collision (thời gian trước khi xảy ra va chạm giữa các phương tiện liên quan-vật thể-chủ thể)
CNN	Convolutional Neural Network (mô hình Deep Learning tiên tiến)
TP	True Positive (Là số lượng mẫu mà mô hình dự đoán đúng là “có”, và thực tế nó đúng là “có”)
FN	False Negative (Là số lượng mẫu mà mô hình dự đoán là “không có”, nhưng thực tế lại có)
FP	False Positive (Là số lượng mẫu mà mô hình dự đoán là “có”, nhưng thực tế lại không có)

DANH SÁCH HÌNH VÀ BẢNG

CHƯƠNG 1: GIỚI THIỆU VÀ ĐẶT VẤN ĐỀ

1.1. Bối cảnh và tầm quan trọng của vấn đề

Dựa trên thông tin https://congan.quangninh.gov.vn, thông tin liên quan đến thống kê tai nạn giao thông do va chạm phía sau trên đường cao tốc trong 7 tháng đầu năm 2024:

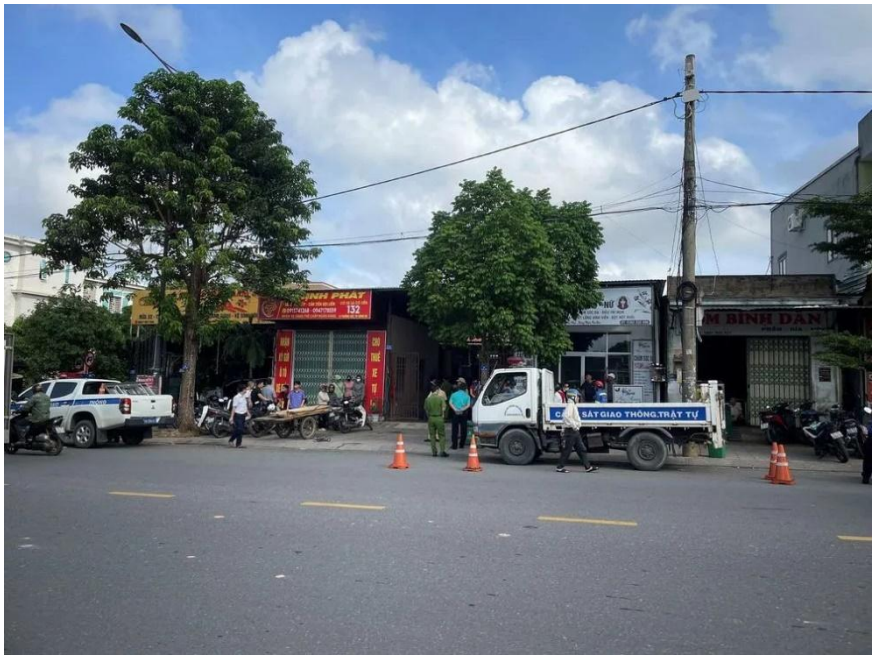
- Tổng số vụ TNGT trên cao tốc: 112 vụ.
- Nguyên nhân liên quan đến va chạm phía sau:
- Không giữ khoảng cách an toàn với xe chạy trước liền kề: 5,3% số vụ.
- Thời gian xảy ra tai nạn: 57,14% số vụ xảy ra từ 18h hôm trước đến 6h sáng hôm sau.
- Các nguyên nhân khác như không chú ý quan sát (22,32%) cũng có thể góp phần vào các vụ va chạm phía sau.

Hình 1.1: Tai nạn giao thông – nguồn: Tai nạn liên hoàn ở Hà Nội, xe con biến dạng – Báo Lao động

Việc lùi xe, một thao tác tưởng chừng đơn giản và diễn ra hàng ngày, lại ẩn chứa những rủi ro chết người và đang trở thành một vấn đề an toàn giao thông nghiêm trọng tại Việt Nam cũng như trên toàn thế giới.

Hình 1.2: Tai nạn khi lùi xe – nguồn: Tài xế lùi xe không quan sát khiến 1 nữ công nhân tử vong – Góc nhìn pháp lý

Các bản tin thời sự thường xuyên ghi nhận những vụ tai nạn thương tâm mà nguyên nhân xuất phát từ việc lùi xe thiếu quan sát. Điển hình là các vụ việc như tài xế điều khiển xe bán tải lùi với tốc độ cao qua đường, tông sập cổng và gây tử vong cho chủ nhà, hay một nữ công nhân đang dừng xe nghe điện thoại bị ô tô lùi cán tử vong tại chỗ. Những sự việc này không chỉ là lời cảnh tỉnh mà còn cho thấy hậu quả thảm khốc của việc thiếu cẩn trọng trong một thao tác mà nhiều người lái xe xem nhẹ.



Hình 1.3: Tai nạn khi lùi xe – nguồn: tài xế lùi xe đâm tử vong chủ nhà - kinhtedothi.vn

Lời khai của các tài xế trong các vụ tai nạn càng củng cố thêm nhận định này. Hầu hết đều thừa nhận nguyên nhân chính là do "thiếu quan sát". Trong vụ tai nạn tại sân

trường ở Đắk Lắk, tài xế khai rằng sau khi đưa con vào lớp, anh ra xe và lùi về. Dù có nhìn gương chiếu hậu hai bên, anh đã bỏ qua gương chiếu hậu trong xe và không bật đèn cảnh báo, dẫn đến việc tông vào ba học sinh phía sau, làm một em tử vong. Lời khai này cho thấy sự thiếu sót nghiêm trọng trong cả kỹ năng lẫn quy trình an toàn cơ bản khi thực hiện thao tác lùi xe.

Hình 1.2: Tai nạn khi tài xe chủ quan và thiếu quan sát - Báo Pháp Luật TP. Hồ Chí Minh

Ta có thể thấy sự cần thiết của các hệ thống hỗ trợ lái xe tiên tiến (ADAS), đặc biệt là các tính năng cảnh báo va chạm phía sau (Rear Collision Warning - RCW). Hệ thống RCW có thể giúp:

- Tăng cường an toàn: Hệ thống Cảnh báo Va chạm Phía sau (RCW) hoạt động như một người bảo vệ luôn cảnh giác, liên tục theo dõi khu vực phía sau xe. Nó cung cấp một lớp bảo vệ bổ sung chống lại các vụ va chạm từ phía sau.
- Hệ thống cảnh báo Sớm: Bằng cách phát hiện các vật thể đang tiến đến, hệ thống cung cấp các cảnh báo kịp thời, giúp người điều khiển có thêm thời gian để phản ứng và có khả năng tránh được va chạm.
- Nâng cao nhận thức trong điều kiện khó khăn: Hệ thống có thể hoạt động hiệu quả ngay cả trong điều kiện ánh sáng yếu hoặc khi tầm nhìn bị hạn chế, nâng cao nhận thức về môi trường xung quanh.
- Giảm tải cho người lái, giảm khối lượng công việc: Bằng cách tự động hóa nhiệm vụ theo dõi phía sau, hệ thống giúp giảm khối lượng công việc của người điều khiển, cho phép họ tập trung vào các nhiệm vụ quan trọng khác, đặc biệt là trên đường cao tốc hoặc trong điều kiện giao thông đông đúc.

1.2. Phân tích vấn đề nghiên cứu

Thách thức ban đầu của đề tài

Đề tài gốc: "Nghiên cứu và triển khai rear collision warning sử dụng LiDAR"

Vấn đề phát sinh:

- Giới hạn phần cứng: LiDAR trên JetRacer là loại 2D (không phải 3D)
- Hạn chế về thông tin không gian: không có thông tin chiều cao của vật thể
- Khả năng phát hiện đối tượng hạn chế:
    - Không phát hiện được objects bay/nhảy
    - Bị che khuất bởi vật thể thấp

Định hướng khắc phục

Thay vì sử dụng LiDAR 3D (chi phí cao, phức tạp), đề tài đề xuất kết hợp camera với LiDAR 2D để xây dựng hệ thống rear collision warning. Việc tích hợp camera giúp bổ sung thông tin về chiều cao và hình ảnh đối tượng, khắc phục hạn chế của LiDAR 2D trong việc nhận diện các vật thể ở nhiều độ cao khác nhau. Hệ thống kết hợp này vừa đảm bảo hiệu quả phát hiện va chạm, vừa tối ưu chi phí và phù hợp với điều kiện phần cứng hiện có.

1.3. Mục tiêu và phạm vi nghiên cứu

1.3.1. Ý nghĩa khoa học

Nghiên cứu này mang lại giá trị khoa học quan trọng bằng cách khám phá và nâng cao hiệu quả của công nghệ cảm biến Lidar 2D trong lĩnh vực an toàn giao thông và tự động hóa. Việc tập trung vào việc xử lý dữ liệu quét 2D từ Lidar giúp phát triển các thuật toán phát hiện va chạm chính xác hơn, đồng thời tối ưu hóa phương pháp phân tích tín hiệu trong môi trường thực tế. Kết quả nghiên cứu đóng góp vào việc mở rộng kiến thức về ứng dụng cảm biến 2D trong hệ thống hỗ trợ lái xe (ADAS), đặc biệt trong việc cải tiến nhận diện vật cản và xử lý dữ liệu không gian hạn chế. Ngoài ra, đề tài còn tạo nền tảng cho các nghiên cứu sâu hơn về tích hợp đa cảm biến và trí tuệ nhân tạo, góp phần thúc đẩy sự phát triển của công nghệ giao thông thông minh trên toàn cầu.

1.3.2. Ý nghĩa thực tiễn

Về mặt thực tiễn, hệ thống hỗ trợ tránh va chạm phía sau xe dựa trên Lidar 2D mang lại lợi ích thiết thực trong việc nâng cao an toàn giao thông. Công nghệ này cho phép phát hiện sớm các chướng ngại vật phía sau xe, giảm thiểu rủi ro tai nạn do va

chạm – một nguyên nhân phổ biến gây thiệt hại về người và tài sản. Với chi phí thấp hơn so với Lidar 3D, hệ thống có thể được triển khai rộng rãi trên các loại xe phổ thông, từ xe cá nhân đến phương tiện công cộng, giúp bảo vệ người lái và hành khách trong nhiều điều kiện giao thông. Ngoài ra, việc ứng dụng này còn hỗ trợ giảm chi phí bảo trì và bảo hiểm liên quan đến tai nạn, đồng thời tạo điều kiện cho các nhà sản xuất phát triển xe thông minh với giá thành hợp lý. Trong dài hạn, nghiên cứu góp phần xây dựng các đô thị an toàn và hiệu quả hơn.

1.3.3. Mục tiêu đề tài

Đề tài này hướng đến việc nghiên cứu và phát triển một hệ thống cảnh báo va chạm phía sau (Rear Collision Warning – RCW) thông qua việc kết hợp dữ liệu từ cảm biến LiDAR 2D và Camera. Hệ thống được thiết kế với khả năng xử lý theo thời gian thực (real-time processing) trên nền tảng JetRacer, nhằm nâng cao độ chính xác trong việc phát hiện và cảnh báo nguy cơ va chạm từ phía sau. Đề tài tập trung vào việc khắc phục những hạn chế của việc sử dụng đơn lẻ cảm biến LiDAR 2D hoặc Camera, đồng thời tận dụng ưu điểm của cả hai loại cảm biến này thông qua kỹ thuật tích hợp dữ liệu cảm biến.

Nghiên cứu và phát triển thuật toán kết hợp dữ liệu LiDAR 2D và Camera

Phân tích các phương pháp tích hợp thông tin về khoảng cách từ LiDAR và đặc tính hình ảnh từ Camera. Đánh giá ưu – nhược điểm của từng cách tiếp cận và chọn ra giải pháp tối ưu nhất cho bài toán cảnh báo va chạm phía sau.

Xây dựng bộ dữ liệu và tinh chỉnh (fine-tuning) mô hình SSD MobileNet v2

Thu thập và chuẩn hóa dataset đặc thù cho kịch bản va chạm phía sau. Áp dụng cách thức tinh chỉnh mô hình SSD MobileNet v2 để cải thiện khả năng nhận dạng đối tượng trong nhiều điều kiện ánh sáng, môi trường khác nhau, đồng thời tối ưu giữa độ chính xác và tốc độ xử lý nhằm đáp ứng yêu cầu real-time.

1.3.4. Phạm vi nghiên cứu

Nghiên cứu này giới hạn trong phạm vi phát triển hệ thống cảnh báo va chạm phía sau trên nền tảng JetRacer, tập trung vào việc kết hợp dữ liệu từ LiDAR 2D và Camera. Cụ thể:

- Về phần cứng:
    - LiDAR 2D và Camera tích hợp sẵn trên JetRacer.

    - Sử dụng thêm máy ảo Ubuntu (VM) chạy trên PC để triển khai và thử nghiệm thuật toán RCW, kết nối với JetRacer qua mạng nội bộ hoặc USB tethering.
- Về môi trường: Thử nghiệm trong môi trường có kiểm soát, với các kịch bản giả lập các tình huống va chạm phía sau.
- Về phát hiện đối tượng: Tập trung vào các đối tượng chính có thể gây va chạm như xe, người và vật cản.
- Về tính năng:
    - Chỉ phát triển chức năng cảnh báo (hiển thị hình ảnh) khi phát hiện nguy cơ va chạm phía sau.
    - Sử dụng ROS để duy trì việc xuất lệnh cmd_vel điều khiển bánh lái như phanh hoặc lái.
- Về xử lý: Đảm bảo thuật toán RCW chạy real-time trên tài nguyên giới hạn của JetRacer và VM Ubuntu (CPU/GPU).

Nghiên cứu không bao gồm việc phát triển các tính năng ADAS khác như giữ làn đường, kiểm soát hành trình thích ứng hay dự đoán hướng đi chính xác cho RCW.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VÀ NGHIÊN CỨU LIÊN QUAN

2.1. Rear collision warning

ADAS là viết tắt của Advanced Driver Assistance Systems – tức là hệ thống hỗ trợ lái xe nâng cao. Đây là tập hợp các công nghệ điện tử thông minh được tích hợp vào xe hơi nhằm giúp người lái điều khiển xe an toàn hơn, giảm thiểu tai nạn và nâng cao trải nghiệm lái.

Rear Collision Warning (RCW) là một tính năng thuộc hệ thống hỗ trợ lái xe nâng cao (ADAS – Advanced Driver Assistance Systems) giúp cảnh báo nguy cơ va chạm từ phía sau xe. Đây là công nghệ ngày càng phổ biến trong các dòng xe hiện đại nhằm tăng cường an toàn giao thông

Các hoạt động của RCW:

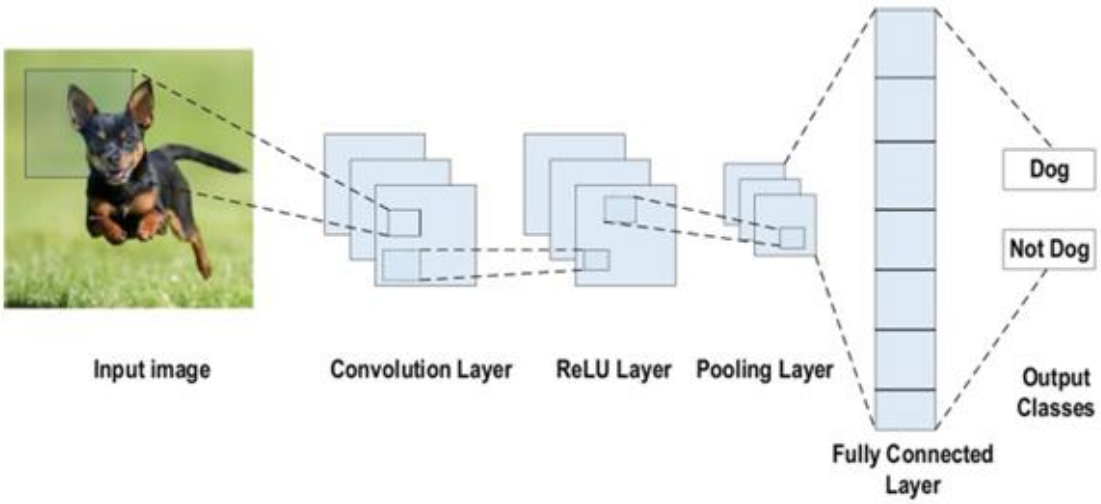
- Hệ thống sử dụng cảm biến radar, ultrasonic, camera hoặc LIDAR để giám sát khu vực phía sau xe.
- Khi phát hiện một phương tiện hoặc vật thể tiếp cận với tốc độ nguy hiểm, RCW sẽ:
    - Phát cảnh báo bằng âm thanh, hình ảnh hoặc rung vô-lăng.
    - Một số xe còn tích hợp với hệ thống phanh khẩn cấp phía sau (Rear AEB) để tự động phanh nếu cần.

2.2. Object detection và deep learning

Trong lĩnh vực Object Detection và Deep Learning, Convolutional Neural Network (CNN) là một mô hình quan trọng. CNN giúp máy tính nhận diện và định vị đối tượng trong ảnh, ví dụ như xác định xe hay người và vẽ hộp bao quanh chúng (bounding box). CNN hoạt động bằng cách quét ảnh để tìm các đặc điểm như cạnh, hình dạng, sau đó phân loại và định vị đối tượng.

CNN đóng vai trò then chốt trong Object Detection nhờ khả năng tự động trích xuất các đặc trưng quan trọng từ hình ảnh, như cạnh, góc, hình dạng... Thay vì phải thiết kế thủ công các đặc trưng, CNN học trực tiếp từ dữ liệu, giúp mô hình nhận diện và định vị đối tượng chính xác hơn. Ngoài ra, CNN còn giúp giảm số lượng tham số so với mạng truyền thống, tăng hiệu quả tính toán và khả năng tổng quát hóa.

.



Hình 2.1: Kiến trúc mạng nơ-ron tích chập phân loại ảnh cơ bản

nguồn: Review of deep learning: concepts, CNN architectures, challenges, applications, future directions

Object Detection (phát hiện đối tượng) là một bài toán quan trọng trong Computer Vision, có nhiệm vụ vừa xác định loại đối tượng vừa định vị chính xác vị trí của chúng trong hình ảnh. Khác với bài toán phân loại ảnh (Image Classification) chỉ xác định nội dung của toàn bộ hình ảnh, Object Detection phải xác định cả loại đối tượng và vị trí chính xác của từng đối tượng.

Cụ thể, Object Detection kết hợp hai nhiệm vụ con:

- Classification (Phân loại): Xác định đối tượng thuộc lớp nào (xe ô tô, người đi bộ, xe máy...)
- Localization (Định vị): Xác định vị trí chính xác của đối tượng thông qua bounding box

Các phương pháp Object Detection phổ biến:

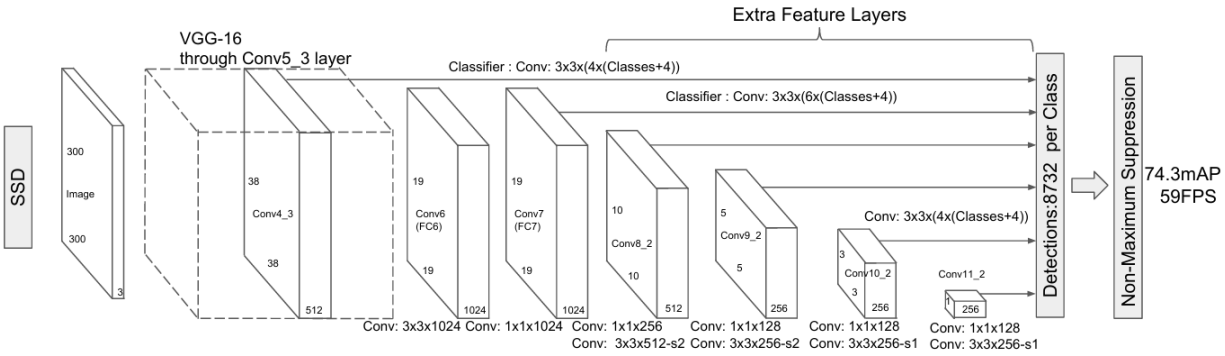
- YOLO (You Only Look Once): Phát hiện đối tượng rất nhanh, phù hợp với các ứng dụng thời gian thực.
- SSD (Single Shot MultiBox Detector): Kết hợp tốc độ nhanh và độ chính xác cao.
- Faster R-CNN: Độ chính xác cao, thường dùng trong các bài toán yêu cầu chất lượng phát hiện tốt.

2.3. Mô hình SSD-mobilenet

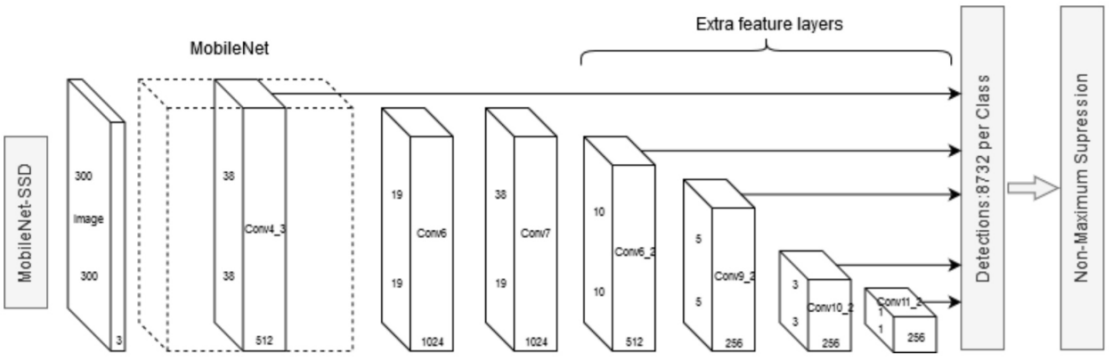
SSD-MobileNet là một biến thể của mô hình SSD, trong đó phần mạng cơ sở (backbone) ban đầu là VGG-16 đã được thay thế bằng MobileNet – một mạng CNN nhẹ và tối ưu cho các thiết bị nhúng. Về bản chất, mô hình vẫn giữ nguyên cấu trúc

của SSD: ảnh đầu vào được truyền qua backbone (mạng CNN đã huấn luyện trước) để trích xuất đặc trưng, sau đó đi qua các lớp đặc trưng bổ sung (extra feature layers) nhằm tạo ra các feature map ở nhiều độ phân giải khác nhau. Việc sử dụng MobileNet giúp giảm đáng kể số lượng tham số và chi phí tính toán, khiến mô hình đặc biệt phù hợp để triển khai trên các nền tảng như Jetson Nano.

Dưới đây hình ảnh kiến trúc của 2 mô hình SSD và SSD-MobileNet:



Hình 2.3: Sơ đồ kiến trúc của mô hình SSD - nguồn: 1512.02325 - SSD



Hình 2.4: Sơ đồ kiến trúc của mô hình SSD-MobileNet – nguồn: https://github.com/dusty-nv/jetson-inference

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Hình 2.5: Bảng so sánh hiệu suất giữa các kiến trúc mạng nơ-ron khác nhau - 1704.04861v1 - MobileNets

Dựa vào bảng trên cho thấy MobileNet cực kỳ hiệu quả cho thiết bị di động hoặc nhúng – vẫn đạt được hiệu suất tốt mà tiết kiệm tài nguyên.

Tên layer	Kích thước	Số kênh	Phát hiện
Conv4_3	38x38	512	Vật thể rất nhỏ
Conv6	19x19	1024	Vật thể nhỏ
Conv7	19x19	1024	Vật thể nhỏ
Conv6_2	10x10	512	Vật thể trung bình
Conv9_2	5x5	256	Vật thể lớn
Conv10_2	3x3	256	Vật thể rất lớn
Conv11_2	1x1	256	Vật thể khổng lồ

Bảng 2.1: Các feature map trong kiến trúc SSD-MobileNet

Trong đó:

- Conv: Lớp xử lý ảnh bằng phép tích chập
- Số sau Conv: Thứ tự layer hoặc vị trí trong block
- Kích thước: Độ phân giải không gian (H×W), ảnh đầu vào input 300x300x3 (HxWxC)
- Số kênh: Số lượng đặc trưng tại mỗi vị trí

2.4. Dữ liệu đầu vào PascalVOC

Pascal VOC (viết tắt của Visual Object Classes) là một định dạng chú thích (annotation) tiêu chuẩn được sử dụng rộng rãi trong các bài toán nhận diện đối tượng, phân đoạn hình ảnh và các nhiệm vụ thị giác máy tính khác, đặc biệt là object detection và classification. Định dạng này đã trở thành một trong những chuẩn mực quan trọng nhất trong lĩnh vực nhận dạng và phát hiện đối tượng.

```
<annotation>
  <folder>JPEGImages</folder>
  <filename>file33.png</filename>
  <path>/home/jetson/Downloads/data/GG/JPEGImages/file33.png</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>300</width>
    <height>300</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>xe4banh</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>33</xmin>
      <ymin>76</ymin>
      <xmax>183</xmax>
      <ymax>182</ymax>
    </bndbox>
  </object>
  <object>
    <name>xemay</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>200</xmin>
      <ymin>110</ymin>
      <xmax>245</xmax>
      <ymax>153</ymax>
    </bndbox>
  </object>
</annotation>
```

Hình 2.6: Pascal VOC sử dụng XML format để lưu thông tin bounding box annotation của ảnh.

2.5. Phần cứng

2.5.1. Camera và Lidar

Camera IMX219-160

Hình 2.7: IMX219-160 Camera

- 8,000,000 pixels (điểm ảnh)
- Chip cảm biến ánh sáng: IMX219
- Độ phân giải: 3280 x 2464
- Góc nhìn đường chéo (FOV): 160°
- Độ méo: <14,3%
- Kích thước ống kính: 6,5mm x 6,5mm
- Kích thước: 25mm × 24mm

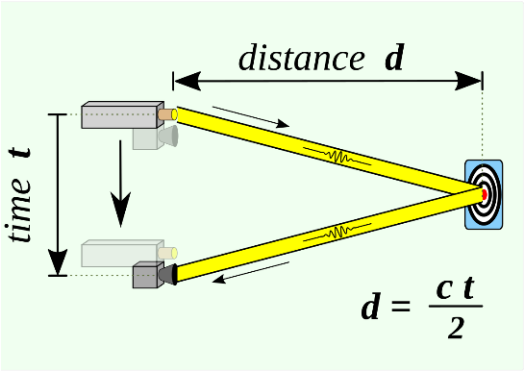
Lidar RPLIDAR A1

RPLidar A1 là một cảm biến lidar 2D, hoạt động dựa trên nguyên lý đo tam giác (triangulation measurement). Thiết bị sử dụng một tia laser phát ra từ bộ phát, sau đó tia này phản xạ lại từ vật thể và được thu nhận bởi bộ thu. Dựa vào góc và thời gian phản xạ, RPLidar A1 tính toán được khoảng cách từ cảm biến đến vật thể tại từng góc quét. Đầu quét của RPLidar A1 có thể xoay 360 độ tầm hoạt động lên đến 12m, cho phép thu thập dữ liệu không gian 2D toàn diện xung quanh thiết bị.

Hình 2.8: RPLidar A1

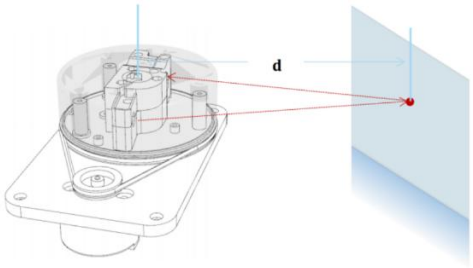
Nguyên lý hoạt động:

- Một tia laser được phát ra và quét liên tục theo vòng tròn nhờ động cơ quay.
- Khi tia laser gặp vật thể, nó sẽ phản xạ lại và được cảm biến thu nhận.
- Dựa vào vị trí góc quay và thời gian nhận tín hiệu phản xạ, thiết bị xác định được tọa độ điểm không gian 2D (góc, khoảng cách).
- Quá trình này lặp lại liên tục, tạo ra một tập hợp các điểm (point cloud) mô tả môi trường xung quanh.



Hình 2.9a Nguyên lý đo lường cơ bản của lidar

nguồn: Onion Tau LiDAR Camera



Hình 2.9b Nguyên lý hoạt động của RPLidar A1 –
nguồn: Triangulation Lidar - Zhejiang Tongzhu Technology Co., Ltd

RPLIDAR A1 vận hành dựa trên nguyên lý đo triangulation measurement. Thiết bị hoạt động hiệu quả trong các không gian nội thất và ngoại thất với điều kiện không tiếp xúc trực tiếp với ánh nắng mặt trời.

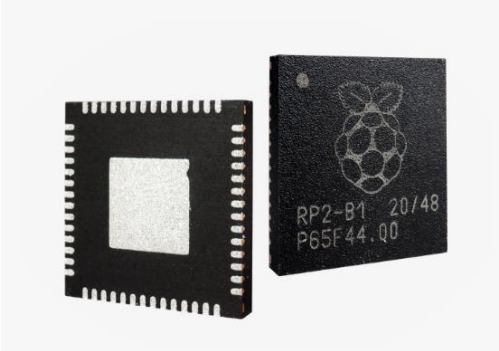
2.5.2. NVIDIA jetson nano và Board điều khiển với Raspberry RP2040

Hình 2.10 NVIDIA Jetson Nano Developer Kit, Small AI Computer – nguồn: NVIDIA Jetson Nano Developer Kit, Small AI Computer

NVIDIA Jetson Nano Developer Kit là một máy tính nhúng nhỏ gọn, mạnh mẽ cho phép chạy song song nhiều mạng neural cho các ứng dụng như phân loại hình ảnh, phát hiện đối tượng, phân đoạn ảnh và xử lý giọng nói trên một nền tảng dễ sử dụng chỉ tiêu thụ từ 5 watt. Được thiết kế đặc biệt cho các ứng dụng AI, machine learning và computer vision.

Thông số:

- GPU: 128 nhân Maxwell™ (472 GFLOPS)
- CPU: Quad-core ARM® Cortex®-A57 (1.43 GHz)
- Bộ nhớ: 4GB LPDDR4 64-bit
- Lưu trữ: Khe cắm thẻ microSD (yêu cầu thẻ nhớ tối thiểu 16GB)



Hình 2.11: Vi điều khiển RP2040 – nguồn: RP2040 – Raspberry Pi

Raspberry Pi RP2040 là vi điều khiển đầu tiên được phát triển bởi Raspberry Pi Foundation, đóng vai trò kết nối thế giới phần mềm với thế giới phần cứng. Vi điều khiển này cho phép các nhà phát triển viết phần mềm tương tác với thế giới vật lý một cách xác định và chính xác đến từng chu kỳ, tương tự như logic kỹ thuật số.

Thông số cơ bản:

- Vi xử lý: Dual ARM Cortex-M0+ @ 133MHz
- Bộ nhớ: 264kB SRAM tích hợp chia thành sáu ngân hàng độc lập

2.6. Phần mềm

2.6.1. Nomachine

Hình 2.12: Logo Nomachine software

NoMachine là phần mềm điều khiển máy tính từ xa, cho phép người dùng truy cập và quản lý JetRacer hoặc các thiết bị khác qua mạng. Nhờ NoMachine, việc cài đặt, cấu hình và giám sát hệ thống trở nên thuận tiện hơn mà không cần thao tác trực tiếp trên thiết bị.

2.6.2. VMware

VMware là phần mềm tạo máy ảo, giúp người dùng mô phỏng nhiều hệ điều hành khác nhau trên cùng một máy tính vật lý. Trong quá trình phát triển và kiểm thử hệ thống, VMware hỗ trợ tạo môi trường ảo để thử nghiệm phần mềm trước khi triển khai lên thiết bị thực tế, đảm bảo an toàn và tiết kiệm thời gian.

Hình 2.13: Logo WMwave softwave

2.6.3. labelImg

LabelImg là một công cụ chú thích ảnh thường được sử dụng trong tiền xử lý dữ liệu học sâu dùng để chú thích (annotation) hình ảnh, đóng vai trò quan trọng trong việc chuẩn bị dữ liệu huấn luyện cho mô hình phát hiện đối tượng. Phần mềm hỗ trợ người dùng vẽ hộp giới hạn (bounding box) quanh các đối tượng cần phát hiện và gán nhãn tương ứng cho từng đối tượng.

LabelImg hỗ trợ xuất dữ liệu theo các định dạng phổ biến như PASCAL VOC và YOLO, phù hợp với nhiều framework học máy khác nhau.

Hình 2.14: Logo labelImg softwave

2.6.4. Jetson-inference

Hình 2.15: Hình giới thiệu thư viện cho Nvidia Jetson - nguồn: https://github.com/dusty-nv/jetson-inference

Jetson-inference là một thư viện suy luận (inference library) mã nguồn mở mạnh mẽ và linh hoạt được phát triển bởi NVIDIA, cung cấp các công cụ tối ưu hóa cho việc triển khai các mô hình deep learning trên nền tảng NVIDIA Jetson. Thư viện này được thiết kế đặc biệt để tận dụng khả năng tính toán GPU của các thiết bị Jetson nhằm chuyên dụng thực hiện cho việc triển khai các ứng dụng Trí tuệ nhân tạo (AI) liên quan đến thị giác máy tính (Computer Vision) trên các bo mạch nhúng NVIDIA Jetson (như Jetson Nano, Xavier NX, Orin Nano, v.v.). Thư viện này được xây dựng với mục tiêu tối ưu hóa hiệu suất suy luận (inference) của các mô hình học sâu (Deep Learning), đồng thời đơn giản hóa quá trình phát triển cho các nhà nghiên cứu và kỹ sư, , phù hợp cho các ứng dụng edge computing và robotics.

Hình 2.16: Logo Pytorch

PyTorch là một framework học máy mã nguồn mở dựa trên thư viện Torch, được sử dụng rộng rãi trong lĩnh vực thị giác máy tính và xử lý ngôn ngữ tự nhiên. Framework này được phát triển bởi Meta AI (trước đây là Facebook AI Research)

và hiện tại là một phần của Linux Foundation. PyTorch được phát hành dưới giấy phép BSD đã qua sửa đổi, đảm bảo tính tự do và mã nguồn mở.

Module training với PyTorch, jetson-inference tích hợp một hệ thống huấn luyện hoàn chỉnh dựa trên PyTorch framework, cho phép:

- Huấn luyện các mô hình tùy chỉnh từ đầu hoặc fine-tuning từ các mô hình pre-trained
- Hỗ trợ nhiều kiến trúc mạng phổ biến như MobileNet, ResNet, VGG cho các bài toán khác nhau
- Tự động chuyển đổi mô hình từ định dạng PyTorch (.pth) sang ONNX (.onnx) và cuối cùng là TensorRT engine (.engine)
- Cung cấp các script huấn luyện được tối ưu hóa cho từng loại bài toán cụ thể

2.6.5. Google Colab

Google Colaboratory, hay còn được gọi là Google Colab, là một dịch vụ điện toán đám mây miễn phí do Google cung cấp. Nền tảng này cho phép người dùng viết và thực thi mã Python trực tiếp trên trình duyệt web mà không cần phải cài đặt bất kỳ phần mềm nào trên máy tính cá nhân.

Về cơ bản, Google Colab là một phiên bản Jupyter Notebook được lưu trữ và chạy trên nền tảng đám mây của Google, cung cấp một môi trường tính toán mạnh mẽ, đặc biệt hữu ích cho các tác vụ liên quan đến khoa học dữ liệu, học máy (machine learning) và trí tuệ nhân tạo (AI).

Hình 2.17: Logo Google Colab

Các đặc điểm chính:

- Miễn phí truy cập tài nguyên tính toán: Người dùng được cung cấp quyền truy cập miễn phí vào các tài nguyên phần cứng mạnh mẽ như GPU (Bộ xử lý đồ họa) và TPU (Bộ xử lý Tensor). Điều này cực kỳ quan trọng đối với việc huấn luyện các mô hình học máy phức tạp, vốn đòi hỏi năng lực tính toán cao mà nhiều máy tính cá nhân không thể đáp ứng.
- Môi trường được cài đặt sẵn: Google Colab đi kèm với nhiều thư viện khoa học dữ liệu và học máy phổ biến đã được cài đặt sẵn, chẳng hạn như TensorFlow, PyTorch, Keras, Pandas, và Scikit-learn. Điều này giúp người dùng tiết kiệm đáng kể thời gian và công sức trong việc thiết lập môi trường làm việc.
- Tích hợp với hệ sinh thái Google: Các tệp Colab (có đuôi .ipynb) được lưu trữ trực tiếp trên Google Drive của người dùng, giúp dễ dàng quản lý, chia sẻ và cộng tác với người khác trong thời gian thực, tương tự như các sản phẩm khác của Google Workspace (ví dụ: Google Docs, Google Sheets).
- Dễ dàng chia sẻ và cộng tác: Người dùng có thể chia sẻ notebook của mình một cách đơn giản thông qua một liên kết, cho phép người khác xem, nhận xét hoặc chỉnh sửa trực tiếp. Tính năng này thúc đẩy tinh thần hợp tác và chia sẻ kiến thức trong cộng đồng nghiên cứu và phát triển.

2.6.6. ROS

ROS (Robot Operating System) là một framework middleware mã nguồn mở được thiết kế để phát triển phần mềm robot. Mặc dù có tên gọi là "hệ điều hành", ROS thực chất là một tập hợp các thư viện và công cụ giúp đơn giản hóa việc tạo ra các ứng dụng robot phức tạp và đa dạng.

Hình 2.18: Logo ROS (Robot Operating System)

Kiến trúc và nguyên lý hoạt động:

- Mô hình node-based

  ROS tổ chức các chức năng thành một mạng lưới các node - những tiến trình độc lập thực hiện các tác vụ cụ thể. Mỗi node có thể:

  - Xử lý dữ liệu từ cảm biến
  - Thực hiện các thuật toán điều khiển
  - Quản lý giao tiếp với phần cứng
  - Thực hiện các tác vụ lập kế hoạch và ra quyết định

- Hệ thống truyền thông

  ROS cung cấp cơ chế truyền thông linh hoạt giữa các node thông qua:

  - Topics: Kênh truyền dữ liệu bất đồng bộ
  - Services: Giao tiếp đồng bộ dạng request-response
  - Actions: Xử lý các tác vụ dài hạn với feedback
  - Parameters: Hệ thống cấu hình động

CHƯƠNG 3: XÂY DỰNG VÀ TRIỂN KHAI HỆ THỐNG CẢNH BÁO VA CHẠM TRÊN JETRACER ROS AI KIT

3.1. Thiết lập môi trường, thu thập và chuẩn bị dữ liệu

3.1.1. Thiết lập môi trường phát triển

Môi trường phát triển được thiết lập dựa trên image Waveshare cung cấp, bao gồm Ubuntu 18.04 và ROS Melodic, được cấu hình sẵn cho Jetracer sử dụng camera IMX219-160 và RPLIDAR A1, bao gồm cả máy ảo Ubuntu.

Sau khi flash image vào thẻ SD và khởi động hệ thống, việc kết nối với JetRacer được thực hiện thông qua NoMachine để thao tác từ xa, giúp dễ dàng quản lý hệ thống mà không cần kết nối trực tiếp màn hình. Các packages ROS đã được cài đặt sẵn cho phép tương tác với các cảm biến IMX219-160 camera và RPLIDAR A1 thông qua các topics tương ứng như /camera/image_raw và /scan.

Hình 3.1: Logo Waveshare

Cách flash thẻ SD cho Jetracer và cài đặt môi trường máy ảo lên VMwave:

- Format thẻ nhớ SD bằng phần mềm SDFormatter
- Tải file img trên trang chủ của Wareshare jetracer_ros.zip - Google Drive và flash file vào thẻ SD bằng phần mềm Win32diskimager
- Khởi động Jetracer
- Kết nối mạng cục bộ, tài khoản mật khẩu “jetracer”, nếu không có màn hình thì dùng MobaxTerm để nhập lệnh trên terminal kết nối mạng cục bộ (có màn hình rời, bỏ qua bước này):
    - Kết nối Jetracer với máy tính dùng dây micro usb để kết nối Jetracer qua giao diện serial hoặc dây ethernet qua giao diện ssh của MobaxTerm.

- Nhập lệnh sau để kết nối mạng wifi nội bộ:

```
#Bật wifi
$ sudo nmcli r wifi on
#Quét wifi
$ sudo nmcli dev wifi
#Nhập SSID và mật khẩu wifi
$ sudo nmcli dev wifi connect wifi_name password wifi_password
```

- Ngắt kết nối dây micro usb hoặc dây ethernet giữa Jetracer và máy tính.

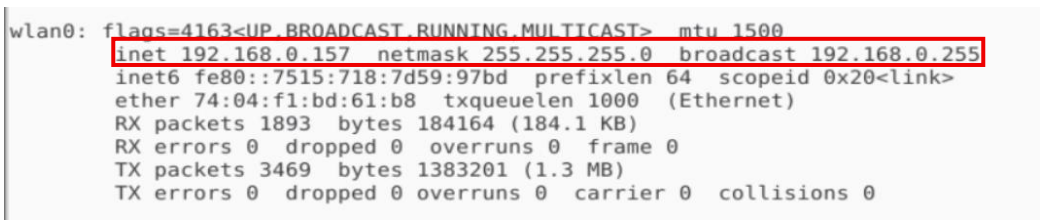
Để giảm tải xử lý cho Jetson Nano, môi trường ROS phân tán được thiết lập giữa JetRacer và máy ảo Ubuntu trên VMware. Kiến trúc này cho phép Jetson Nano tập trung vào việc thu thập dữ liệu từ cảm biến và điều khiển robot, trong khi máy ảo Ubuntu với tài nguyên tính toán mạnh hơn đảm nhận các tác vụ xử lý ảnh nặng. Việc cấu hình được thực hiện bằng cách chỉnh sửa các file /etc/hosts và ~/.bashrc trên cả hai máy, thiết lập ROS_MASTER_URI và ROS_HOSTNAME, tạo nên một hệ thống phân tán hiệu quả với việc truyền dữ liệu giữa các nodes thông qua network.

3.1.2. Giao tiếp giữa máy ảo Ubuntu và JetRacer

Việc cấu hình được thực hiện bằng cách chỉnh sửa các file /etc/hosts và ~/.bashrc trên cả hai máy, thiết lập ROS_MASTER_URI và ROS_HOSTNAME, tạo nên một hệ thống phân tán hiệu quả với việc truyền dữ liệu giữa các nodes thông qua network.

- Lấy IP qua lệnh ifconfig

  Để biết địa chỉ IP của máy ảo Ubuntu, dùng cho việc cấu hình ROS, chỉnh sửa file hosts. Tương tự cho Jetracer



- Chỉnh sửa file .bachrc (Thêm ROS_MASTER_URI và ROS_HOSTNAME) Thiết lập các biến môi trường để ROS biết địa chỉ máy chủ (ROS Master) và hostname của từng máy, giúp các node ROS trên hai máy nhận diện và kết nối với nhau qua mạng. Tương tự cho jetracer.

Lệnh chỉnh sửa .bachrc: $ sudo vim /.bachrc

```
source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash
export ROS_MASTER_URI=http://nano-4gb-jp451:11311    #将机器人设置为主机
export ROS_HOSTNAME=nano-4gb-jp451
```

- Chỉnh sửa IP trỏ về IP của máy ảo Ubuntu trong file hosts trên Jetracer Đảm bảo JetRacer có thể phân giải hostname của máy ảo Ubuntu thành đúng địa chỉ IP, giúp giao tiếp ROS diễn ra thuận lợi. Tương tự cho Jetracer.

Lệnh chỉnh sửa file hosts: $ sudo vim /etc/hosts

```
127.0.0.1       localhost
127.0.1.1       nano-4gb-jp451.localdomain      nano-4gb-jp451
10.27.226.68    ubuntu
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Các bước này giúp JetRacer và máy ảo Ubuntu có thể chia sẻ và giao tiếp các node, topic với nhau thông qua ROS. Điều này cho phép các node ROS trên hai máy xuất bản (publish) và đăng ký (subscribe) topic qua mạng, tạo thành một hệ thống phân tán hợp lý.

3.1.3. Thu thập dữ liệu giao thông thực tế

Để mô hình có khả năng hoạt động tốt trong môi trường thực tế nên đã tiến hành thu thập dữ liệu video trực tiếp từ các tuyến đường giao thông đô thị.

- Phương pháp: Dữ liệu được ghi lại bằng cách sử dụng camera IMX219-160 gắn trên xe JetRacer. Việc thu thập bằng chính thiết bị phần cứng sẽ được triển khai giúp đảm bảo dữ liệu có các đặc tính (như góc nhìn, độ phân giải, độ nhiễu) tương đồng với điều kiện hoạt động thực tế của sản phẩm.

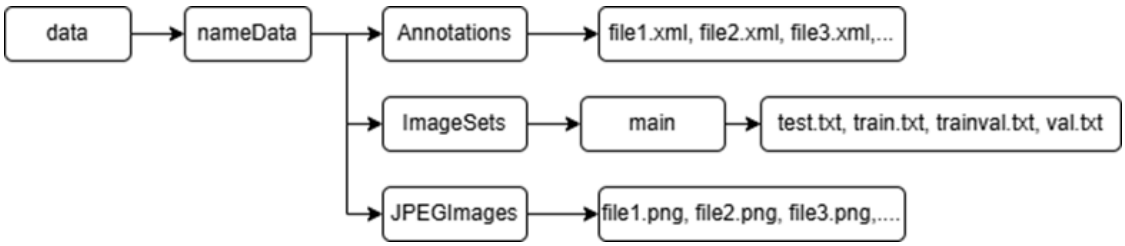
- Công cụ lưu trữ: Tôi đã sử dụng công cụ rosbag của ROS để ghi lại toàn bộ luồng dữ liệu hình ảnh từ topic /camera/image_raw và dữ liệu khoảng cách /scan. rosbag là một công cụ mạnh mẽ cho phép lưu trữ và phát lại các thông điệp ROS, đảm bảo dữ liệu được ghi lại một cách đồng bộ, có gắn nhãn thời gian (timestamp), và đảm bảo tính toàn vẹn của dữ liệu. Lệnh dưới đây được sử dụng để bắt đầu quá trình ghi dữ liệu:

  ```
  # Ghi dữ liệu từ topic camera và lưu vào tệp có tiền tố 'traffic_data'
  rosbag record -a/camera/image_raw/compession…
  ```

- Tổng hợp video giao thông đô thị

3.1.4. Chuẩn bị dữ liệu ban đầu và gán nhãn dữ liệu theo định dạng Pascal VOC

Pascal VOC (Pattern Analysis, Statistical Modeling, Computational Learning Visual Object Challenge) là một định dạng dữ liệu chuẩn dùng trong các bài toán nhận diện đối tượng (object detection). Định dạng này cung cấp dữ liệu gồm ảnh và các annotation (chú thích) đi kèm.



Hình 3.2: Cấu trúc thư mục dữ liệu theo chuẩn Pascal VOC

- Trong đó:
  - JPEGImages: Lưu trữ dưới dạng file ảnh (.jpg, .png,…).
  - Annotation: Lưu trữ dưới dạng file XML, chứa thông tin về bounding box (hộp giới hạn) của các đối tượng trong ảnh và được tạo tự động bởi công cụ labelImg dựa trên bounding box mà người dùng đã vẽ.
  - ImageSets: giúp xác định rõ ràng ảnh nào thuộc tập train, val, test, đảm bảo quá trình huấn luyện và đánh giá nhất quán.
- Trích xuất frames từ rosbag, sau đó xuất một tập dữ liệu ảnh (chưa đánh bounding box) và phải theo cấu trúc dữ liệu Pascal VOC

Bộ dữ liệu gồm 561 hình ảnh đã được phân chia theo phương pháp Hold-out để phục vụ cho việc huấn luyện và đánh giá mô hình máy học.

Tập huấn luyện (Training Set)

- Mục đích: Dữ liệu trong tập này được dùng để "dạy" mô hình, giúp nó học cách nhận diện các đặc trưng và điều chỉnh các tham số (weights) bên trong.
- Quy mô: Chiếm 80% tổng dữ liệu (khoảng 448 ảnh).
- Tệp định nghĩa: Danh sách các ảnh huấn luyện được liệt kê trong file train.txt (hoặc trainval.txt).

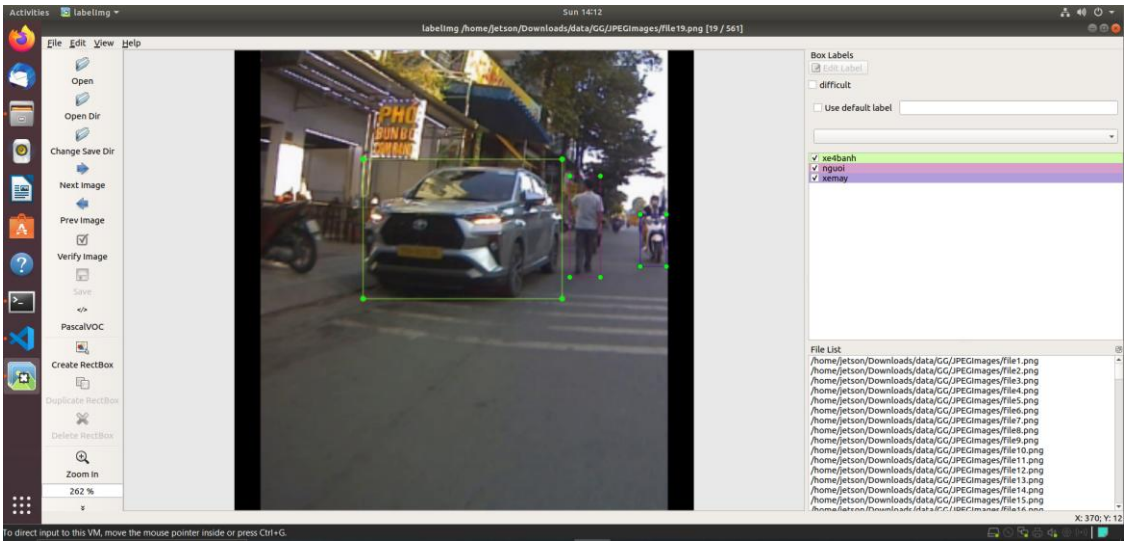
Tập kiểm thử (Testing Set)

- Mục đích: Dữ liệu trong tập này chỉ được sử dụng một lần duy nhất sau khi mô hình đã huấn luyện xong, nhằm đánh giá hiệu suất cuối cùng của mô hình trên dữ liệu hoàn toàn mới (unseen data).
- Quy mô: Chiếm 10% tổng dữ liệu (khoảng 56 ảnh).
- Tệp định nghĩa: Danh sách các ảnh kiểm thử được liệt kê trong file test.txt.

Tập xác thực (Validation Set)

- Mục đích: Dữ liệu này được dùng trong quá trình huấn luyện để đánh giá hiệu suất tạm thời của mô hình sau mỗi epoch, nhằm điều chỉnh siêu tham số (hyperparameters) như tốc độ học (learning rate), số lớp, v.v.
- Quy mô: Chiếm khoảng 10% tổng dữ liệu (tương ứng 56 ảnh).
- Tệp định nghĩa: Danh sách các ảnh xác thực được liệt kê trong tệp val.txt

Sử dụng công cụ labelImg tạo hộp giới hạn quanh đối tượng trên từng ảnh. Thông tin nhãn và vị trí được lưu dưới dạng file XML để phục vụ huấn luyện mô hình.



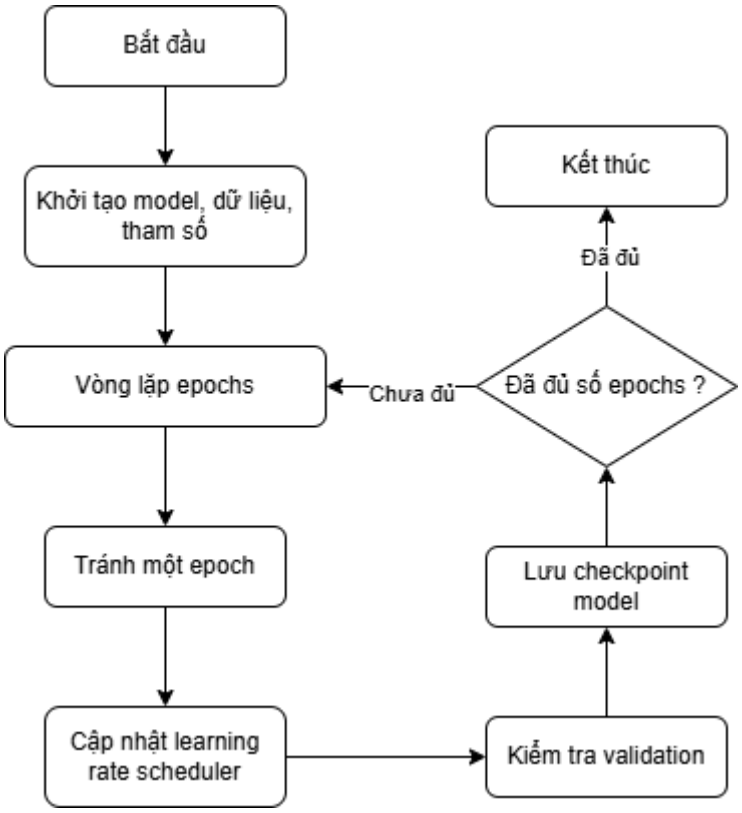
Hình 3.3: Giao diện làm việc labelImg

- Open Dir -> đường dẫn đến thư mục JPEGImages
- Change Save Dir -> đường dẫn đến thư mục Annotations
- Create ReatBox: đánh hộp giới hạn

3.2. Huấn luyện SSD Mobilenet model

3.2.1. Huấn luyện

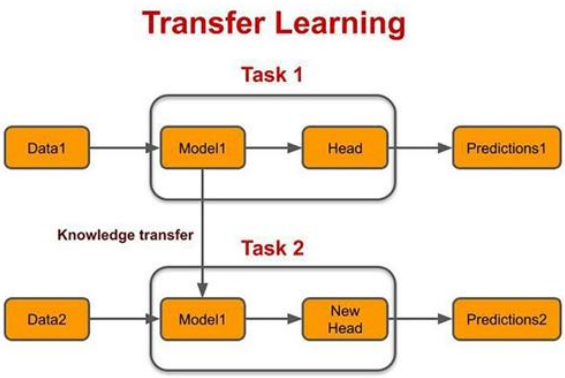
Mô hình SSD MobileNet được lựa chọn cho dự án Jetracer nhờ sự cân bằng giữa độ chính xác và tốc độ xử lý, rất phù hợp với khả năng tính toán của Jetson Nano. Quá trình huấn luyện sử dụng mô hình pre-trained (đã được huấn luyện trước trên tập dữ liệu lớn) để thực hiện transfer learning trên Google Colab, tận dụng tài nguyên GPU miễn phí. Sau khi huấn luyện lại với dữ liệu của dự án, mô hình được triển khai lên Jetson Nano thông qua framework jetson-inference.



Hình 3.4: Quy trình huấn luyện mô hình học sâu

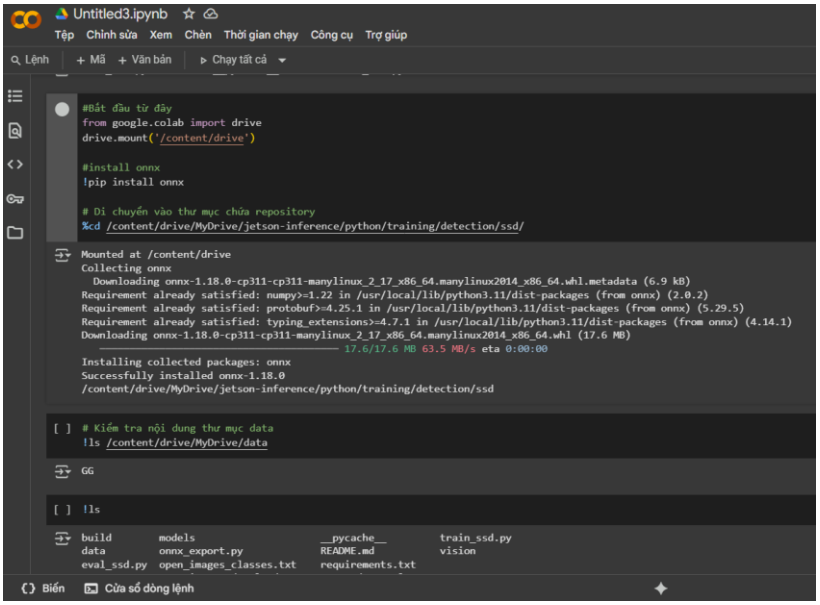
- Trong đó:
    - Epoch: Một lần duyệt toàn bộ dữ liệu huấn luyện qua mô hình.
    - Learning rate scheduler: Tự động điều chỉnh tốc độ học sau mỗi epoch để tối ưu quá trình huấn luyện.
    - Validation: Đánh giá mô hình trên tập dữ liệu riêng sau mỗi epoch để kiểm tra chất lượng mô hình.

Transfer learning (học chuyển giao) là kỹ thuật tận dụng một mô hình đã được huấn luyện trước (pre-trained model) trên một tập dữ liệu lớn, tổng quát, sau đó tinh chỉnh (fine-tune) mô hình này trên tập dữ liệu mới, nhỏ hơn, cho bài toán cụ thể.



Hình 3.5: Hình dung Transfer Learning – nguồn: An Ultimate Guide To Transfer Learning In NLP

Sử dụng Google Colab trong việc training model, tận dụng sức mạnh GPU hỗ trợ CUDA và thời gian training nhanh chóng.



Hình 3.6: Giao diện làm việc Google Colab

- Tải framework jetson-inference về máy (máy hỗ trợ card đồ họa nvidia hoặc các board tương tự như jetson nano) có hỗ trợ NVIDIA Jetson platform (CUDA, cuDNN, and TensorRT) và đẩy lên thư mục chính của Google Driver

Trên jetracer dùng lệnh sau để tải framework của jetson-inference:
`$sudo apt-get update`
`$sudo apt-get install git cmake libpython3-dev python3-numpy`

```
$git clone --recursive --depth=1
https://github.com/dusty-nv/jetson-inference
$cd jetson-inference
$mkdir build
$cd build
$cmake ../
$make -j$(nproc)
$sudo make install
$sudo ldconfig
```

Sau đó đẩy thư mục jetson-inference lên driver và mount để sử dụng framework

- Mount Driver với Google Colab:

Lệnh mount trên Google Colab:

```
from google.colab import drive
drive.mount('/content/drive')
```

- Lệnh training trên Google Colab :

Lệnh train ssd-mobilenet:

```
%cd/content/drive/MyDrive/jetson-
inference/python/training/detection/ssd/

!python train_ssd.py --net=mb1-ssd --dataset-type=voc
--data=/duongdanfolderA  --model-dir=/duongdanfolderB
--batch-size=8 --workers=4 --epochs=150
```

  - /duongdanfolderA chứa dữ liệu PascalVOC format
  - /duongdanfolderB chứa các file model được train (quá trình train sinh ra nhiều file.pth)

3.2.2. Đánh giá mô hình huấn luyện

Dựa vào log khi training ta có thể nhận biết được:

- Overfitting: Nếu Training Loss giảm liên tục và thấp, nhưng Validation Loss ngừng giảm hoặc tăng lên, mô hình đang học quá kỹ dữ liệu huấn luyện và kém tổng quát hóa cho dữ liệu mới.
- Underfitting: Nếu cả Training Loss và Validation Loss đều cao và không giảm, mô hình chưa học đủ từ dữ liệu.

Đánh giá hiệu năng của một mô hình phát hiện đối tượng (object detection) đối với từng lớp (class) riêng biệt dựa vào Average Precision Per-class (AP):

$$AP = \int_0^1 p(r)dr$$

Trong đó:

- $p(r)$: precision tại Recall r
- $Precision = TP / (TP + FP)$
- $Recall = TP / (TP + FN)$

AP được tính riêng cho mỗi lớp, nên gọi là "AP per-class". Để có đánh giá tổng thể, thường tính mAP (mean Average Precision) bằng trung bình AP của tất cả các lớp.

Lệnh để xem hiệu suất tổng thể của mô hình:

```
!python eval_ssd.py --net=mb1-ssd --model=/duongdanModel.pth --dataset_type=voc --dataset=/content/drive/MyDrive/data/GG --use_cuda=True
```

- /duongdanModel.pth: file model đã train

3.3. Mô hình triển khai hệ thống dùng SSD MobileNet

3.3.1. Chuyển đổi model sang định dạng tối ưu

Sau khi hoàn thành quá trình huấn luyện, mô hình SSD MobileNet cần được chuyển đổi sang định dạng tối ưu để triển khai trên thiết bị Jetson Nano. Quá trình này gồm hai bước chính:

- Chuyển đổi từ PyTorch (.pth) sang ONNX:

Sử dụng công cụ chuyển đổi của jetson-inference để xuất mô hình từ định dạng PyTorch (.pth) sang định dạng ONNX. Định dạng ONNX giúp mô hình dễ dàng tương thích với các công cụ tối ưu hóa và các nền tảng khác nhau.

Hình 3.7: Logo onnx

- Tối ưu hóa với TensorRT:
  Sử dụng NVIDIA TensorRT để chuyển đổi mô hình ONNX thành file engine tối ưu (.engine) cho phần cứng Jetson Nano. File .engine này sẽ được sử dụng trực tiếp khi chạy mô hình, giúp tăng tốc độ suy luận và hiệu suất xử lý.

Nhờ đó, mô hình đạt hiệu quả cao khi triển khai thực tế trên thiết bị Jetson Nano.

Lệnh convert .pth → .onnx trong jetson-inference:

```
!python onnx_export.py --net mb1-ssd --input /duongdan/file.pth --output /duongdan/file.onnx --labels /content/drive/MyDrive/models/model4/labels.txt --resolution 300
```

3.3.2. Thực hiện nhận diện đối tượng bằng SSD MobileNet

Để thực hiện nhận diện đối tượng trên JetRacer, hệ thống sử dụng mô hình SSD MobileNet đã được huấn luyện và chuyển đổi sang định dạng ONNX, kết hợp với thư viện jetson.inference và jetson.utils tối ưu cho GPU NVIDIA Jetson.

Mô hình SSD MobileNet (được nạp qua jetson.inference.detectNet) thực hiện nhận diện trực tiếp trên GPU, trả về danh sách các đối tượng phát hiện được (bounding box, class, confidence).

```python
def main():
    global net, detection_pub
    rospy.init_node('detect_object_basic')
    detection_pub = rospy.Publisher('detection_data_basic', String, queue_size=1)

    # Initialize model
    model_path = "/home/jetson/Desktop/kl_model.onnx"
    labels_path = "/home/jetson/Desktop/labels.txt"
    threshold = 0.5

    try:
        net = jetson.inference.detectNet(
            model=model_path,
            labels=labels_path,
            input_blob="input_0",
            output_cvg="scores",
            output_bbox="boxes",
            threshold=threshold
        )
    except Exception as e:
        rospy.logerr(f"Model initialization error: {e}")
        return

    # Subscribe to camera
    camera_topic = '/csi_cam_0/image_raw/compressed'
    rospy.Subscriber(camera_topic, CompressedImage, callback, queue_size=1, buff_size=2**24)
    rospy.spin()

if __name__ == '__main__':
    main()
```

Hình 3.8: Hàm main nhận dữ liệu từ camera, khởi tạo mô hình nhận diện và publish kết quả lên topic ROS trên Jetracer.

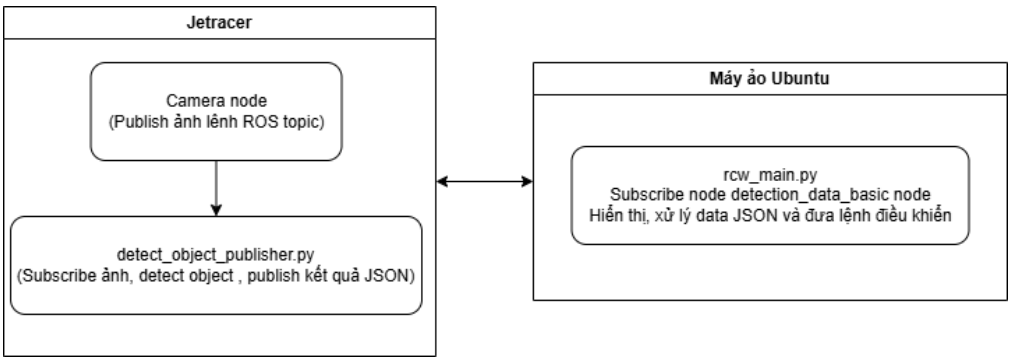
Chạy detect_object_publisher.py, kết quả nhận diện được chuyển thành định dạng JSON, bao gồm: tên lớp đối tượng, độ tin cậy, và tọa độ bounding box.

Kết quả này được publish lên topic ROS (detection_data_basic) để các node khác (máy ảo Ubuntu) sử dụng.

```python
def main():
    rospy.init_node('rcw_fusion_core')
    rc.load_velocity_data()
    rc.reset_velocity()
    rospy.loginfo("Velocity data loaded - simulation will start on first detection!")
    rospy.Subscriber('/csi_cam_0/image_raw/compressed', CompressedImage, rc.image_callback)
    rospy.Subscriber('/detection_data_basic', String, rc.detection_callback)
    rospy.Subscriber('/scan', LaserScan, rc.scan_callback)
    global pub
    pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
    cv2.namedWindow("Camera View", cv2.WINDOW_NORMAL)
    cv2.namedWindow("LiDAR View", cv2.WINDOW_NORMAL)
    rospy.loginfo("RCW Sensor Fusion Started!")
    rate = rospy.Rate(10)
    try:
        loop_count = 0
        while not rospy.is_shutdown():
            key = cv2.waitKey(1) & 0xFF
            if key == 27:
                break  # ESC to exit
            rc.update_velocity_from_file() # Cập nhật vận tốc từ file
            process_detections() # Xử lý các phát hiện mới
            draw_camera_view() # Vẽ góc nhìn camera
            draw_lidar_view() # Vẽ góc nhìn LiDAR
            if loop_count % 1 == 0: # 10hz * 1 = 10Hz tuong duong 0.1s
                # In thông tin TTC và vận tốc vật thể mỗi 0.1 giây
                print_object_velocities(objects) # In thông tin vận tốc của các vật thể
                print_ttc_info() # In thông tin TTC đơn giản ra terminal
            loop_count += 1
            rate.sleep() # Đảm bảo vòng lặp chạy với tần suất 10Hz
    except KeyboardInterrupt:
        pass
    finally:
        rospy.loginfo("Shutting down...")
        cv2.destroyAllWindows()

if __name__ == '__main__':
    try:
        main()
```

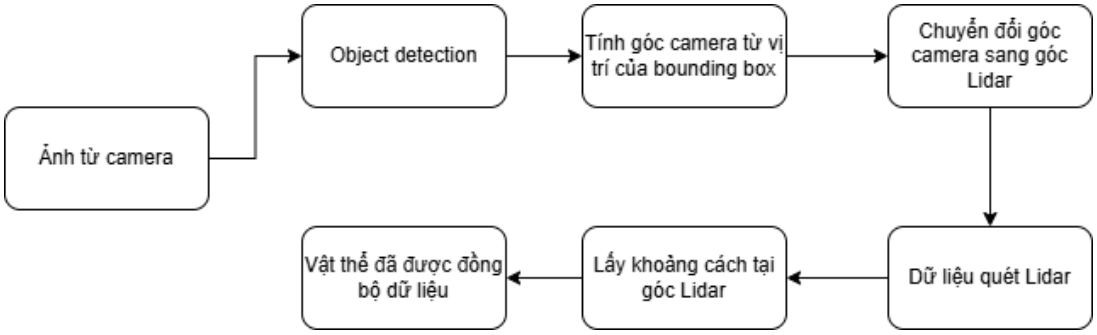
Hình 3.9: Hàm main xử lý RCW trên node máy ảo Ubuntu



Hình 3.10: Sơ đồ khối luồng dữ liệu phát hiện đối tượng giữa JetRacer và máy ảo Ubuntu

3.4. Đồng bộ dữ liệu vật thể

Đồng bộ dữ liệu vật thể là quá trình kết hợp thông tin phát hiện từ nhiều cảm biến khác nhau (ví dụ: Camera và LiDAR) để xác định chính xác vị trí, khoảng cách và đặc điểm của vật thể trong môi trường. Trong hệ thống này, sau khi Camera phát hiện vật thể và xác định vị trí của chúng trên ảnh, hệ thống sẽ tính toán góc quan sát tương ứng trên LiDAR. Từ đó, lấy dữ liệu khoảng cách thực tế tại góc này từ LiDAR và gán cho vật thể đã phát hiện. Kết quả là mỗi vật thể sẽ có đầy đủ thông tin về loại, vị trí trên ảnh, góc quan sát, cũng như khoảng cách thực tế đo được từ LiDAR. Việc đồng bộ này giúp tăng độ chính xác trong nhận diện và định vị vật thể, đồng thời hỗ trợ hiệu quả cho các chức năng cảnh báo và điều khiển an toàn.



Hình 3.11: Sơ đồ khối quá trình đồng bộ dữ liệu vật thể giữa Camera và LiDAR

- **Lấy góc của vật thể trên màn ảnh**:

$$\text{camera\_angle} = \text{CAMERA\_ANGLE\_MAX} - \left(\frac{center\_x}{image\_width}\right) \times CAMERA\_FOV$$

Trong đó:

  - camera_angle là góc của vật thể (0 ° → -90 °)
  - CAMERA_ANGLE_MAX là góc lớn nhất, tương đương 0 ° (góc ngoài cùng bên trái khi nhìn vào khung hình)
  - $center\_x$ tọa độ x của bounding box trên ảnh

- $image\_width$ là chiều rộng của ảnh camera (tính bằng pixel).
- $CAMERA\_FOV$ là góc nhìn ngang (field of view) của camera.

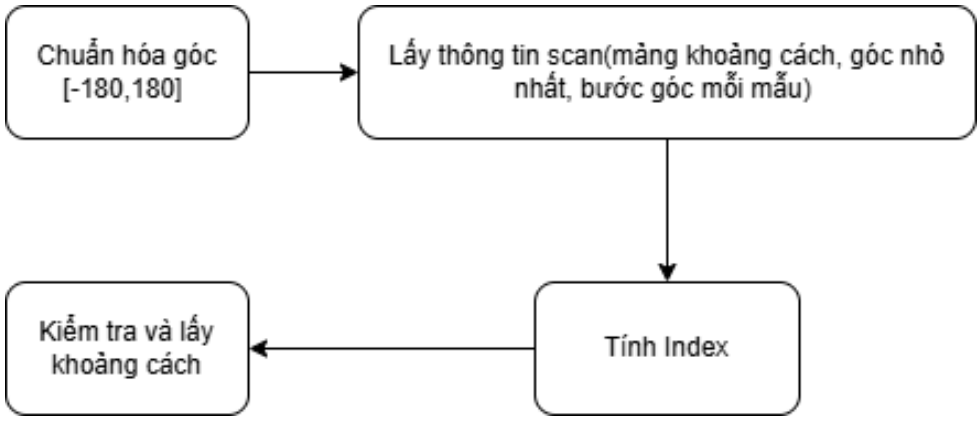
- **Lấy góc của vật thể trên vùng lidar**:

$$lidar\_angle = (camera\_angle + 180)\ \%\ 360 + ANGLE\_OFFSET$$

Trong đó:

- lidar_angle là góc của vật thể trên vùng quét lidar (220° → 130°) dựa vào camera_angle
- camera_angle góc vật thể trên camera
- ANGLE_OFFSET là hằng số chỉnh cho camera và lidar khớp vùng với nhau.

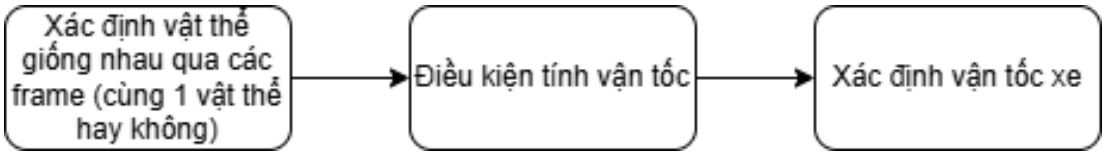
- **Tính khoảng cách vật thể**:



Hình 3.12: Sơ đồ xử lý khoảng cách từ thông tin scan của node lidar

- Đầu tiên, góc cần lấy khoảng cách được chuẩn hóa về khoảng [-180°, 180°] để phù hợp với hệ quy chiếu của LiDAR.
- Hệ thống lấy các thông tin từ dữ liệu quét LiDAR, bao gồm: mảng khoảng cách (ranges), góc nhỏ nhất (angle_min) và bước tăng góc mỗi mẫu (angle_increment).
- Sử dụng góc đã chuẩn hóa, hệ thống tính toán chỉ số (index) tương ứng trong mảng dữ liệu LiDAR. Dữ liệu thô của LiDAR là một mảng các khoảng cách, mỗi giá trị ứng với một góc quét nhất định, giúp bạn tái dựng bản đồ 2D xung quanh cảm biến (360 °).
- Cuối cùng, hệ thống kiểm tra chỉ số hợp lệ và lấy giá trị khoảng cách tại vị trí đó. Nếu giá trị hợp lệ, đây chính là khoảng cách thực tế từ xe đến vật thể tại góc mong muốn.

- **Tính vận tốc của vật thể:**



Hình 3.13: Sơ đồ tính vận tốc xe

  - Xác định vật thể giống nhau bằng cách so sánh hai lần phát hiện được coi là cùng một vật thể nếu có cùng loại (class_name) và góc camera (đã làm tròn theo bội số 3 độ) giống nhau.
  - Điều kiện tính vận tốc: ít nhất có 2 frame, phải có dữ liệu distance (khoảng cách) và thời gian dt (denta t) > 0.1s.
  - Công thức tính vận tốc:
    - Vận tốc tương đối:

$$velocity\_relative = \frac{s2 - s1}{dt}$$

Trong đó:

- $velocity\_relative$ là vận tốc vật thể so với Jetracer (lấy Jetracer làm mốc)
- s2 là khoảng cách vật thể của frame trước
- s1 là khoảng cách vật thể của frame sau
- dt là khoảng thời gian giữa s2 và s1

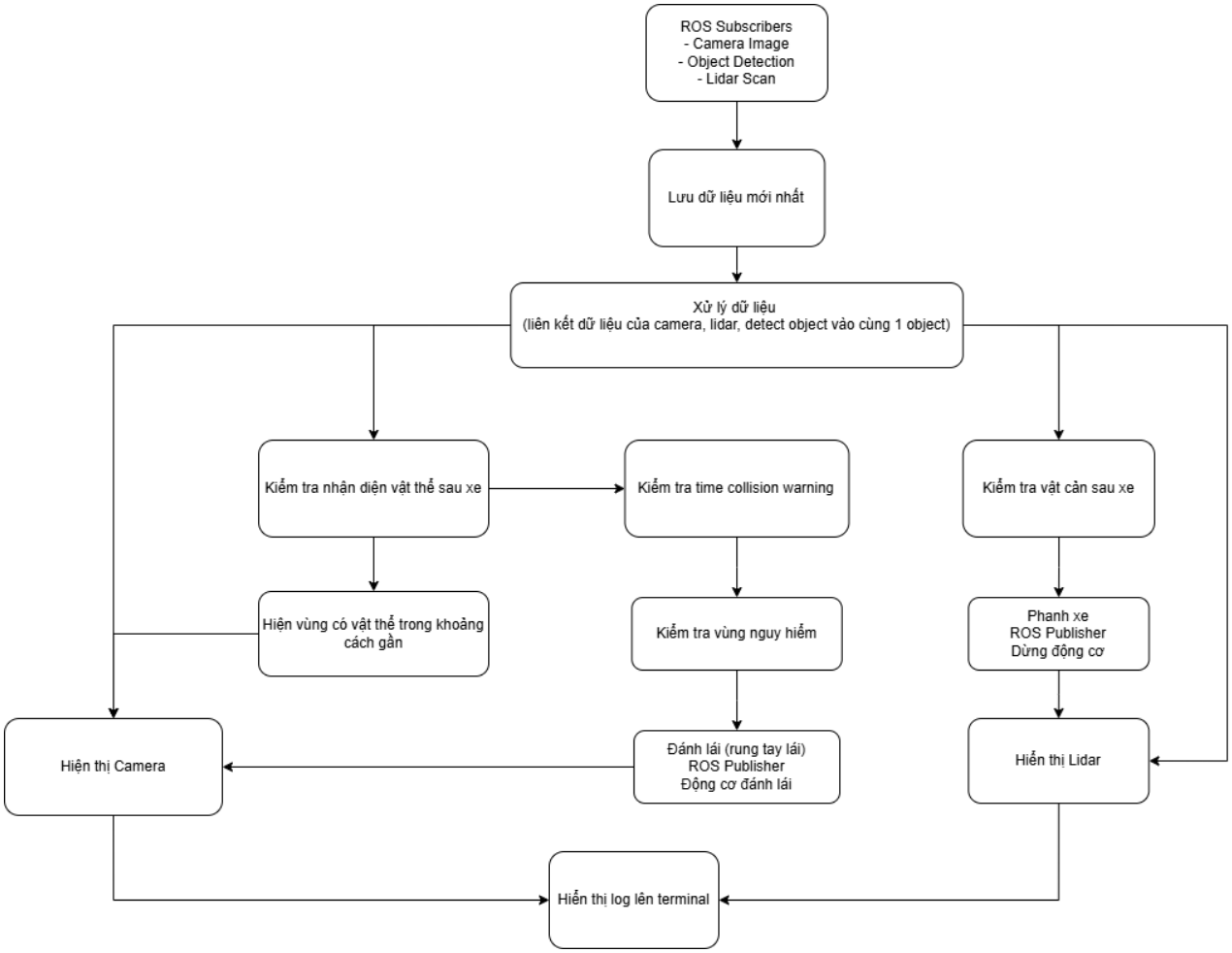
    - Vận tốc tuyệt đối:

$$velocity\_absolute = velocity\_relative + jetracer\_velocity\_ms$$

Trong đó:

- $velocity\_absolute$ là vận tốc của vật thể
- $jetracer\_velocity\_ms$ là vận tốc của Jetracer (Vận tốc được mô phỏng)

3.5. Cảnh báo va chạm phía sau và phản ứng tự động



Hình 3.14: Sơ đồ khối tổng quan về hoạt động hệ thống RCW

3.5.1. Phát hiện vật thể phía sau

Hệ thống sử dụng kết hợp camera và LiDAR để phát hiện các vật thể xuất hiện trong vùng phía sau xe. Camera đảm nhận vai trò nhận diện loại vật thể và vị trí trên ảnh, trong khi LiDAR cung cấp thông tin khoảng cách chính xác đến vật thể. Dữ liệu từ hai cảm biến được đồng bộ hóa, giúp xác định chính xác vị trí, khoảng cách và vận tốc của từng vật thể phía sau xe. Chỉ các vật thể nằm trong vùng góc phía sau (theo cấu hình hệ thống) mới được đưa vào danh sách kiểm tra nguy cơ va chạm.

Để xác định nguy cơ va chạm phía sau, hệ thống không chỉ dựa vào khoảng cách mà còn xét đến góc xuất hiện của vật thể so với xe. Cụ thể, các điều kiện được áp dụng như sau:

- Vùng nguy hiểm chính giữa: Nếu vật thể nằm trong khoảng cách nhỏ hơn 3 mét và góc quan sát từ -65° đến -45° (tương ứng với vùng chính giữa phía sau xe), hệ thống sẽ kích hoạt cảnh báo nguy hiểm trung tâm
- Vùng cảnh báo trái/phải:

  Nếu vật thể nằm trong khoảng cách nhỏ hơn 5 mét, hệ thống sẽ tiếp tục kiểm tra góc để xác định vật thể ở bên trái hay bên phải xe:
  - Nếu góc nhỏ hơn -45°, cảnh báo bên trái được kích hoạt.
  - Nếu góc lớn hơn -45°, cảnh báo bên phải được kích hoạt.

3.5.2. Tính toán, đánh giá và phản ứng tự động với nguy hiểm tiềm tàng

Sau khi phát hiện vật thể, hệ thống tiến hành các phép tính và phân tích để lượng hóa rủi ro.

- Tính toán chỉ số nguy hiểm (TTC):

  Tính toán Time-To-Collision (TTC): Dựa trên khoảng cách và vận tốc tiếp cận, hệ thống tính TTC để dự đoán thời gian còn lại trước khi xảy ra va chạm. Nếu TTC nhỏ hơn ngưỡng cho phép, hệ thống xác định có nguy cơ va chạm phía sau.

  $$TTC = distance / object\_velocity$$

  Trong đó:
  - Khoảng cách trực tiếp (D): Là cự ly tức thời từ xe đến vật thể, được tính bằng giá trị trung bình của các điểm LiDAR trên bề mặt của vật thể đó.
  - $object\_velocity$: Vận tốc tương đối của vật thể được xác định từ sự thay đổi khoảng cách của vật thể theo thời gian (vận tốc > 0 vật thể tiến lại gần, vận tốc < 0 vật thể đi ra xa).

  Chỉ số này cho biết thời gian dự kiến còn lại trước khi xảy ra va chạm nếu cả hai đối tượng tiếp tục di chuyển với vận tốc hiện tại.

- Đánh giá và phân cấp mức độ nguy hiểm được áp dụng:
    - Thông báo về TTC:

Mức độ cảnh báo	Điều kiện TTC	Màu sắc hiển thị	Ý nghĩa cảnh báo
CRITICAL	TTC < 1.0 s	Đỏ	Cực kỳ nguy hiểm, va chạm sắp xảy ra
DANGER	1.0 ≤ TTC < 2.0 s	Cam	Nguy hiểm, cần can thiệp gấp
WARNING	2.0 ≤ TTC < 3.0 s	Vàng	Cảnh báo, nên chú ý
CAUTION	TTC ≥ 3.0 s	Vàng nhạt/Xanh	Thận trọng, an toàn tạm thời

Bảng 3.1: Điều kiện thông báo nguy hiểm trên giao diện camera

    - Điều kiện đánh lái trái/phải và dừng xe:

Vận tốc jetracer	TTC	Góc so với trục xe	Cờ cảnh báo
> 0	< 2.0 s	< - 45 ° (camera_angle)	left_danger=True
> 0	< 2.0 s	≥ -45 ° (camera_angle)	right_danger=True
> 0	≥ 2.0 s	_	Không đánh dấu
≤ 0	–	135° → 180° và -180° → -135° (lidar_angle được chuẩn hóa)	flag_stop = True

Bảng 3.2: Điều kiện đánh lái hoặc phanh

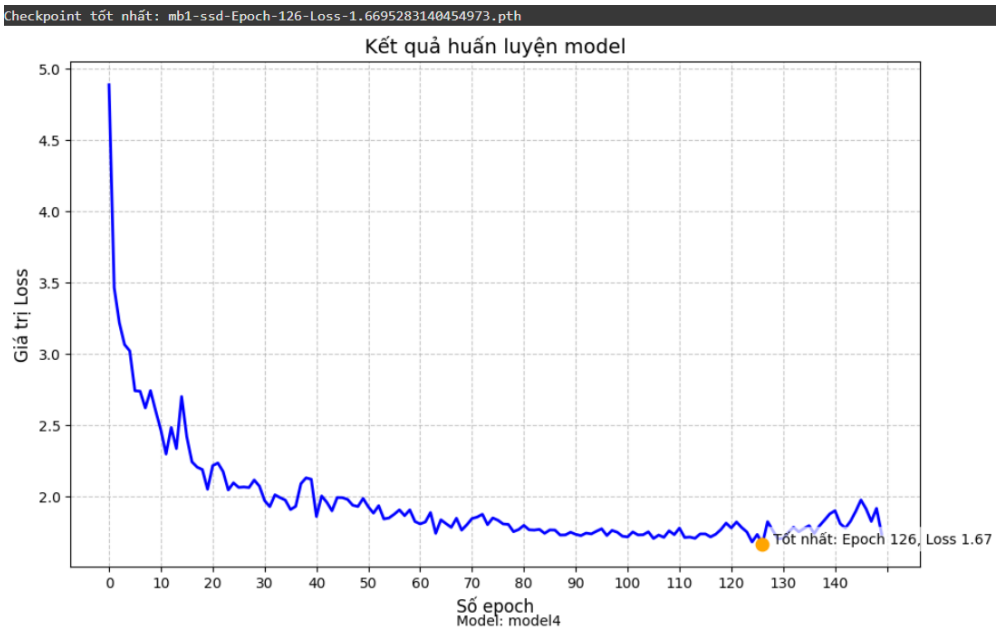
CHƯƠNG 4: KẾT QUẢ THỬ NGHIỆM VÀ ĐÁNH GIÁ

4.1. Mô hình huấn luyện, triển khai hệ thống và kế hoạch thử nghiệm hệ thống

4.1.1. Kết quả mô hình huấn luyện

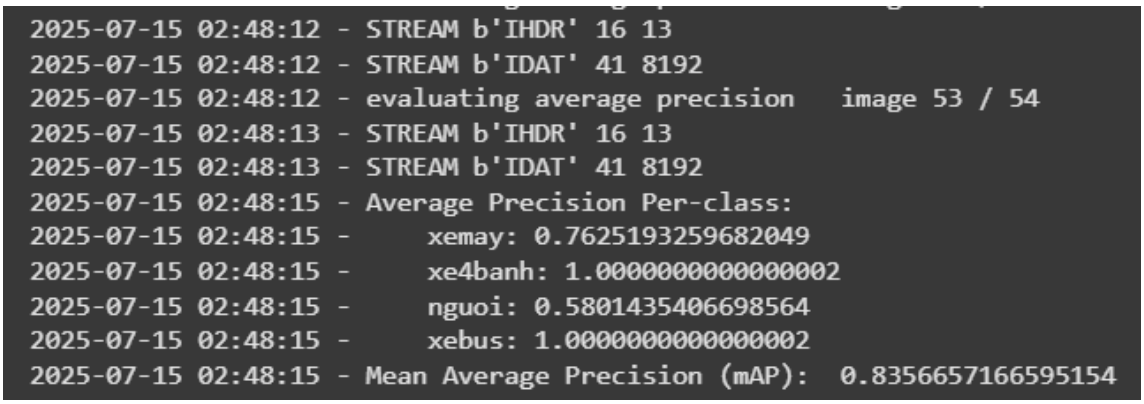
```
2025-07-13 07:18:05 - Epoch: 146, Step: 10/63, Avg Loss: 2.2416, Avg Regression Loss 0.5751, Avg Classification Loss: 1.6665
2025-07-13 07:18:06 - Epoch: 146, Step: 20/63, Avg Loss: 1.4956, Avg Regression Loss 0.3731, Avg Classification Loss: 1.1224
2025-07-13 07:18:07 - Epoch: 146, Step: 30/63, Avg Loss: 1.6679, Avg Regression Loss 0.4233, Avg Classification Loss: 1.2446
2025-07-13 07:18:08 - Epoch: 146, Step: 40/63, Avg Loss: 1.7079, Avg Regression Loss 0.4832, Avg Classification Loss: 1.2247
2025-07-13 07:18:09 - Epoch: 146, Step: 50/63, Avg Loss: 1.5591, Avg Regression Loss 0.4289, Avg Classification Loss: 1.1302
2025-07-13 07:18:10 - Epoch: 146, Step: 60/63, Avg Loss: 2.0697, Avg Regression Loss 0.4922, Avg Classification Loss: 1.5775
2025-07-13 07:18:11 - Epoch: 146, Training Loss: 1.7684, Training Regression Loss 0.4548, Training Classification Loss: 1.3136
2025-07-13 07:18:11 - Epoch: 146, Validation Loss: 1.8965, Validation Regression Loss 0.4238, Validation Classification Loss: 1.4727
2025-07-13 07:18:12 - Saved model /content/drive/MyDrive/models/model5/mb1-ssd-Epoch-146-Loss-1.8964866740362984.pth
2025-07-13 07:18:13 - Epoch: 147, Step: 10/63, Avg Loss: 1.9063, Avg Regression Loss 0.5071, Avg Classification Loss: 1.3992
2025-07-13 07:18:16 - Epoch: 147, Step: 20/63, Avg Loss: 1.6661, Avg Regression Loss 0.4778, Avg Classification Loss: 1.1883
2025-07-13 07:18:17 - Epoch: 147, Step: 30/63, Avg Loss: 1.7407, Avg Regression Loss 0.4573, Avg Classification Loss: 1.2835
2025-07-13 07:18:19 - Epoch: 147, Step: 40/63, Avg Loss: 1.4187, Avg Regression Loss 0.3598, Avg Classification Loss: 1.0589
2025-07-13 07:18:20 - Epoch: 147, Step: 50/63, Avg Loss: 1.6899, Avg Regression Loss 0.4156, Avg Classification Loss: 1.2743
2025-07-13 07:18:21 - Epoch: 147, Step: 60/63, Avg Loss: 1.9349, Avg Regression Loss 0.5381, Avg Classification Loss: 1.3968
2025-07-13 07:18:21 - Epoch: 147, Training Loss: 1.6991, Training Regression Loss 0.4481, Training Classification Loss: 1.2510
2025-07-13 07:18:22 - Epoch: 147, Validation Loss: 2.0169, Validation Regression Loss 0.5030, Validation Classification Loss: 1.5139
2025-07-13 07:18:22 - Saved model /content/drive/MyDrive/models/model5/mb1-ssd-Epoch-147-Loss-2.016913993018014.pth
2025-07-13 07:18:24 - Epoch: 148, Step: 10/63, Avg Loss: 1.5879, Avg Regression Loss 0.3755, Avg Classification Loss: 1.2124
2025-07-13 07:18:25 - Epoch: 148, Step: 20/63, Avg Loss: 1.6399, Avg Regression Loss 0.4375, Avg Classification Loss: 1.2024
2025-07-13 07:18:26 - Epoch: 148, Step: 30/63, Avg Loss: 1.9093, Avg Regression Loss 0.4846, Avg Classification Loss: 1.4248
2025-07-13 07:18:28 - Epoch: 148, Step: 40/63, Avg Loss: 1.7742, Avg Regression Loss 0.4188, Avg Classification Loss: 1.3554
2025-07-13 07:18:29 - Epoch: 148, Step: 50/63, Avg Loss: 2.0013, Avg Regression Loss 0.5287, Avg Classification Loss: 1.4726
2025-07-13 07:18:31 - Epoch: 148, Step: 60/63, Avg Loss: 1.4302, Avg Regression Loss 0.3750, Avg Classification Loss: 1.0552
2025-07-13 07:18:31 - Epoch: 148, Training Loss: 1.6871, Training Regression Loss 0.4251, Training Classification Loss: 1.2620
2025-07-13 07:18:32 - Epoch: 148, Validation Loss: 1.9218, Validation Regression Loss 0.4425, Validation Classification Loss: 1.4793
2025-07-13 07:18:33 - Saved model /content/drive/MyDrive/models/model5/mb1-ssd-Epoch-148-Loss-1.9218241316931588.pth
2025-07-13 07:18:34 - Epoch: 149, Step: 10/63, Avg Loss: 1.8672, Avg Regression Loss 0.4660, Avg Classification Loss: 1.4012
2025-07-13 07:18:36 - Epoch: 149, Step: 20/63, Avg Loss: 1.4527, Avg Regression Loss 0.3781, Avg Classification Loss: 1.0747
2025-07-13 07:18:37 - Epoch: 149, Step: 30/63, Avg Loss: 1.7876, Avg Regression Loss 0.4182, Avg Classification Loss: 1.3694
2025-07-13 07:18:38 - Epoch: 149, Step: 40/63, Avg Loss: 1.7378, Avg Regression Loss 0.4409, Avg Classification Loss: 1.2969
2025-07-13 07:18:39 - Epoch: 149, Step: 50/63, Avg Loss: 1.8058, Avg Regression Loss 0.4703, Avg Classification Loss: 1.3355
2025-07-13 07:18:40 - Epoch: 149, Step: 60/63, Avg Loss: 2.3732, Avg Regression Loss 0.7559, Avg Classification Loss: 1.6173
2025-07-13 07:18:41 - Epoch: 149, Training Loss: 1.8108, Training Regression Loss 0.4841, Training Classification Loss: 1.3267
2025-07-13 07:18:41 - Epoch: 149, Validation Loss: 2.0118, Validation Regression Loss 0.4816, Validation Classification Loss: 1.5301
2025-07-13 07:18:41 - Saved model /content/drive/MyDrive/models/model5/mb1-ssd-Epoch-149-Loss-2.011770640100752.pth
2025-07-13 07:18:41 - Task done, exiting program.
```

Hình 4.1: Ảnh log ghi nhận các chỉ số Training Loss và Validation Loss trong quá trình huấn luyện



Hình 4.2: Biểu đồ quá trình huấn luyện: Loss theo epoch qua 150 epoch

Đánh giá mô hình trên tập dữ liệu test gồm 54 ảnh (ảnh thuộc data test của 10% data) bằng cách áp dụng mô hình vào từng ảnh để dự đoán vị trí và lớp của các đối tượng.



Hình 4.3: Log thể hiện độ chính xác AP

Class	AP (Độ chính xác trung bình)	Đánh giá
xemay	76.3 %	Mô hình phát hiện xe máy khá tốt
xe4banh	100%	Hiệu xuất rất tốt khi phát hiện xe 4 bánh
nguoi	0.58 %	Hiệu xuất thấp khi phát hiện người
xebus	100%	Hiệu suất rất tốt khi phát hiện xe bus

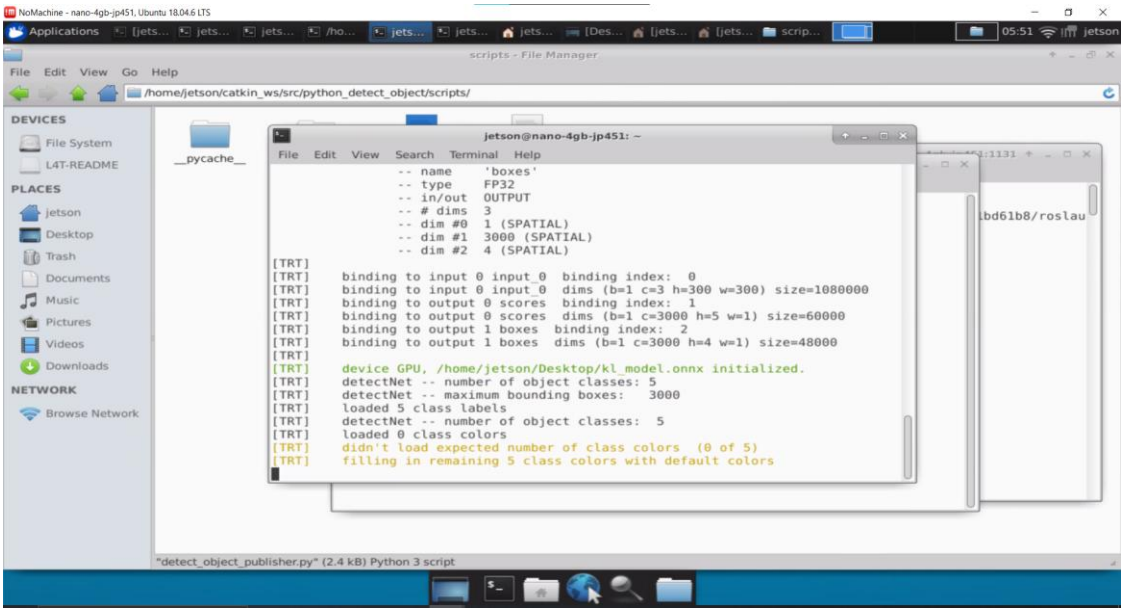
Bảng 4.1: AP của các class trong model áp dụng trong detect object

mAP = 83.6 % là trung bình cộng của tất cả các AP trên. Đây là chỉ số hiệu suất tổng thể của mô hình. Tổng thể mô hình vẫn duy trì hiệu quả cao, đặc biệt tốt cho xe bốn bánh và xe buýt.

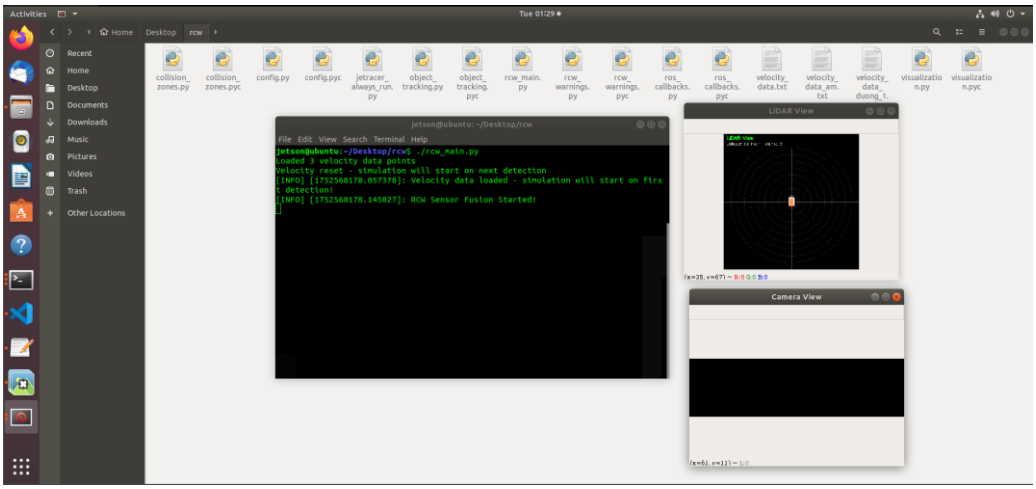
4.1.2. Chạy hệ thống và kịch bản thử nghiệm

Trong quá trình phát triển và thử nghiệm hệ thống RCW, máy ảo Ubuntu sẽ chạy chương trình chính (node xử lý RCW). Các node nhận diện đối tượng (detection) và thu thập dữ liệu cảm biến (camera, LiDAR) chạy trên Jetson thực tế. Dữ liệu nhận diện và cảm biến được publish qua ROS topic, sau đó máy ảo Ubuntu sẽ subscribe các topic này để nhận và xử lý dữ liệu (tracking, tính toán vận tốc, cảnh báo...).

Việc sử dụng máy ảo giúp tách biệt xử lý nặng, thuận tiện cho việc phát triển, kiểm thử và ghi log mà không ảnh hưởng đến hiệu năng của Jetson.



Hình 4.4: Chạy thành công lệnh load model, detect object
và pulish data JSON chứa dữ liệu các object đã đánh bounding box



Hình 4.5: Chạy thành công lệnh lấy dữ liệu các node phía jetracer
và thực hiện cảnh báo, điều khiển, mô phỏng chức năng RCW

Mục tiêu đánh giá khả năng của hệ thống RCW trên JetRacer ROS AI Kit trong việc phát hiện, theo dõi vật thể phía sau, tính toán khoảng cách, vận tốc, TTC và phản ứng cảnh báo/phanh tự động dựa trên dữ liệu cảm biến camera và LiDAR.

Kịch bản:

- Trong quá trình thử nghiệm, do JetRacer có cảm biến IMU đo gia tốc tức thời không ổn định hoặc thiếu chính xác, nên vận tốc xe được mô phỏng và lấy từ file velocity_data.txt. File này chứa các giá trị vận tốc phù hợp

với các kịch bản thử nghiệm, giúp kiểm tra logic hệ thống mà không phụ thuộc vào phần cứng thực tế.

- Gắn jetracer quay ngược ra sau xe máy và chạy trên đường có nhiều xe kiểm tra chức năng RCW.

  Trường hợp này, đã thu thập dữ liệu tình huống khi collect data, sử dụng rosbag đã record để mô phỏng, không chạy các node camera và lidar để tránh xung đột.
- Cho jetracer chạy tới (tương đương chạy lùi với xe máy, và không gắn trên xe máy) để kiểm tra chức năng phanh tự động khi lùi xe.

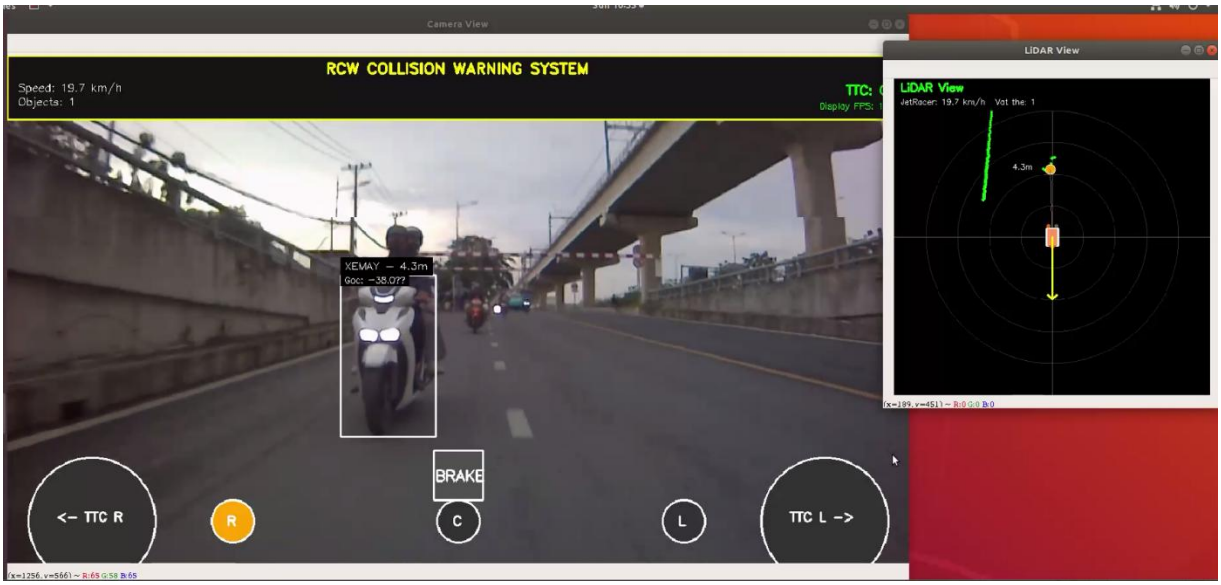
4.2. Kết quả thực nghiệm

Hệ thống cảnh báo va chạm phía sau trên JetRacer đã được kiểm thử thực tế với nhiều kịch bản khác nhau. Dưới đây là các kết quả tiêu biểu ghi nhận được qua quá trình thử nghiệm:

Nhận diện và cảnh báo vật thể phía sau xe

Hình 4.6: Không có object nào gần xe thu thập data và không detect được object nào

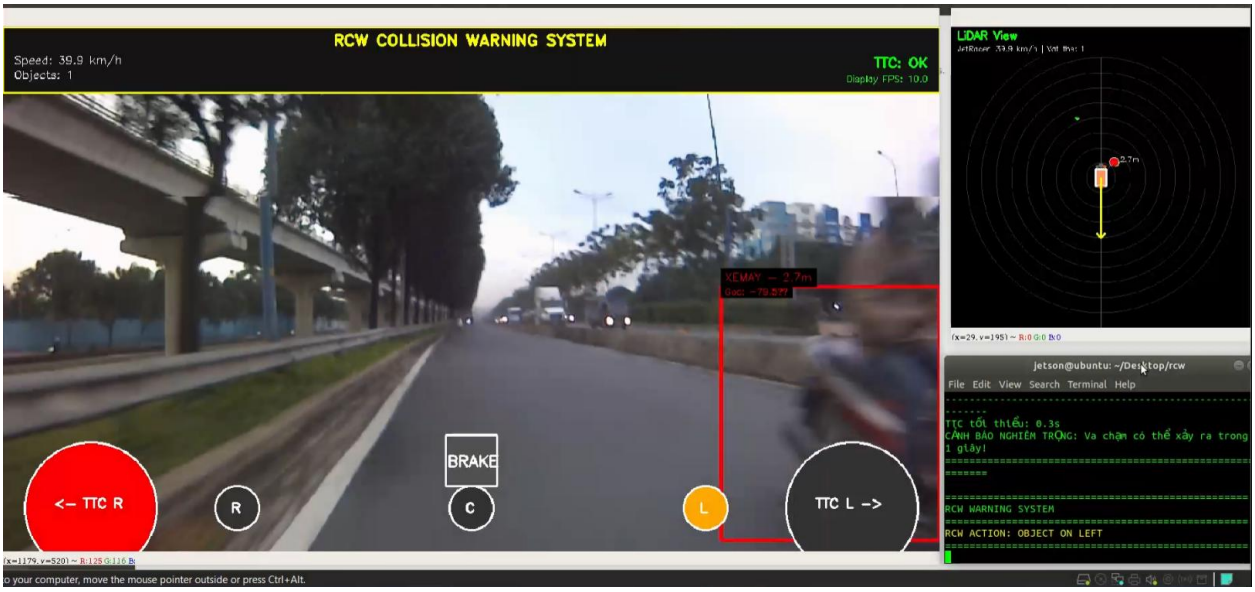
Hình 4.7: Nhận diện có object bên trái xe máy thu thập data (bên phải jetracer)



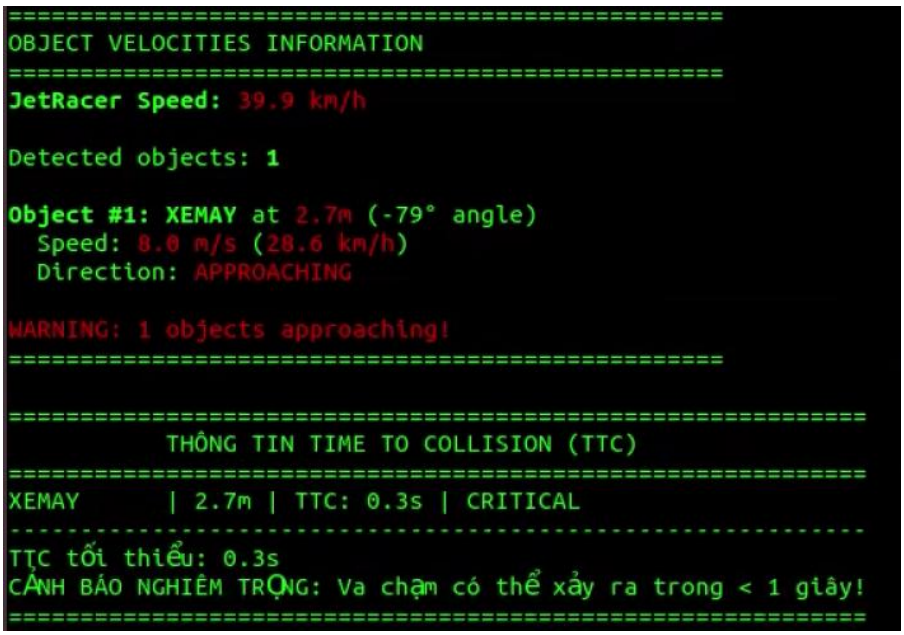
Hình 4.8: Nhận diện object bên phải xe máy thu thập data (bên trái jetracer)

Cảnh báo TTC và đề xuất đánh lái tránh va chạm

Hình 4.9: TTC trong ngưỡng cho phép (chưa cảnh báo và hành động)

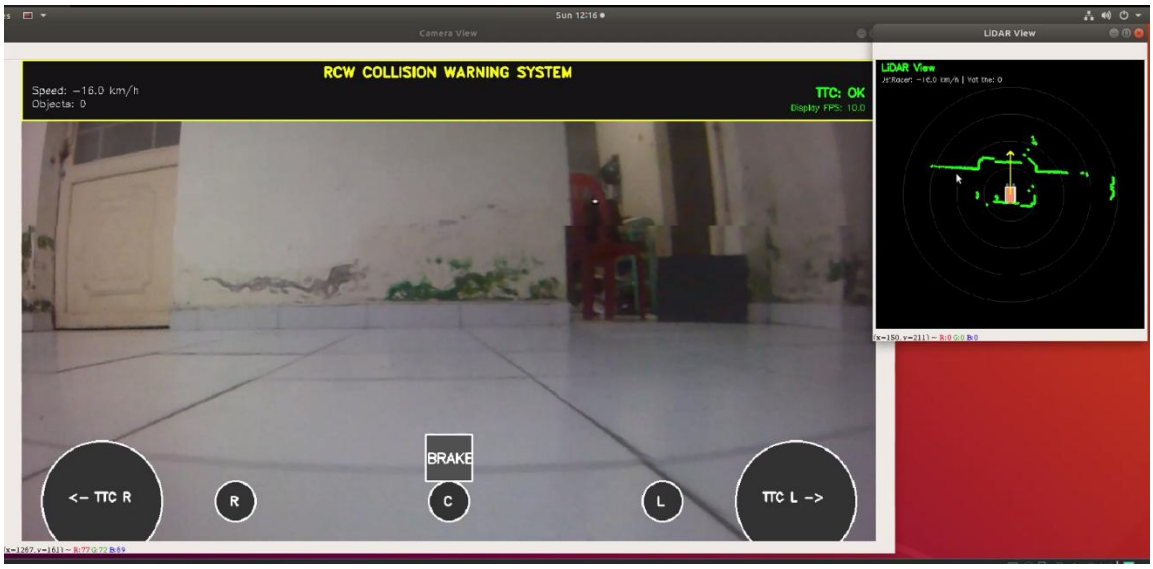


Hình 4.10: TTC không trong ngưỡng cho phép (đánh lái sang phải)



Hình 4.11: Log trong terminal hiện thông tin vật thể TTC < 2

Phanh tự động khi lùi xe



Hình 4.12: Lùi xe chưa có vật cản (Jetracer di chuyển tới và chưa dừng)

Hình 4.13: Lùi xe khi có vật cản (Jetracer di chuyển tới và dừng)

4.3. Đề xuất hướng phát triển

4.3.1. Hiện trạng chức năng của hệ thống

Mặc dù hệ thống đã được xây dựng và thử nghiệm với LiDAR 2D kết hợp camera, một số chức năng vẫn chưa hoàn thiện đầy đủ. Cụ thể, các chức năng đã triển khai gồm:

- Phanh tự động khi lùi xe: Hệ thống sử dụng dữ liệu LiDAR để phát hiện vật cản phía sau và tự động gửi lệnh phanh (dừng xe) khi có vật thể xuất hiện trong vùng nguy hiểm.
- Đánh lái dựa vào TTC và hướng vật thể: Hệ thống có khả năng phát hiện các vật thể phía sau xe, tính toán Time to Collision (TTC) và tự động đánh lái sang trái hoặc phải nếu phát hiện nguy hiểm. Tuy nhiên, chức năng này chưa phát hiện được vật thể phía trước xe, nên việc đánh lái chưa thực sự tối ưu và an toàn trong mọi tình huống.
- Cảnh báo vật thể:

  Hệ thống cảnh báo khi phát hiện vật thể ở phía sau xe với các ngưỡng khoảng cách:

  - < 3m ở vùng trung tâm
  - < 5m ở hai bên rìa

Các cảnh báo này được hiển thị trực quan trên giao diện và in ra terminal.

Các chức năng còn hạn chế/chưa hoàn thiện:

- Chưa phát hiện và xử lý vật thể phía trước xe để hỗ trợ đánh lái an toàn hơn.
- Chưa tích hợp các cảm biến bổ sung (camera chiều sâu, radar, siêu âm) để tăng độ tin cậy.
- Chưa áp dụng các thuật toán dự đoán chuyển động hoặc học sâu để nâng cao khả năng nhận diện và cảnh báo.

Kết luận:

Hệ thống hiện tại đã đáp ứng được các chức năng cơ bản về cảnh báo và phanh tự động khi lùi xe dựa trên dữ liệu LiDAR 2D. Tuy nhiên, để hoàn thiện và nâng cao hiệu quả, cần tiếp tục phát triển các chức năng phát hiện phía trước, tích hợp thêm cảm biến và cải tiến thuật toán xử lý dữ liệu đa nguồn.

4.3.2. Cải thiện hệ thống

Tối ưu hóa khai thác dữ liệu LiDAR 2D

- Tăng tần suất quét và độ phân giải góc (nếu phần cứng cho phép) để phát hiện vật thể nhỏ hoặc di chuyển nhanh tốt hơn.
- Lọc nhiễu và làm mượt dữ liệu: Áp dụng các thuật toán lọc (median filter, moving average) để giảm nhiễu từ LiDAR, giúp phát hiện vật thể ổn định hơn.

Cải tiến thuật toán phát hiện và cảnh báo

- Phân tích hình dạng vật thể từ dữ liệu LiDAR: Nhóm các điểm gần nhau thành cụm (clustering) để xác định số lượng và vị trí các vật thể, thay vì chỉ dựa vào khoảng cách nhỏ nhất.
- Cảnh báo thích ứng: Điều chỉnh ngưỡng cảnh báo (khoảng cách, TTC) dựa trên vận tốc thực tế của xe và mật độ vật thể phía sau.

Nâng cao giao diện cảnh báo và điều khiển

- Hiển thị trực quan vùng nguy hiểm: Vẽ rõ các vùng cảnh báo trên giao diện camera/LiDAR để người dùng dễ nhận biết.
- Tối ưu logic phanh và đánh lái: Cải tiến thuật toán để chỉ phanh hoặc đánh lái khi thực sự cần thiết, tránh phản ứng sai do nhiễu hoặc vật thể không nguy hiểm.

Ghi log và phân tích dữ liệu

Lưu lại dữ liệu vận hành: Ghi lại dữ liệu LiDAR, camera, vận tốc, cảnh báo để phân tích offline, phục vụ cải tiến thuật toán và đánh giá hiệu quả hệ thống.

TÀI LIỆU THAM KHẢO

[1]. *JetRacer ROS AI Kit, Web: JetRacer ROS AI Kit - Waveshare Wiki*

[2]. *Tips for using NoMachine on NVIDIA Jetson Nano, Web: NoMachine - Tips For Using NoMachine On NVIDIA Jetson Nano – Knowledge Base*

[3]. *Transfer learning using VGG-16 with Deep Convolutional Neural Network for Classifying Images, DOI:10.29322/IJSRP.9.10.2019.p9420*

[4]. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Web:1704.04861v1*

[5]. *SSD: Single Shot MultiBox Detector, Web: 1512.02325*

[6]. *Deploying Deep Learning, Web: GitHub - dusty-nv/jetson-inference: Hello AI World guide to deploying deep-learning inference networks and deep vision primitives with TensorRT and NVIDIA Jetson.*

[7]. *Rear Collision Warning: ADAS Features Explained, Web: Rear Collision Warning: ADAS Features Explained*

[8]. *Blog ROS (Robot Operating System), Web: ROS (Robot Operating System) là gì?*

[9]. *Label Studio is a modern, multi-modal data annotation tool, Web: GitHub - HumanSignal/labelImg.*