**1. Tổng quan hệ thống**

**Mục tiêu**:

* Phát hiện nguy cơ va chạm xe bằng cảm biến siêu âm và GPS.
* Gửi dữ liệu (vị trí, khoảng cách) lên Firebase để lưu trữ và phân tích.
* Sử dụng AI để tối ưu hóa tuyến đường, tránh các khu vực có lịch sử tai nạn.
* Hiển thị thông tin GPS, trạng thái tai nạn, và tuyến đường trên giao diện web trực quan.

**Kiến trúc hệ thống**:

* **Phần cứng/IoT**: Raspberry Pi Pico W điều khiển cảm biến siêu âm, module GPS, động cơ, và kết nối WiFi để gửi dữ liệu.
* **Firebase**: Lưu trữ dữ liệu thời gian thực (vị trí, khoảng cách, tai nạn lịch sử).
* **Server Flask**: Xử lý dữ liệu, phát hiện tai nạn, gọi OSRM API, và tích hợp mô hình AI để chọn tuyến đường.
* **Web**: Hiển thị thông tin GPS, bản đồ tuyến đường, bảng tai nạn, và biểu đồ thống kê.
* **AI**: Mô hình Random Forest dự đoán độ an toàn của tuyến đường dựa trên khoảng cách và lịch sử tai nạn.

**Luồng dữ liệu**:

1. Raspberry Pi Pico W thu thập dữ liệu từ cảm biến siêu âm (khoảng cách) và GPS (tọa độ).
2. Dữ liệu được gửi lên Firebase (data và accidents).
3. Server Flask đọc dữ liệu từ Firebase, phát hiện tai nạn, gọi OSRM API để lấy tuyến đường, và sử dụng AI để chọn tuyến an toàn nhất.
4. Web lắng nghe dữ liệu Firebase, gọi API từ server, và hiển thị thông tin trên bản đồ, bảng, và biểu đồ.

**2. Các thành phần chính**

**2.1. Phần cứng và IoT (final.py)**

* **Thiết bị**:
  + **Raspberry Pi Pico W**: Vi điều khiển chính, hỗ trợ WiFi để kết nối mạng.
  + **Cảm biến siêu âm HC-SR04**: Đo khoảng cách đến vật cản (GPIO20: Trigger, GPIO28: Echo).
  + **Module GPS**: Cung cấp tọa độ (vĩ độ, kinh độ) qua UART (9600 baud).
  + **Động cơ**: Điều khiển xe mô hình (IN1, IN2, ENA cho động cơ A; IN3, IN4, ENB cho động cơ B).
  + **Còi (Buzzer)**: Cảnh báo khi khoảng cách nhỏ (GPIO16).
  + **LED nội bộ**: Báo hiệu trạng thái kết nối WiFi.
* **Chức năng** (final.py):
  + **Đo khoảng cách**: Sử dụng HC-SR04 để đo khoảng cách mỗi 100ms. Nếu khoảng cách từ 5-12 cm, coi là nguy cơ va chạm, kích hoạt còi và dừng xe.
  + **Đọc GPS**: Phân tích câu lệnh NMEA $GPRMC để lấy tọa độ (vĩ độ, kinh độ) mỗi 500ms.
  + **Điều khiển xe**: Các hàm car\_forward, car\_backward, car\_left, car\_right, car\_stop để điều khiển động cơ.
  + **Kết nối WiFi**: Kết nối với mạng WiFi (SSID: thantanmadai, Password: 12345678t) và chạy server socket để nhận lệnh điều khiển từ xa.
  + **Gửi dữ liệu**: Đẩy dữ liệu (tọa độ, khoảng cách, liên kết Google Maps) lên Firebase (data) và lưu tai nạn vào bảng accidents nếu phát hiện va chạm.
* **Cải tiến**:
  + Thêm trường severity (nhẹ, trung bình, nặng) vào dữ liệu tai nạn để tăng tính chi tiết.
  + Tối ưu vòng lặp chính với thời gian kiểm tra ngắn (100ms cho cảm biến, 500ms cho GPS, 1000ms cho Firebase).

**2.2. Firebase**

* **Cơ sở dữ liệu**: Firebase Realtime Database lưu trữ dữ liệu thời gian thực:
  + **Bảng data**: Lưu thông tin GPS và cảm biến (latitude, longitude, distance, google\_maps\_link, timestamp).
  + **Bảng accidents**: Lưu lịch sử tai nạn (latitude, longitude, timestamp, severity).
* **Chức năng**:
  + Lưu trữ dữ liệu từ Raspberry Pi Pico W.
  + Cung cấp dữ liệu cho server Flask và web để phân tích và hiển thị.
* **Dữ liệu giả lập**:
  + Script simulate\_accidents.py tạo 100 tai nạn giả lập trong khu vực TP.HCM (tâm: 10.8231, 106.6297, bán kính 10km).
  + 30% tai nạn tại các giao lộ lớn (Nguyễn Huệ, Lê Lợi, Cống Quỳnh).
  + Mức độ nghiêm trọng: Nhẹ (50%), Trung bình (30%), Nặng (20%).

**2.3. Server Flask (server.py)**

* **Chức năng**:
  + **Lấy dữ liệu**: Đọc dữ liệu từ Firebase (data và accidents) qua REST API.
  + **Phát hiện tai nạn**: Nếu distance < 8 cm, coi là tai nạn, lưu vào bảng accidents.
  + **Tính tuyến đường**: Gọi OSRM API để lấy nhiều tuyến đường (alternatives=true).
  + **AI tối ưu hóa**: Sử dụng mô hình Random Forest để dự đoán safety\_score dựa trên:
    - Khoảng cách (distance).
    - Số tai nạn lịch sử gần tuyến (accident\_count).
    - Mức độ nghiêm trọng trung bình (avg\_severity).
  + **API /analyze**: Trả về thông tin tai nạn, tuyến đường, và lý do chọn tuyến.
* **Mô hình AI** (train\_model.py):
  + **Thuật toán**: Random Forest Regressor (50 cây).
  + **Đặc trưng**: distance, accident\_count, avg\_severity.
  + **Nhãn**: safety\_score (thấp hơn là an toàn hơn).
  + **Dữ liệu huấn luyện**: 1000 mẫu giả lập với công thức safety\_score = 0.5\*distance + 10\*accident\_count + 15\*avg\_severity.
  + **Lưu trữ**: Mô hình được lưu vào route\_optimizer\_model.pkl.
* **Cải tiến**:
  + Sử dụng DataFrame (pandas) để đảm bảo dữ liệu đầu vào AI khớp với dữ liệu huấn luyện, tránh cảnh báo scikit-learn.
  + Thêm numpy để tính avg\_severity, khắc phục lỗi name 'np' is not defined.

**2.4. Giao diện web (wet.html)**

* **Giao diện**:
  + Sử dụng **Tailwind CSS** để tạo bố cục đẹp, responsive, với các phần:
    - **Phần trên**: Hiển thị thông tin GPS (vĩ độ, kinh độ, khoảng cách, liên kết Google Maps) và trạng thái tai nạn.
    - **Phần giữa**: Nhập tọa độ xuất phát, hiển thị bản đồ Leaflet, và thông tin tuyến đường (khoảng cách, số tai nạn, mức độ nghiêm trọng, điểm an toàn, lý do).
    - **Phần dưới**: Bảng tai nạn lịch sử, biểu đồ cột (tai nạn theo mức độ), và biểu đồ đường (tai nạn theo ngày).
  + **Bảng tai nạn**: Hiển thị 10 tai nạn mỗi trang, với phân trang (nút Trang trước/sau) và bộ lọc theo mức độ (Tất cả, Nhẹ, Trung bình, Nặng).
  + **Biểu đồ**:
    - **Cột**: Số lượng tai nạn theo mức độ (Nhẹ: xanh lá, Trung bình: cam, Nặng: đỏ).
    - **Đường**: Số lượng tai nạn theo ngày, với đường cong xanh dương.
  + **Marker bản đồ**:
    - Marker tuyến đường: Điểm xuất phát và tai nạn hiện tại (mặc định Leaflet).
    - Marker tai nạn lịch sử: Hình tròn màu:
      * Nhẹ: Xanh lá (#10b981).
      * Trung bình: Cam (#f59e0b).
      * Nặng: Đỏ (#ef4444).
    - Popup hiển thị vĩ độ, kinh độ, mức độ, và thời gian.
* **Chức năng**:
  + Lắng nghe dữ liệu thời gian thực từ Firebase (data và accidents).
  + Gọi API /analyze để lấy thông tin tai nạn và tuyến đường.
  + Cập nhật bảng, biểu đồ, và marker khi dữ liệu thay đổi.
  + Lưu tọa độ xuất phát vào localStorage để giữ giá trị khi tải lại trang.
* **Cải tiến**:
  + Phân trang và bộ lọc để quản lý danh sách tai nạn.
  + Marker màu tùy chỉnh để trực quan hóa mức độ nghiêm trọng.
  + Biểu đồ đường để phân tích xu hướng tai nạn theo thời gian.

**2.5. AI và tối ưu hóa tuyến đường**

* **Mô hình Random Forest**:
  + Dự đoán safety\_score để chọn tuyến đường an toàn nhất từ các tuyến thay thế của OSRM.
  + Đặc trưng: Khoảng cách, số tai nạn lịch sử, mức độ nghiêm trọng trung bình.
  + Đào tạo trên dữ liệu giả lập, nhưng có thể mở rộng với dữ liệu thực tế.
* **Tích hợp**:
  + Server Flask sử dụng mô hình để xếp hạng các tuyến đường và chọn tuyến có safety\_score thấp nhất.
  + Trả về lý do chọn tuyến (ví dụ: "Tuyến đường có 0 tai nạn, mức nghiêm trọng trung bình 1.00").
* **Kết quả**:
  + Tránh các tuyến đường có nhiều tai nạn hoặc mức độ nghiêm trọng cao.
  + Cân bằng giữa khoảng cách và độ an toàn.

**3. Luồng hoạt động**

1. **Phát hiện va chạm**:
   * Raspberry Pi Pico W đo khoảng cách bằng cảm biến siêu âm.
   * Nếu khoảng cách 5-12 cm, kích hoạt còi, dừng xe, và gửi dữ liệu (tọa độ, khoảng cách, mức độ ngẫu nhiên) lên Firebase (data và accidents).
2. **Lưu trữ dữ liệu**:
   * Firebase lưu dữ liệu GPS vào data và tai nạn vào accidents.
3. **Phân tích và tối ưu hóa**:
   * Server Flask đọc dữ liệu từ data, phát hiện tai nạn nếu distance < 8 cm.
   * Gọi OSRM API để lấy các tuyến đường từ điểm xuất phát (do người dùng nhập) đến vị trí tai nạn.
   * Mô hình AI tính safety\_score dựa trên khoảng cách và lịch sử tai nạn, chọn tuyến an toàn nhất.
4. **Hiển thị trên web**:
   * Web lắng nghe Firebase để cập nhật thông tin GPS và tai nạn.
   * Gọi API /analyze để lấy tuyến đường và thông tin tai nạn.
   * Hiển thị trên bản đồ (tuyến đường, marker tai nạn với màu sắc), bảng (tai nạn lịch sử), và biểu đồ (thống kê mức độ và ngày).

**4. Công nghệ sử dụng**

* **Phần cứng**: Raspberry Pi Pico W, HC-SR04, module GPS, động cơ, còi.
* **Ngôn ngữ lập trình**:
  + **MicroPython**: Cho Raspberry Pi Pico W (final.py).
  + **Python**: Cho server Flask (server.py) và huấn luyện AI (train\_model.py).
  + **HTML/JavaScript**: Cho giao diện web (wet.html).
* **Thư viện/framework**:
  + **MicroPython**: machine, utime, network, socket, urequests.
  + **Python**: flask, requests, joblib, numpy, pandas, scikit-learn.
  + **Web**: Tailwind CSS, Leaflet.js, Chart.js.
* **Dịch vụ**:
  + Firebase Realtime Database: Lưu trữ dữ liệu.
  + OSRM API: Tính tuyến đường.

**5. Điểm nổi bật**

* **Tích hợp IoT và AI**: Kết hợp cảm biến siêu âm, GPS, và mô hình Random Forest để phát hiện va chạm và tối ưu hóa tuyến đường.
* **Giao diện web trực quan**:
  + Bản đồ với marker màu (xanh lá, cam, đỏ) cho các mức độ tai nạn.
  + Bảng tai nạn với phân trang và bộ lọc.
  + Biểu đồ cột và đường để phân tích mức độ và xu hướng tai nạn.
* **Dữ liệu giả lập**: Script simulate\_accidents.py tạo dữ liệu tai nạn thực tế trong khu vực TP.HCM, phù hợp khi không thể thu thập thực tế.
* **Responsive**: Giao diện web hoạt động tốt trên cả máy tính và điện thoại.
* **Hiệu quả cho dự án nhỏ**: Giải pháp đơn giản, dễ triển khai, phù hợp với thời gian môn học.

**6. Hạn chế**

* **Dữ liệu giả lập**: Dữ liệu tai nạn không phản ánh hoàn toàn thực tế giao thông tại TP.HCM.
* **Phạm vi AI**: Mô hình Random Forest chỉ xem xét khoảng cách và tai nạn, chưa tính đến các yếu tố như mật độ giao thông hay thời tiết.
* **Hiệu suất**:
  + Bảng và bản đồ có thể chậm nếu Firebase chứa hàng nghìn tai nạn.
  + Marker tai nạn chỉ hiển thị cho trang hiện tại, có thể gây nhầm lẫn.
* **Phần cứng**: Cảm biến siêu âm có độ chính xác hạn chế, không phát hiện được va chạm thực tế (chỉ mô phỏng).
* **Dữ liệu thời gian thực**: Không tích hợp dữ liệu giao thông thời gian thực do hạn chế về API miễn phí.

**7. Đề xuất cải tiến**

* **Phần cứng**:
  + Thêm cảm biến gia tốc để phát hiện va chạm thực tế.
  + Sử dụng camera và học sâu để phân tích tình trạng đường.
* **AI**:
  + Tích hợp dữ liệu thời gian thực (giao thông, thời tiết) từ API như Google Maps.
  + Sử dụng học tăng cường (Reinforcement Learning) để tối ưu hóa tuyến đường động.
* **Web**:
  + Thêm bộ lọc theo thời gian hoặc khu vực tọa độ.
  + Sử dụng Leaflet.markercluster để nhóm marker khi có nhiều tai nạn.
  + Thêm nút xuất bảng tai nạn thành CSV.
* **Firebase**:
  + Tối ưu truy vấn bằng cách giới hạn số lượng tai nạn tải về (ví dụ: 100 tai nạn gần nhất).
  + Sử dụng Firestore để hỗ trợ truy vấn phức tạp hơn.
* **Dữ liệu**:
  + Thu thập dữ liệu thực tế từ các buổi thử nghiệm hoặc nguồn mở (nếu có).
  + Tích hợp dữ liệu giao thông từ các cơ quan địa phương.

**8. Tài liệu và báo cáo dự án**

* **Tài liệu**:
  + Mô tả kiến trúc hệ thống: Phần cứng (Raspberry Pi Pico W, cảm biến), Firebase, server Flask, AI, và web.
  + Giải thích các thành phần: Chức năng của final.py, server.py, wet.html, train\_model.py, simulate\_accidents.py.
  + Nêu rõ cách tích hợp AI để tối ưu hóa tuyến đường và giao diện web để trực quan hóa dữ liệu.
  + Đính kèm ảnh chụp:
    - Giao diện web với bản đồ (marker màu), bảng tai nạn, và biểu đồ.
    - Firebase Console với bảng data và accidents.
    - Log server Flask không có lỗi.
* **Thuyết trình**:
  + Nhấn mạnh sự kết hợp giữa IoT (phát hiện va chạm), AI (tối ưu hóa tuyến đường), và web (trực quan hóa).
  + Trình bày demo: Nhập tọa độ, kích hoạt cảm biến siêu âm, xem tuyến đường và marker tai nạn trên web.
  + Nêu hạn chế (dữ liệu giả lập) và đề xuất cải tiến.
* **Tài liệu tham khảo**:
  + MicroPython: <https://micropython.org/>
  + Firebase: <https://firebase.google.com/docs/database>
  + OSRM: <http://project-osrm.org/docs/v5.5.1/api/>
  + Leaflet.js: <https://leafletjs.com/>
  + Chart.js: <https://www.chartjs.org/>
  + Scikit-learn: <https://scikit-learn.org/stable/>

**9. Kết luận**

Hệ thống cảnh báo va chạm xe này là một giải pháp tích hợp hiệu quả giữa **IoT**, **AI**, và **web**, đáp ứng tốt yêu cầu của một dự án môn học. Với Raspberry Pi Pico W, cảm biến siêu âm, và GPS, hệ thống phát hiện nguy cơ va chạm và lưu trữ dữ liệu lên Firebase. Server Flask và mô hình AI đảm bảo chọn tuyến đường an toàn, trong khi giao diện web cung cấp cái nhìn trực quan với bản đồ, bảng, và biểu đồ. Các tính năng như marker màu, phân trang, và bộ lọc làm tăng tính tương tác và hữu ích của hệ thống.