CSCE 5214: Project

Thai Ha Dang

Phase 4 Report: User Interface Development for AI-Based Vehicle Tracking and Speed Estimation

Project Link: https://github.com/hadt222/CSCE5214.git

Video: https://drive.google.com/drive/folders/1aY3I0KjYnuP3NMojqZyLT9_CLYdZ9fgx

https://www.youtube.com/watch?v=74VHgz9ETHk

Introduction

This Phase 4 report details the development of a user interface (UI) for the "AI-Based Vehicle Tracking and Speed Estimation" system, implementing the extended use cases from Phase 3: Multi-Camera Synchronization and Real-Time Alert Configuration. The report describes communication diagrams for these use cases, provides the UI source code, and outlines the video demonstration.

Communication Diagrams

UML communication diagrams were designed to illustrate how the UI integrates with the system to implement the use case as follows:

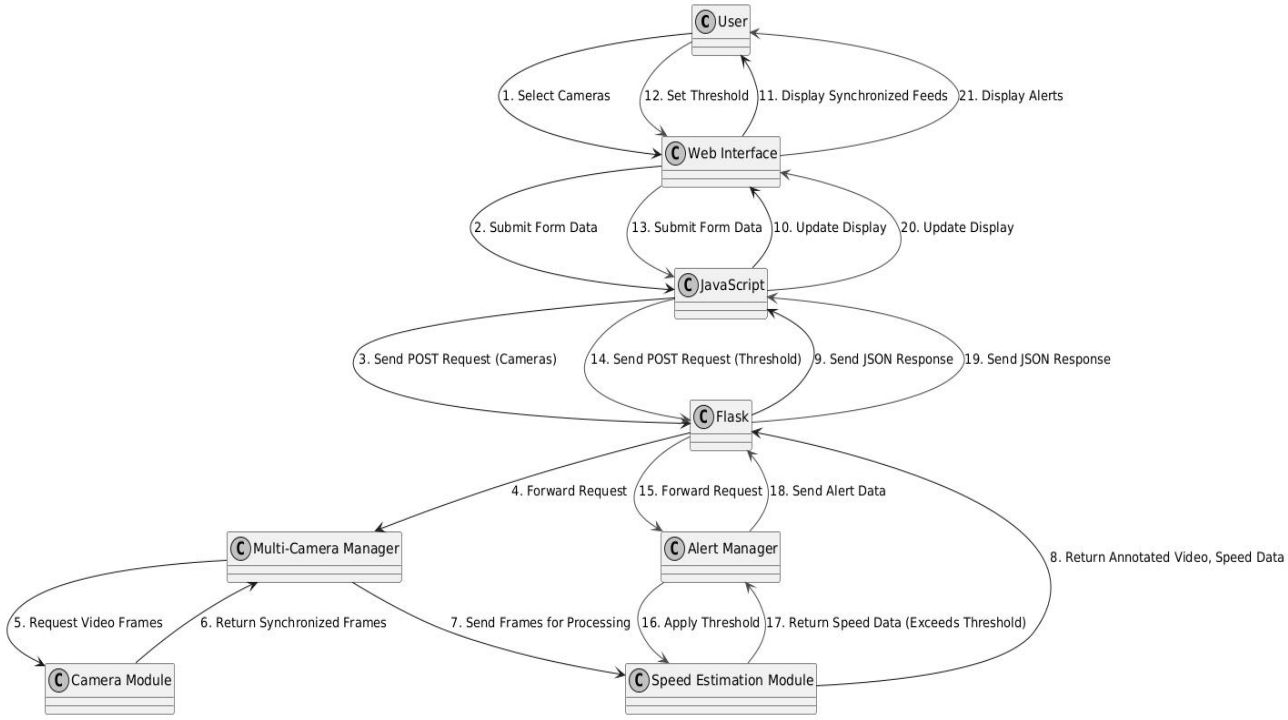

Figure 1: communication diagrams

The system architecture is layered, comprising a Front-End Layer (Web Interface, Form Component, Results Display), a Back-End Layer (Flask API, Multi-Camera Manager, Alert Manager, Camera Module, Speed Estimation Module), and a Data Layer (Training Data, Saved Models). The communication flow begins with user inputs via the Web Interface, processed by JavaScript and forwarded to the Flask API. The Flask API coordinates with the Multi-Camera Manager for synchronization or the Alert Manager for alerts, interacting with the Camera Module and Speed Estimation Module. Processed data (synchronized feeds, alerts) is returned to the Web Interface for display, ensuring a seamless flow between layers.

Sequence:

1. The UI Module requests the Multi-Camera Manager to synchronize feeds.
2. The Multi-Camera Manager instructs the Camera Module to fetch frames.
3. The Camera Module returns synchronized frames.
4. The Multi-Camera Manager forwards frames to the Speed Estimation Module.
5. Speed estimation of detected vehicles is saved in log data file.
6. The Speed Estimation Module processes frames and sends annotated videos to the UI Module.
7. The UI Module displays synchronized feeds to the User.

Purpose: Shows how the UI facilitates user interaction with the Multi-Camera Manager to achieve synchronized monitoring.

UI Implementation

Data Flow and Processing

Data flows from the User through the Web Interface to the Flask API, which requests camera feed from the Multi-Camera Manager or speed data from the Alert Manager. The Camera Module provides video frames, processed by the Speed Estimation Module using mock data (simulating YOLOv11/DeepSORT outputs) to generate annotated feeds and speed values. The Alert Manager compares speeds against the threshold, producing alerts. Processed data is sent back through the Flask API to the Web Interface, updating the Results Display. The Data Layer (Training Data, Saved Models) supports initial model training, though not actively used in the UI.

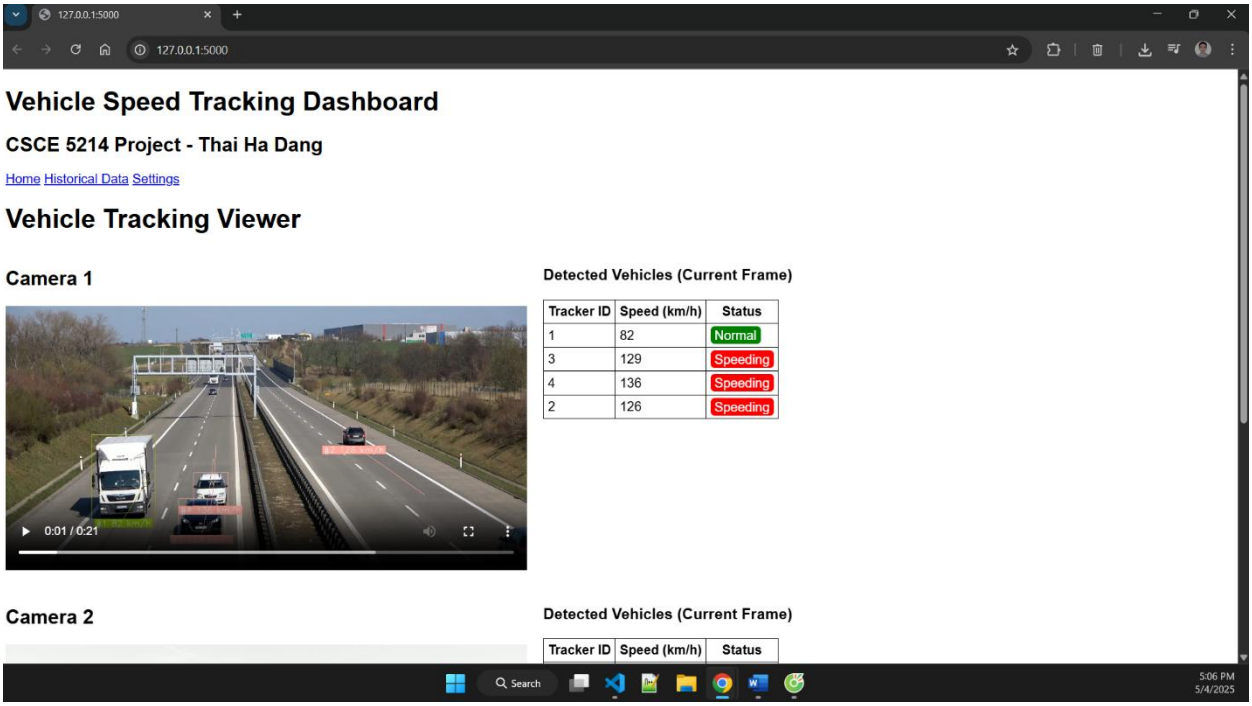

Figure 2: Camara dashboard

The UI is a Flask-based web dashboard, detailed in the source code section below. It supports:

- **Display Multi-Camera Synchronization**: Users select cameras from a dropdown to view synchronized annotated feeds.
- **Real-Time Alert Configuration**: Users set speed thresholds via a settings page, with alerts displayed on the dashboard.

User Input: The operator selects "Camera 1" and "Camera 2"

System Output: The dashboard updates to display two video feed sections with the detected vehicle and these estimation speed.

Interaction: The Web Interface sends the selection to the Flask API via JavaScript. The Flask API processes the request, simulating the Multi-Camera Manager and Speed Estimation Module

Backend process

We use python programming to load videos and trained Yolo_v11 model weight, program will detect the vehicle on the zone as follows, then annotate with its estimated speed.


Figure 1
A
B
C
D
#19.42 km/h
#17.46 km/h
#2.45 km/h
#25.100 km/h
#23.60 km/h
#15.109 km/h
abacus