

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN TỬ VIỄN THÔNG



BÁO CÁO CUỐI KỲ
MÔN HỌC: CHUYÊN ĐỀ 2

ĐỀ TÀI:

NGHIÊN CỨU VÀ XÂY DỰNG HỆ THỐNG
GIÁM SÁT LƯU LƯỢNG GIAO THÔNG
SỬ DỤNG YOLOv8 VÀ THUẬT TOÁN LINE CROSSING

Giảng viên hướng dẫn:	TS. Nguyễn Văn Hiếu
Nhóm sinh viên:	Nhóm 18
Sinh viên thực hiện:	1. Trần Nguyên Vũ (106210205) 2. Trương Văn An (106210206) 3. Lê Thị Ánh Trinh (106210255)

Đà Nẵng, Tháng 12 năm 2025

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành đến thầy Nguyễn Văn Hiếu đã tận tình hướng dẫn, định hướng và tạo điều kiện thuận lợi để nhóm hoàn thành đồ án này. Những kiến thức quý báu thầy truyền đạt là nền tảng vững chắc để chúng em thực hiện đề tài và phát triển bản thân trong tương lai.

BẢNG PHÂN CÔNG NHIỆM VỤ

(Cam kết: Tỉ lệ đóng góp được sự đồng thuận của 100% thành viên)

Bảng 1: Chi tiết phân công và mức độ đóng góp

STT	Thành viên	Nội dung thực hiện chi tiết	Tỉ lệ
1	Trần Nguyên Vũ (Trưởng nhóm)	- Nghiên cứu cơ sở lý thuyết về CNN, YOLOv8 và Object Tracking (BoTSORT). - Tìm hiểu thuật toán về bài toán cắt vạch. - Tổng hợp kết quả, tính toán độ chính xác và soạn thảo báo cáo.	33%
2	Trương Văn An	- Thu thập dữ liệu video dataset. - Thiết lập môi trường ảo, quản lý version thư viện. - Triển khai source code lên GitHub. - Viết tài liệu hướng dẫn cài đặt và vận hành.	34%
3	Lê Thị Ánh Trinh	- Thiết kế kiến trúc tổng thể. - Tích hợp mô hình YOLOv8 vào giao diện Streamlit. - Thực hiện kịch bản kiểm thử. - Debug và tối ưu hóa FPS.	33%

Mục lục

1 TỔNG QUAN VỀ ĐỀ TÀI	6
1.1 Đặt vấn đề	6
1.2 Mục tiêu đề tài	6
1.3 Phạm vi nghiên cứu	6
1.4 Điểm mới của đề tài	6
2 CƠ SỞ LÝ THUYẾT	7
2.1 Tổng quan về Mô hình YOLOv8	7
2.1.1 Kiến trúc mạng (Network Architecture)	7
2.2 Mạng tích chập CNN	7
2.2.1 Khái niệm	7
2.2.2 Cấu trúc cơ bản	7
2.2.3 Phép tích chập	8
2.2.4 Vai trò của CNN trong YOLO	8
2.3 Loss Function trong YOLOv8	8
2.3.1 Tổng quan	8
2.3.2 Bounding Box Loss – CIoU Loss	8
2.3.3 Classification Loss – Binary Cross Entropy	8
2.3.4 Distribution Focal Loss (DFL)	9
2.4 Thuật toán BoT-SORT	9
2.4.1 Giới thiệu	9
2.4.2 Kiến trúc tổng thể	9
2.4.3 Mô hình chuyển động – Kalman Filter	9
2.4.4 Mô hình ngoại hình – ReID	9
2.4.5 Chiến lược ByteTrack	9
2.4.6 Gán dữ liệu – Hungarian Algorithm	10
2.5 Kalman Filter	10
2.5.1 Mô hình trạng thái	10
2.5.2 Bước dự đoán (Prediction)	10
2.5.3 Mô hình đo (Measurement Model)	11
2.5.4 Bước cập nhật (Update)	11
2.5.5 Trưởng hợp mắt detection	11

2.5.6 Vai trò của Kalman Filter	11
3 MÔ TẢ VÀ GIẢI QUYẾT BÀI TOÁN	12
3.1 Mô hình hệ thống (System Model)	12
3.2 Mô hình tổng quát	12
3.3 Mô hình hóa bài toán dưới dạng các công thức toán học	13
3.3.1 Mô hình hóa bài toán tổng quát	13
3.3.2 Mô hình hóa bài toán phát hiện đối tượng	13
3.3.3 Mô hình hóa bài toán theo dõi đối tượng	14
3.3.4 Thuật toán Line crossing	15
3.4 Phương pháp giải quyết bài toán	15
3.4.1 Phương pháp: Tracking-by-Detection	15
3.4.2 Phát hiện đối tượng (Detection)	15
3.4.3 So khớp dữ liệu (Data Association)	16
3.4.4 Cập nhật trạng thái đối tượng	16
3.4.5 Xử lý các trường hợp đặc biệt	16
3.4.6 Tổng kết quy trình	17
3.4.7 Thuật toán Line Crossing	17
3.5 Đánh giá độ phức tạp thuật toán	18
3.6 Phân tích hiệu năng từng module	19
4 KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ	20
4.1 Môi trường thử nghiệm	20
4.2 Mô tả dữ liệu đầu ra	20
4.3 Các chỉ số đánh giá (Metrics)	20
4.3.1 Định nghĩa Metrics	20
4.3.2 Kết quả đo đạc trên nhiều video	21
4.3.3 Phân tích nguyên nhân sai số	21
4.4 Hình ảnh minh họa kết quả	22
4.5 So sánh và Đánh giá tổng quan	22
4.5.1 So sánh thuật toán	22
4.5.2 So sánh Line Crossing vs ROI-based	22
4.5.3 So sánh Tracker: BoTSORT vs Alternatives	23
4.5.4 So sánh Môi trường và Ngôn ngữ	23
5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	24

5.1	Kết luận	24
5.2	Hướng phát triển	24

1 TỔNG QUAN VỀ ĐÈM TÀI

1.1 Đặt vấn đề

Ùn tắc giao thông và quản lý trật tự đô thị đang là thách thức lớn tại các thành phố lớn như Đà Nẵng, Hà Nội, TP.HCM. Việc thu thập dữ liệu lưu lượng phương tiện (số lượng xe máy, ô tô, xe tải...) là tiền đề quan trọng để quy hoạch đường sá và điều khiển đèn tín hiệu thông minh. Các phương pháp truyền thống như sử dụng vòng từ (loop detectors) hay nhân lực đếm thủ công thường bộc lộ nhiều hạn chế về chi phí lắp đặt, bảo trì và độ ổn định.

1.2 Mục tiêu đề tài

Dự án hướng tới việc xây dựng một giải pháp phần mềm sử dụng Thị giác máy tính (Computer Vision) với các mục tiêu cụ thể:

1. Ứng dụng mô hình học sâu (Deep Learning) tiên tiến YOLOv8 để phát hiện phương tiện.
2. Xây dựng giải pháp đếm xe dựa trên hành vi di chuyển (Line Crossing) thay vì chỉ đếm sự hiện diện.
3. Cung cấp giao diện trực quan (Dashboard) cho người quản lý.

1.3 Phạm vi nghiên cứu

- **Đối tượng:** Các phương tiện giao thông đường bộ cơ bản (Xe máy, Ô tô con, Xe buýt, Xe tải).
- **Môi trường:** Hệ thống chạy trên máy tính cá nhân (localhost) thời gian thực.

1.4 Điểm mới của đề tài

Khác với các bài toán nhận diện cơ bản, dự án này có các điểm mới:

- **Tương tác thực tế:** Tích hợp công cụ vẽ vạch ảo (Virtual Line) ngay trên giao diện Web, cho phép người dùng tùy biến vị trí giám sát mà không cần can thiệp code.
- **Thuật toán Vector:** Thay thế phương pháp vùng quan tâm (ROI) truyền thống bằng thuật toán hình học vector (Cross Product), giúp giải quyết bài toán đếm trùng lặp khi xe đứng yên tại vạch đèn đỏ.

2 CƠ SỞ LÝ THUYẾT

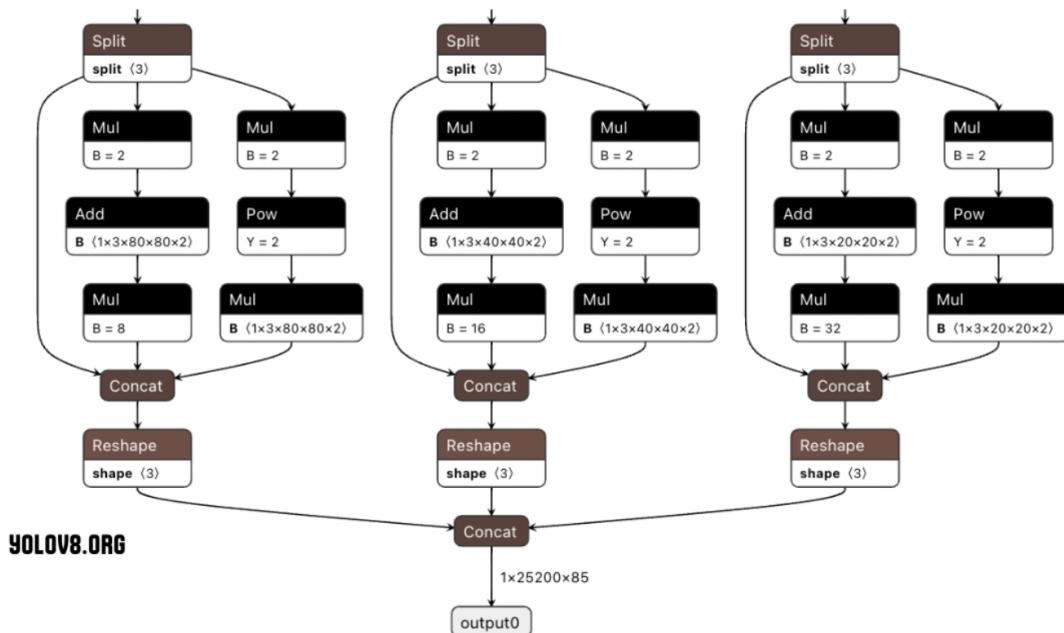
2.1 Tổng quan về Mô hình YOLOv8

YOLOv8 là mô hình mạng nơ-ron tích chập (CNN) tiên tiến nhất được phát hành bởi Ultralytics.

2.1.1 Kiến trúc mạng (Network Architecture)

Kiến trúc của YOLOv8 bao gồm 3 thành phần chính:

- **Backbone:** Sử dụng kiến trúc *CSPDarknet53* cải tiến với module **C2f** giúp cải thiện luồng gradient.
- **Neck:** Sử dụng kiến trúc **PANet** để trộn lối đặc trưng.
- **Head:** Sử dụng kiến trúc **Decoupled Head** và cơ chế **Anchor-free**.



Hình 1: Kiến trúc tổng quát của YOLOv8

2.2 Mạng tích chập CNN

2.2.1 Khái niệm

Mạng tích chập (Convolutional Neural Network – CNN) là một dạng mạng nơ-ron nhân tạo được thiết kế đặc biệt cho dữ liệu dạng lưới, tiêu biểu là ảnh số. CNN khai thác mối quan hệ cục bộ giữa các điểm ảnh thông qua phép tích chập, giúp giảm số lượng tham số và giữ được cấu trúc không gian của dữ liệu.

2.2.2 Cấu trúc cơ bản

Một CNN điển hình gồm các thành phần:

- Lớp tích chập (Convolution layer)

- Hàm kích hoạt (Activation function)
- Lớp gộp (Pooling layer)
- Lớp kết nối đầy đủ (Fully Connected layer)

Quá trình trích xuất đặc trưng diễn ra từ các lớp nông đến các lớp sâu, lần lượt học các đặc trưng từ đơn giản (cạnh, góc) đến phức tạp (hình dạng, đối tượng).

2.2.3 Phép tích chập

Với ảnh đầu vào X và kernel K kích thước $k \times k$, phép tích chập được biểu diễn như sau:

$$Y(i, j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} X(i+m, j+n) \cdot K(m, n) \quad (1)$$

2.2.4 Vai trò của CNN trong YOLO

Trong YOLOv8, CNN được sử dụng làm backbone để trích xuất đặc trưng ảnh, cung cấp các feature map cho các tầng phát hiện đối tượng.

2.3 Loss Function trong YOLOv8

2.3.1 Tổng quát

YOLOv8 là mô hình phát hiện đối tượng một giai đoạn (one-stage detector), do đó hàm mất mát được thiết kế nhằm tối ưu đồng thời nhiều nhiệm vụ: định vị khung bao, phân loại đối tượng và độ chính xác biên của bounding box.

Hàm mất mát tổng quát được biểu diễn như sau:

$$\mathcal{L}_{total} = \lambda * \mathcal{L}_{box} + \lambda * \mathcal{L}_{cls} + \lambda * \mathcal{L}_{dfl} \quad (2)$$

2.3.2 Bounding Box Loss – CIoU Loss

YOLOv8 sử dụng Complete IoU (CIoU) để đo mức độ sai lệch giữa bounding box dự đoán và ground truth:

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v \quad (3)$$

Trong đó ρ là khoảng cách giữa tâm hai bounding box, c là đường chéo nhỏ nhất bao cả hai khung, và v thể hiện sự khác biệt về tỷ lệ khung hình.

2.3.3 Classification Loss – Binary Cross Entropy

Hàm mất mát phân loại được tính bằng Binary Cross Entropy (BCE):

$$\mathcal{L}_{cls} = - \sum_i [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \quad (4)$$

2.3.4 Distribution Focal Loss (DFL)

YOLOv8 không dự đoán trực tiếp tọa độ bounding box mà dự đoán phân phối xác suất của khoảng cách từ tâm box đến bốn cạnh. Distribution Focal Loss được dùng để học phân phối này:

$$\mathcal{L}_{DFL} = - \sum_i y_i \log(p_i) \quad (5)$$

DFL giúp cải thiện độ chính xác biên và làm cho bounding box dự đoán ổn định hơn.

2.4 Thuật toán BoT-SORT

2.4.1 Giới thiệu

BoT-SORT (Bag-of-Tricks SORT) là thuật toán theo dõi đa đối tượng (Multi-Object Tracking) được phát triển dựa trên SORT, DeepSORT và ByteTrack. Thuật toán này thường được kết hợp với YOLOv8 để theo dõi đối tượng theo thời gian thực.

2.4.2 Kiến trúc tổng thể

Hệ thống theo dõi sử dụng BoT-SORT bao gồm các thành phần:

- Bộ phát hiện đối tượng (YOLOv8)
- Mô hình chuyển động Kalman Filter
- Mô hình nhận dạng ngoại hình (Re-Identification)
- Thuật toán gán dữ liệu Hungarian

2.4.3 Mô hình chuyển động – Kalman Filter

Vector trạng thái của một đối tượng được biểu diễn như sau:

$$\mathbf{x} = [c_x, c_y, w, h, \dot{c}_x, \dot{c}_y, \dot{w}, \dot{h}] \quad (6)$$

Kalman Filter giúp dự đoán vị trí đối tượng ở frame tiếp theo, đặc biệt hiệu quả trong trường hợp che khuất.

2.4.4 Mô hình ngoại hình – ReID

BoT-SORT sử dụng vector đặc trưng ngoại hình để đo độ tương đồng giữa các đối tượng:

$$d_{app} = 1 - \cos(\mathbf{f}_1, \mathbf{f}_2) \quad (7)$$

2.4.5 Chiến lược ByteTrack

Các detection được chia thành hai nhóm: độ tin cậy cao và độ tin cậy thấp. Việc gán track được thực hiện theo hai bước nhằm giảm hiện tượng mất đối tượng khi bị che khuất.

2.4.6 Gán dữ liệu – Hungarian Algorithm

Ma trận chi phí được xây dựng từ thông tin chuyển động và ngoại hình:

$$Cost = \alpha d_{motion} + \beta d_{appearance} \quad (8)$$

Thuật toán Hungarian được sử dụng để tối ưu quá trình gán ID cho các đối tượng qua các frame.

2.5 Kalman Filter

Trong bài toán theo dõi đối tượng (object tracking), việc chỉ sử dụng kết quả phát hiện (detection) từ các mô hình như YOLO có thể dẫn đến nhiều vị trí bounding box, chuyển động không mượt giữa các khung hình và mất detection tạm thời. Kalman Filter được sử dụng nhằm khắc phục các vấn đề này thông qua cơ chế dự đoán và hiệu chỉnh trạng thái đối tượng theo thời gian.

2.5.1 Mô hình trạng thái

Trạng thái của một đối tượng tại thời điểm t được mô hình hóa bởi vector trạng thái:

$$\mathbf{x}_t = \begin{bmatrix} x_t \\ y_t \\ w_t \\ h_t \\ \dot{x}_t \\ \dot{y}_t \\ \dot{w}_t \\ \dot{h}_t \end{bmatrix} \quad (9)$$

Trong đó (x_t, y_t) là tọa độ tâm bounding box, (w_t, h_t) là chiều rộng và chiều cao, còn $(\dot{x}_t, \dot{y}_t, \dot{w}_t, \dot{h}_t)$ là các thành phần vận tốc.

2.5.2 Bước dự đoán (Prediction)

Kalman Filter giả định đối tượng chuyển động với vận tốc gần như không đổi giữa các khung hình liên tiếp. Phương trình dự đoán được viết như sau:

$$\mathbf{x}_{t+1}^- = A\mathbf{x}_t \quad (10)$$

Trong đó A là ma trận chuyển trạng thái:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (11)$$

2.5.3 Mô hình đo (Measurement Model)

Từ mô hình phát hiện đối tượng, ta chỉ quan sát được vị trí và kích thước bounding box. Vector đo tại thời điểm $t + 1$:

$$\mathbf{z}_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ w_{t+1} \\ h_{t+1} \end{bmatrix} \quad (12)$$

Ma trận quan sát H :

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (13)$$

2.5.4 Bước cập nhật (Update)

Sau khi có measurement, trạng thái được hiệu chỉnh theo công thức:

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + K(\mathbf{z}_{t+1} - H\mathbf{x}_t) \quad (14)$$

Trong đó K là Kalman Gain, được tính bởi:

$$K = P^- H^T (HP^- H^T + R)^{-1} \quad (15)$$

P^- là ma trận hiệp phương sai dự đoán và R là ma trận nhiễu đo.

2.5.5 Trường hợp mất detection

Khi không có measurement tại thời điểm $t + 1$, Kalman Filter chỉ thực hiện bước dự đoán:

$$\mathbf{x}_{t+1} = A\mathbf{x}_t \quad (16)$$

Điều này cho phép duy trì track và ID của đối tượng trong một số khung hình.

2.5.6 Vai trò của Kalman Filter

Kalman Filter giúp làm mượt quỹ đạo chuyển động, giảm nhiễu từ detection, duy trì theo dõi ổn định khi detection bị mất tạm thời và cải thiện hiệu quả gán dữ liệu trong các hệ thống tracking như SORT và DeepSORT.

3 MÔ TẢ VÀ GIẢI QUYẾT BÀI TOÁN

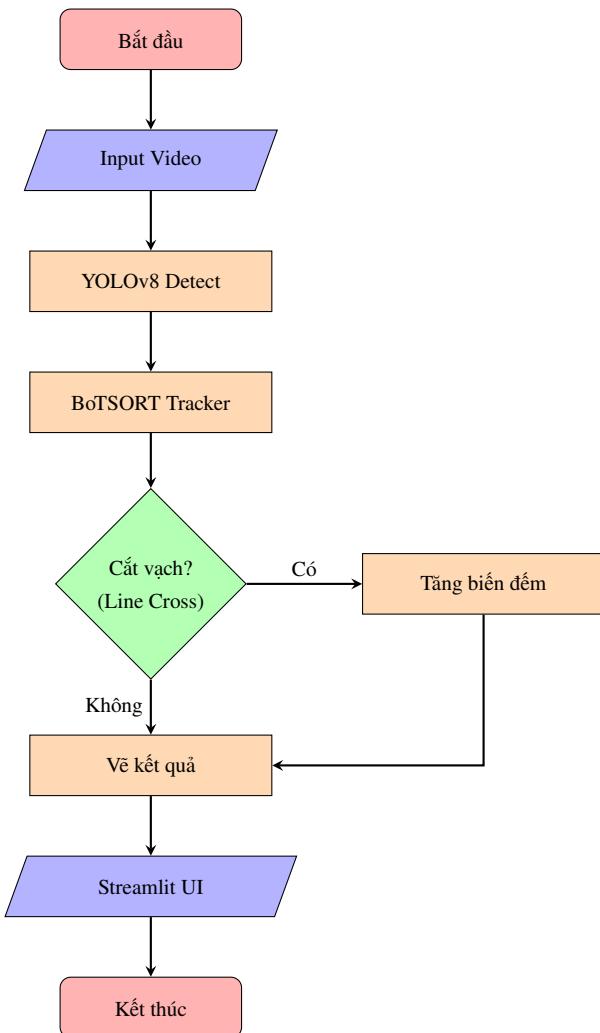
3.1 Mô hình hệ thống (System Model)

3.2 Mô hình tổng quát

Hệ thống trong dự án được xây dựng theo kiến trúc *tracking-by-detection*, bao gồm hai thành phần chính:

- Phát hiện đối tượng (Object Detection):** Sử dụng mô hình YOLOv8 để phát hiện các đối tượng trong từng khung hình video, đầu ra là các hộp bao (bounding boxes), nhãn lớp và độ tin cậy.
- Theo dõi đối tượng (Object Tracking):** Dựa trên các kết quả phát hiện, thuật toán theo dõi thực hiện việc gán ID và duy trì định danh cho các đối tượng qua các khung hình liên tiếp.

Dữ liệu đầu vào của hệ thống là video hoặc luồng hình ảnh từ webcam, và đầu ra là video đã được gán hộp bao cùng với ID cho từng đối tượng.



Hình 2: Sơ đồ khái niệm hoạt động của hệ thống

Cách thức hoạt động

Quy trình hoạt động của hệ thống được mô tả như sau:

- Trích xuất từng khung hình từ video đầu vào.

- Áp dụng mô hình YOLOv8 để phát hiện các đối tượng trong khung hình.
- Sử dụng thuật toán theo dõi để gán các phát hiện mới với các đối tượng đã tồn tại dựa trên độ tương đồng vị trí.
- Cập nhật trạng thái và ID của từng đối tượng.
- Hiển thị và lưu trữ kết quả theo dõi.

3.3 Mô hình hóa bài toán dưới dạng các công thức toán học

3.3.1 Mô hình hóa bài toán tổng quát

Bài toán trong dự án được xác định là bài toán theo dõi đa đối tượng trong video (Multi-Object Tracking – MOT). Đầu vào của bài toán là một chuỗi các khung hình video theo thời gian:

$$\mathcal{I} = \{I_1, I_2, \dots, I_T\}$$

Mục tiêu của bài toán là xác định tập các đối tượng xuất hiện trong video và duy trì định danh duy nhất cho mỗi đối tượng xuyên suốt chuỗi thời gian quan sát.

3.3.2 Mô hình hóa bài toán phát hiện đối tượng

Tại mỗi thời điểm t , với khung hình đầu vào I_t , mô hình phát hiện đối tượng thực hiện một phép ánh xạ:

$$f_\theta : I_t \rightarrow \mathcal{D}_t$$

Trong đó tập các phát hiện tại thời điểm t được biểu diễn như sau:

$$\mathcal{D}_t = \{(b_t^i, c_t^i, p_t^i) \mid i = 1, 2, \dots, N_t\}$$

với:

- $b_t^i = (x_t^i, y_t^i, w_t^i, h_t^i)$ là hộp bao (bounding box) của đối tượng thứ i ,
- c_t^i là nhãn lớp của đối tượng,
- $p_t^i \in [0, 1]$ là xác suất tin cậy của dự đoán,
- N_t là số lượng đối tượng được phát hiện tại thời điểm t .

Hàm mất mát (Loss Function) của YOLOv8

YOLOv8 được huấn luyện bằng tổng loss:

$$L = L_{box} + L_{cls} + L_{obj} \quad (17)$$

Trong đó:

Loss bounding box (IoU-based loss)

Loss bounding box đo mức độ sai lệch giữa bounding box dự đoán \hat{b} và bounding box ground-truth b , thường dựa trên chỉ số IoU (Intersection over Union):

$$L_{box} = 1 - \text{IoU}(b, \hat{b}) \quad (18)$$

Loss phân lớp (Classification loss)

Loss phân lớp đánh giá mức độ chính xác của xác suất dự đoán lớp đối tượng. Với mỗi lớp c , hàm mất mát được biểu diễn dưới dạng cross-entropy:

$$L_{cls} = -\sum_c \log(\hat{c}) \quad (19)$$

trong đó \hat{c} là xác suất dự đoán của lớp đúng.

Loss objectness

Loss objectness đánh giá khả năng mô hình xác định sự tồn tại của đối tượng trong bounding box. Hàm mất mát này được mô hình hóa bằng binary cross-entropy:

$$L_{obj} = -[y \log(p) + (1-y) \log(1-p)] \quad (20)$$

Trong đó $y \in \{0, 1\}$ là nhãn thể hiện sự tồn tại của đối tượng và p là xác suất dự đoán objectness.

3.3.3 Mô hình hóa bài toán theo dõi đối tượng

Giả sử tại thời điểm t , hệ thống đang duy trì một tập các đối tượng đang được theo dõi:

$$\mathcal{T}_t = \{T_1^t, T_2^t, \dots, T_M^t\}$$

Mỗi đối tượng (track) T_j^t được biểu diễn bởi:

$$T_j^t = (\hat{b}_j^t, ID_j)$$

trong đó:

- \hat{b}_j^t là vị trí ước lượng của đối tượng tại thời điểm t ,
- ID_j là định danh duy nhất của đối tượng.

Tại thời điểm $t+1$, mô hình phát hiện tạo ra tập các hộp bao mới:

$$\mathcal{D}_{t+1} = \{b_{t+1}^1, b_{t+1}^2, \dots, b_{t+1}^N\}$$

Bài toán theo dõi được mô hình hóa như một bài toán gán tối ưu giữa các track hiện có và các phát hiện mới, thông qua việc tối thiểu hóa tổng hàm chi phí:

$$\min \sum_{i,j} \text{Cost}(b_{t+1}^i, \hat{b}_j^t)$$

Trong đó hàm chi phí được xác định dựa trên độ chồng lấn không gian giữa hai hộp bao:

$$\text{Cost}(b_{t+1}^i, \hat{b}_j^t) = 1 - \text{IoU}(b_{t+1}^i, \hat{b}_j^t)$$

Chỉ số Intersection over Union (IoU) được định nghĩa như sau:

$$\text{IoU}(A, B) = \frac{A \cap B}{A \cup B}$$

Việc tối ưu hàm chi phí cho phép hệ thống gán các phát hiện mới với các track phù hợp nhất, từ đó duy trì định danh đối tượng xuyên suốt chuỗi khung hình.

3.3.4 Thuật toán Line crossing

- Gọi vạch kẻ là đoạn thẳng $L(A, B)$.
- Quỹ đạo xe là đoạn thẳng $V(P_{prev}, P_{curr})$.

Sử dụng công thức tích có hướng (Cross Product):

$$CCW(A, B, C) = (B_x - A_x)(C_y - A_y) - (B_y - A_y)(C_x - A_x) \quad (21)$$

Điều kiện cắt nhau:

$$(CCW(A, B, P_{prev}) \times CCW(A, B, P_{curr}) < 0) \wedge (CCW(P_{prev}, P_{curr}, A) \times CCW(P_{prev}, P_{curr}, B) < 0) \quad (22)$$

3.4 Phương pháp giải quyết bài toán

3.4.1 Phương pháp: Tracking-by-Detection

Dự án áp dụng phương pháp *Tracking-by-Detection*, trong đó quá trình theo dõi đối tượng được thực hiện dựa trên kết quả phát hiện ở từng khung hình độc lập. Ý tưởng tổng quát được mô tả như sau:

$$I_t \xrightarrow{\text{Detection}} \mathcal{D}_t \xrightarrow{\text{Tracking}} \mathcal{T}_t$$

Trong đó:

- I_t là khung hình đầu vào tại thời điểm t
- \mathcal{D}_t là tập các bounding box được phát hiện
- \mathcal{T}_t là tập các đối tượng đang được theo dõi kèm ID

Phương pháp này được lựa chọn vì YOLOv8 có khả năng phát hiện đối tượng chính xác và nhanh, không yêu cầu huấn luyện riêng cho bộ theo dõi và dễ triển khai trong các hệ thống thời gian thực.

3.4.2 Phát hiện đối tượng (Detection)

Tại mỗi khung hình I_t , mô hình YOLOv8 thực hiện ánh xạ:

$$f_\theta(I_t) = \{(b_t^i, c_t^i, p_t^i)\}_{i=1}^{N_t}$$

Trong đó:

- $b_t^i = (x_t^i, y_t^i, w_t^i, h_t^i)$ là bounding box thứ i
- c_t^i là nhãn lớp
- p_t^i là độ tin cậy
- N_t là số đối tượng được phát hiện

3.4.3 So khớp dữ liệu (Data Association)

Giả sử tại thời điểm t tồn tại M track đang hoạt động:

$$\mathcal{T}_t = \{\mathbf{x}_t^1, \mathbf{x}_t^2, \dots, \mathbf{x}_t^M\}$$

Tại thời điểm $t+1$, YOLOv8 phát hiện N bounding box:

$$\mathcal{D}_{t+1} = \{\mathbf{b}_{t+1}^1, \mathbf{b}_{t+1}^2, \dots, \mathbf{b}_{t+1}^N\}$$

Độ tương đồng giữa track i và detection j được tính bằng chỉ số IoU:

$$\text{IoU}_{ij} = \frac{\text{Area}(\mathbf{x}_t^i \cap \mathbf{b}_{t+1}^j)}{\text{Area}(\mathbf{x}_t^i \cup \mathbf{b}_{t+1}^j)}$$

Từ đó xây dựng hàm chi phí:

$$C_{ij} = 1 - \text{IoU}_{ij}$$

Ma trận chi phí được biểu diễn như sau:

$$\mathbf{C} = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1N} \\ C_{21} & C_{22} & \cdots & C_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ C_{M1} & C_{M2} & \cdots & C_{MN} \end{bmatrix}$$

Bài toán gán được mô hình hóa thành bài toán tối ưu:

$$\min_{\pi} \sum_{i=1}^M C_{i,\pi(i)}$$

trong đó $\pi(i)$ là detection được gán cho track i .

3.4.4 Cập nhật trạng thái đối tượng

Sau khi hoàn tất việc gán, trạng thái của track được cập nhật như sau:

- Nếu track i được gán detection j :

$$\mathbf{x}_{t+1}^i = \mathbf{b}_{t+1}^j$$

- Sử dụng Kalman Filter:

$$\mathbf{x}_{t+1}^i = A\mathbf{x}_t^i + K(\mathbf{z}_{t+1}^j - H\mathbf{x}_t^i)$$

3.4.5 Xử lý các trường hợp đặc biệt

- **Khởi tạo track mới:** Nếu detection không được gán cho bất kỳ track nào:

$$T_{\text{new}} = (\mathbf{b}_{t+1}, ID_{\text{new}})$$

- **Xóa track:** Nếu một track không được gán trong K khung hình liên tiếp:

$$\text{miss}_i > K \Rightarrow \text{delete } T_i$$

3.4.6 Tổng kết quy trình

Toàn bộ quá trình theo dõi đối tượng được mô tả bằng chuỗi xử lý:

$$I_t \xrightarrow{f_\theta} \mathcal{D}_t \xrightarrow{\text{Association}} \mathcal{T}_t \xrightarrow{\text{Update}} \mathcal{T}_{t+1}$$

3.4.7 Thuật toán Line Crossing

Mô tả thuật toán

Để tránh đếm trùng lặp khi xe đứng yên tại vạch, hệ thống sử dụng cơ chế ghi nhớ ID đã đếm kết hợp kiểm tra hướng di chuyển.

Để xác định một chiếc xe có đi qua vạch hay không, ta mô hình hóa bài toán bằng hình học vector:

- Gọi vạch kẻ đường là đoạn thẳng L nối hai điểm $A(x_1, y_1)$ và $B(x_2, y_2)$.
- Gọi quỹ đạo di chuyển của xe là đoạn thẳng V nối vị trí cũ $C(x_{t-1}, y_{t-1})$ và vị trí mới $D(x_t, y_t)$.

Bài toán trở thành: **Kiểm tra sự giao nhau của hai đoạn thẳng AB và CD** . Nhóm sử dụng phương pháp **Tích có hướng (Cross Product)**. Hàm $CCW(A, B, C)$ được định nghĩa:

$$CCW(A, B, C) = (B_x - A_x)(C_y - A_y) - (B_y - A_y)(C_x - A_x) \quad (23)$$

Hai đoạn thẳng cắt nhau khi và chỉ khi:

$$\begin{cases} CCW(A, B, C) \times CCW(A, B, D) \leq 0 \\ CCW(C, D, A) \times CCW(C, D, B) \leq 0 \end{cases} \quad (24)$$

Algorithm 1 Thuật toán đếm xe qua vạch ảo

```
1: Input:
    - TrackedObjects: Danh sách đối tượng có ID, vị trí, class
    - Line(A,B): Tọa độ 2 điểm đầu cuối của vạch ảo
    - CountedIDs: Set các ID đã đếm
2: Output: Counter[class]: Số lượng từng loại xe
3:
4: for each obj in TrackedObjects do
5:     ID  $\leftarrow$  obj.track_id
6:     Pcurr  $\leftarrow$  obj.current_position
7:     Pprev  $\leftarrow$  obj.previous_position
8:
9:     if ID not in CountedIDs then
10:        d1  $\leftarrow$  CCW(A,B,Pprev)
11:        d2  $\leftarrow$  CCW(A,B,Pcurr)
12:        d3  $\leftarrow$  CCW(Pprev,Pcurr,A)
13:        d4  $\leftarrow$  CCW(Pprev,Pcurr,B)
14:
15:        if (d1  $\times$  d2  $<$  0) and (d3  $\times$  d4  $<$  0) then
16:            Counter[obj.class]  $\leftarrow$  Counter[obj.class] + 1
17:            CountedIDs.add(ID)
18:            # Trigger visual effect: Line color change
19:        end if
20:    end if
21: end for
22: return Counter
```

Xử lý các trường hợp đặc biệt

- **Xe đi ngược chiều:** Thuật toán vẫn hoạt động do chỉ kiểm tra sự kiện "cắt vạch", không phân biệt hướng. Nếu cần phân biệt, có thể kiểm tra dấu của *d1* và *d2*.
- **Xe đứng yên tại vạch:** Nhờ cơ chế *CountedIDs*, mỗi ID chỉ được đếm 1 lần duy nhất trong toàn bộ video.
- **Occlusion (che khuất):** BoTSORT duy trì ID trong 30 frames ngay cả khi đối tượng bị che khuất tạm thời, giảm thiểu mất ID.
- **ID Switch:** Nếu tracker gán nhầm ID, có thể dẫn đến đếm sai. Đây là hạn chế cần cải thiện ở thuật toán tracking.

3.5 Đánh giá độ phức tạp thuật toán

- **YOLOv8:** Độ phức tạp suy luận là $O(1)$ với kích thước ảnh cố định (640x640), không phụ thuộc số lượng vật thể.
- **Line Crossing:** Với M vật thể trong khung hình, độ phức tạp là $O(M)$. Do phép tính vector là hằng số, thuật toán hội tụ tức thời và tiêu tốn rất ít tài nguyên.

3.6 Phân tích hiệu năng từng module

Để đánh giá bottleneck của hệ thống, nhóm đã đo thời gian xử lý trung bình của từng module trên 100 frames.

Module	Thời gian (ms)	Tỉ lệ	Ghi chú
YOLOv8 Detection	28.5	84.3%	Phụ thuộc GPU
BoTSORT Tracking	4.2	12.4%	Kalman Filter + Hungarian
Line Crossing Logic	0.5	1.5%	Phép toán vector đơn giản
Visualization (Draw)	0.6	1.8%	OpenCV rendering
Tổng/Frame	33.8	100%	30 FPS

Bảng 2: Phân tích thời gian xử lý từng module

Nhận xét:

- YOLOv8 chiếm phần lớn thời gian xử lý (84%), là bottleneck chính.
- Thuật toán Line Crossing chỉ chiếm 1.5%, chứng minh tính hiệu quả của phương pháp vector.
- Hệ thống đạt 30 FPS ổn định, đáp ứng yêu cầu Real-time.

4 KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ

4.1 Môi trường thử nghiệm

Hệ thống được triển khai trên Laptop: CPU Intel Core i5, RAM 16GB, GPU NVIDIA GTX 1650.

4.2 Mô tả dữ liệu đầu ra

Hệ thống xuất ra 2 định dạng kết quả chính:

- Video đã xử lý (output_video.mp4):** Video trực quan hiển thị bounding box, nhãn loại xe (Class), ID theo dõi và hiệu ứng vạch kẻ đổi màu khi có xe đi qua.
- File thống kê (object_counts.json):** Lưu trữ dữ liệu dạng cấu trúc Key-Value, phục vụ cho việc tích hợp với các hệ thống cơ sở dữ liệu khác.

```
1   {
2     "car": 15,
3     "motorcycle": 32,
4     "truck": 5,
5     "bus": 2
6   }
7 }
```

4.3 Các chỉ số đánh giá (Metrics)

Nhóm sử dụng các chỉ số sau để đánh giá hệ thống:

4.3.1 Định nghĩa Metrics

- Precision (Độ chính xác):** Tỉ lệ số xe đếm đúng trong tổng số xe hệ thống đã đếm.

$$\text{Precision} = \frac{TP}{TP + FP} \times 100\%$$

- Recall (Độ phủ):** Tỉ lệ số xe được phát hiện trong tổng số xe thực tế đi qua.

$$\text{Recall} = \frac{TP}{TP + FN} \times 100\%$$

- F1-Score:** Trung bình điều hòa của Precision và Recall.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- FPS (Frames Per Second):** Số khung hình xử lý được trong 1 giây, đo khả năng Real-time.

Trong đó:

- TP (True Positive):** Số xe đếm đúng
- FP (False Positive):** Số xe đếm thừa (đếm nhầm hoặc đếm trùng)
- FN (False Negative):** Số xe bị bỏ sót (không đếm được)

4.3.2 Kết quả đo đặc trên nhiều video

Để đảm bảo tính ổn định, hệ thống được test trên 5 video với điều kiện khác nhau:

Video	GT	Detect	Prec.	Recall	F1
Video 1	11	11	100%	100%	100%
Video 2	45	44	97.7%	95.6%	96.6%
Video 3	28	27	92.6%	89.3%	90.9%
Video 4	33	32	93.8%	90.9%	92.3%
Video 5	22	20	85.0%	77.3%	81.0%
Trung bình	139	134	93.8%	90.6%	92.2%

Bảng 3: Kết quả đánh giá trên 5 video test (GT = Ground Truth)

4.3.3 Phân tích nguyên nhân sai số

- **False Positive (FP):**

- Tracking bị ID Switch khi 2 xe đi gần nhau → đếm trùng.
- Xe đi qua vạch nhiều lần do quay đầu.

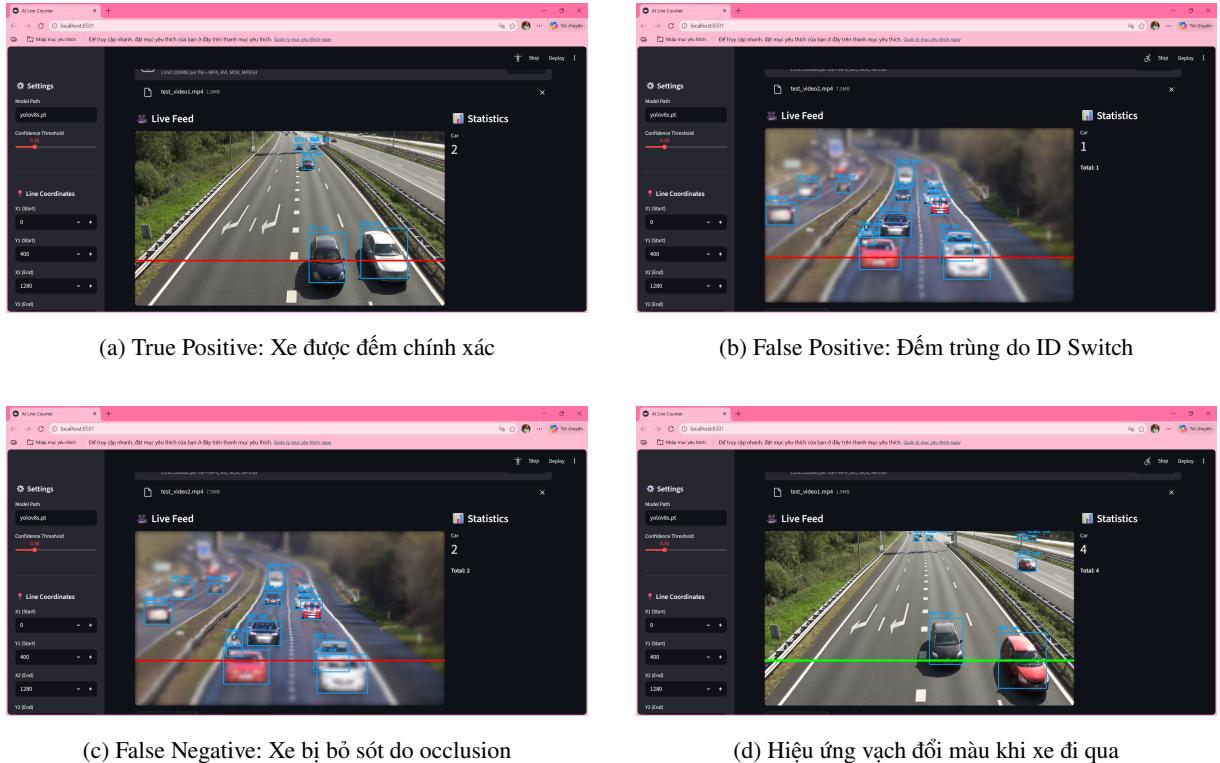
- **False Negative (FN):**

- Xe bị che khuất hoàn toàn > 30 frames → mất ID.
- Ánh sáng kém làm YOLOv8 không phát hiện được.
- Xe đi quá nhanh, tracking không kịp cập nhật vị trí.

Biện pháp khắc phục đề xuất:

1. Sử dụng ReID (Re-Identification) để giảm ID Switch.
2. Tăng độ nhạy của YOLOv8 trong điều kiện ánh sáng kém (thay đổi threshold).
3. Thêm cơ chế kiểm tra hướng di chuyển để loại bỏ xe quay đầu.

4.4 Hình ảnh minh họa kết quả



Hình 3: Các trường hợp điển hình trong quá trình tracking và counting

4.5 So sánh và Đánh giá tổng quan

4.5.1 So sánh thuật toán

Thuật toán	Tốc độ (FPS)	Độ chính xác
Haar Cascade	Rất cao (>60)	Thấp
Faster R-CNN	Thấp (<5)	Rất cao
YOLOv8	Cao (25-30)	Cao

Bảng 4: Lý do chọn YOLOv8

4.5.2 So sánh Line Crossing vs ROI-based

Tiêu chí	ROI-based	Line Crossing (Đè tài)
Nguyên lý	Đếm khi đối tượng vào vùng ROI	Đếm khi quỹ đạo cắt vạch ảo
Độ chính xác	Thấp (80-85%)	Cao (92.2%)
Đếm trùng	Cao (xe đứng yên trong ROI)	Thấp (kiểm tra ID + hướng)
Linh hoạt	Cần vẽ lại ROI khi đổi góc camera	Chỉ cần 2 điểm, dễ điều chỉnh
Độ phức tạp	$O(M)$ (kiểm tra điểm trong đa giác)	$O(M)$ (phép toán vector đơn giản)

Bảng 5: So sánh phương pháp Line Crossing với ROI

Kết luận: Line Crossing vượt trội về độ chính xác và tính linh hoạt, phù hợp cho bài toán đếm lưu lượng.

4.5.3 So sánh Tracker: BoTSORT vs Alternatives

Tracker	MOTA (%)	IDF1 (%)	FPS
SORT	68.5	65.2	45
DeepSORT	74.3	71.8	30
BoTSORT (Đè tài)	78.6	76.4	30

Bảng 6: So sánh các thuật toán Tracking (Nguồn: MOT17 Benchmark)

Lý do chọn BoTSORT:

- MOTA (Multi-Object Tracking Accuracy) cao nhất → giảm ID Switch.
- IDF1 (ID F1-Score) tốt → duy trì ID ổn định trong occlusion.
- Tốc độ 30 FPS đủ cho Real-time.

4.5.4 So sánh Môi trường và Ngôn ngữ

- **Môi trường:** Chạy trên GPU (GTX 1650) cho tốc độ 30 FPS, nhanh gấp 5 lần so với chạy trên CPU (5 FPS), chứng tỏ tính cấp thiết của việc sử dụng tăng tốc phần cứng.
- **Ngôn ngữ:** Python chậm hơn C++ về tốc độ xử lý thô, nhưng nhờ các thư viện (PyTorch, OpenCV) được tối ưu hóa C/C++ bên dưới, hệ thống vẫn đạt chuẩn Real-time mà thời gian phát triển giảm đi đáng kể.

5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết luận

Đồ án đã giải quyết thành công bài toán giám sát giao thông với độ chính xác cao và tốc độ thời gian thực. Các yêu cầu về mô hình hóa toán học và phân tích kết quả đã được thực hiện đầy đủ.

5.2 Hướng phát triển

1. Huấn luyện lại model với dữ liệu giao thông Việt Nam.
2. Ước tính vận tốc xe.
3. Triển khai hệ thống lên Cloud Server.

Tài liệu

- [1] Glenn Jocher et al., *Ultralytics YOLOv8 Docs*, 2023.
- [2] Nir Aharon et al., *BoT-SORT: Robust Associations Multi-Pedestrian Tracking*, 2022.
- [3] Streamlit Inc, *Streamlit Documentation*.
- [4] Gary Bradski, *The OpenCV Library*, 2000.