VIETNAMESE-GERMAN UNIVERSITY

MULTI-DISCIPLINARY PROJECT REPORT

Speed Violation Warning System

Major: Computer Science

Instructor	**: TRUONG TUAN ANH, PhD**
Student 1	**: Pham Dinh Trung Hieu - 104240027**
Student 2	**: Truong Quoc Phong - 104240579**
Student 3	**: Le Minh Khang - 104240055**
Student 4	**: Nguyen Hoang Minh - 104240012**
Student 5	**: Nguyen Luu Ngoc Thanh - 104240585**

---o0o---

BINH DUONG, 04/2025

Table of contents

CHAPTER 1 – INTRODUCTION

I. Context

In today's transportation environment, adhering to speed limits is crucial for ensuring the safety of both drivers and pedestrians. Traditional methods of speed monitoring, however, face limitations when dealing with a growing number of vehicles and the inherent inaccuracies of manual control. Recognizing these challenges, the "Speed Violation Warning System" project was initiated to leverage advanced technologies for automatically detecting speed limit signs and measuring vehicle speed using sensor data. This approach not only enhances road safety by providing timely alerts but also offers a more efficient alternative to conventional monitoring methods.

II. System Description

The project integrates three main technological components:

- Image Processing and Object Detection with YOLO: A fine-tuned version of the YOLO model (e.g., YOLO11n) (Ultralytics, 2024) is used to detect speed limit signs from video footage captured by an on-board camera.
- Sensor Data Processing: The system retrieves accelerometer and magnetometer data from the Arduino IoT Cloud via a REST API using OAuth2 for secure communication. The sensor data, originally in a local coordinate system, is converted into a global coordinate system, and acceleration is integrated over time to accurately compute the vehicle's speed.
- Automated Warning Mechanism: Once the system identifies that a vehicle is exceeding the recognized speed limit, it displays the current speed and the speed limit on the video stream while issuing an audible warning to alert the driver.

By integrating knowledge from IoT, Machine Learning, and Signal Processing, the system is designed with a multi-threaded architecture to concurrently handle sensor data acquisition, speed limit sign detection, and real-time information display. This project not only demonstrates practical applications in traffic safety but also showcases the seamless integration of modern technologies to address real-world challenges.
Github Repository of this project: https://github.com/hieupham1103/Speed-Limit-Detection

CHAPTER 2 – SYSTEM ANALYSIS

I. System Stakeholders

- **Drivers/Users:** End users who operate vehicles and benefit from the speed violation warning system.
- **System Administrators:** Personnel responsible for monitoring system performance, managing updates, and troubleshooting issues.
- **Developers:** Hardware and software engineers who design, implement, and maintain the integration between the sensor data, video processing, and the YOLO detection module.
- **Data Providers:** External entities such as the Arduino IoT Cloud that supply sensor data via REST APIs.

II. System Requirements

1. Functional Requirements

- **Sensor Data Acquisition:**
  The system must continuously fetch sensor data (accelerometer and magnetometer readings) from the Arduino IoT Cloud using a secure REST API (with OAuth2 authentication).
- **Data Processing and Conversion:**
  Convert sensor data from the local coordinate system to the global coordinate system using Euler transformations, then integrate acceleration over time to estimate the vehicle's speed.
- **Speed Limit Sign Detection:**
  Utilize a fine-tuned YOLO model (e.g., YOLO11n) to detect speed limit signs (such as 30 km/h, 70 km/h, 120 km/h) from live video input.
- **Warning Mechanism:**
  Compare the computed speed against the detected speed limit. If the vehicle exceeds the limit, trigger an audible warning and display the current speed and speed limit on the video stream.
- **History Recording:**
  Maintain a record of the vehicle's speed and acceleration data, associated with a unique `CLIENT_ID`, for further analysis and lookup.
-

2. Non-Functional Requirements

- **Performance:**
  The system should operate in real time with minimal latency, ensuring timely detection of speed limit signs and sensor data processing.
- **Reliability:**
  The multi-threaded architecture must ensure continuous, fault-tolerant operation for both sensor data processing and video analysis.
- **Scalability:**
  The design should allow for future expansion, such as integration of additional sensors or extended functionality like more complex warning algorithms.

- **Security:**
  Data transmission, particularly when retrieving sensor data via REST APIs, must be secured using protocols such as OAuth2.
- **Usability:**
  The system should provide clear visual feedback (via video overlays) and audible alerts to ensure the driver is immediately informed about any speed violations.

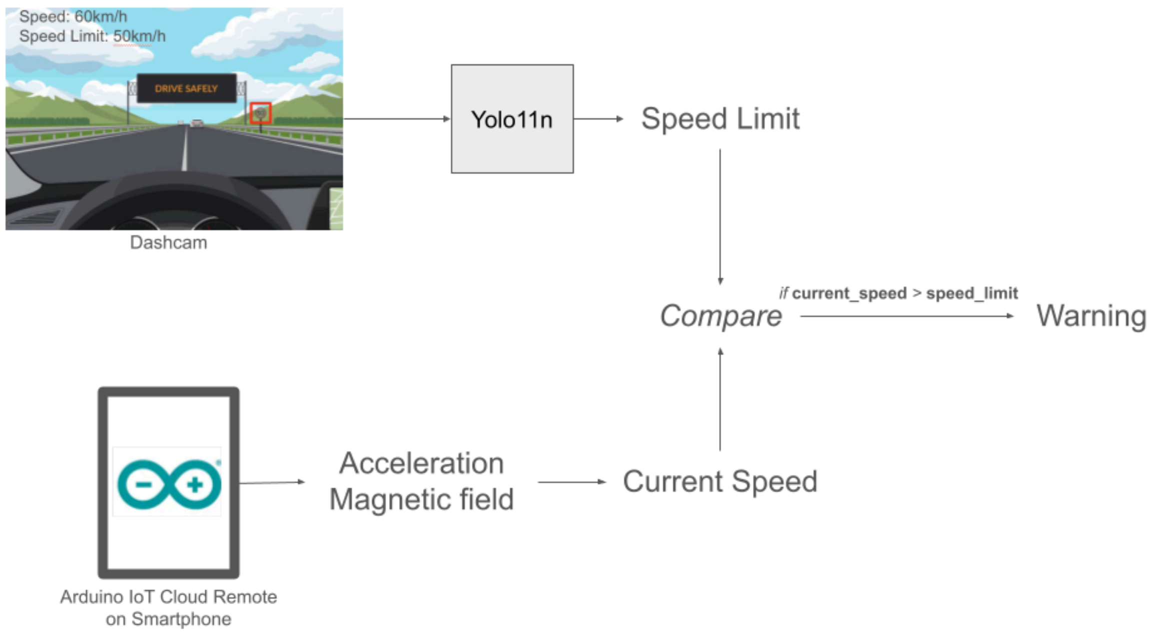


Figure 1: The pipeline outlines the interactions among the stakeholders and the system. Key use cases include: fetching sensor data, processing video input and detecting speed limit signs, calculating vehicle speed, issuing alerts when the speed limit is exceeded.

CHAPTER 3 – SYSTEM DESIGN

I. Architectural Design

The system is built using a multi-threaded architecture to handle simultaneous data acquisition, processing, and display. Its main components include:

- **Sensor Data Acquisition Module:** A dedicated sensor thread continuously retrieves data from the Arduino IoT Cloud via a REST API secured with OAuth2. The raw accelerometer and magnetometer readings are then converted from a local coordinate system to a global one using Euler transformations, and integrated over time to calculate the vehicle's speed.
- **Video Processing and Sign Detection Module:** The main thread is responsible for capturing real-time video using OpenCV. It processes each frame with a fine-tuned YOLO model (e.g., YOLO11n) to detect speed limit signs. The detected speed limit is then used to compare against the calculated speed.
- **Warning and Display Module:** Once the current speed exceeds the detected speed limit, the system triggers an audible warning using a sound library (such as playsound) and overlays both the current speed and the speed limit on the video stream.
- **History Recording Module:** The system logs the speed data along with corresponding timestamps and `CLIENT_ID` information into a lightweight database (e.g., a JSON file) for future reference and analysis.

This modular and multi-threaded design ensures that sensor data and video processing are handled concurrently, reducing latency and ensuring real-time responsiveness in issuing warnings.

II. Database Design

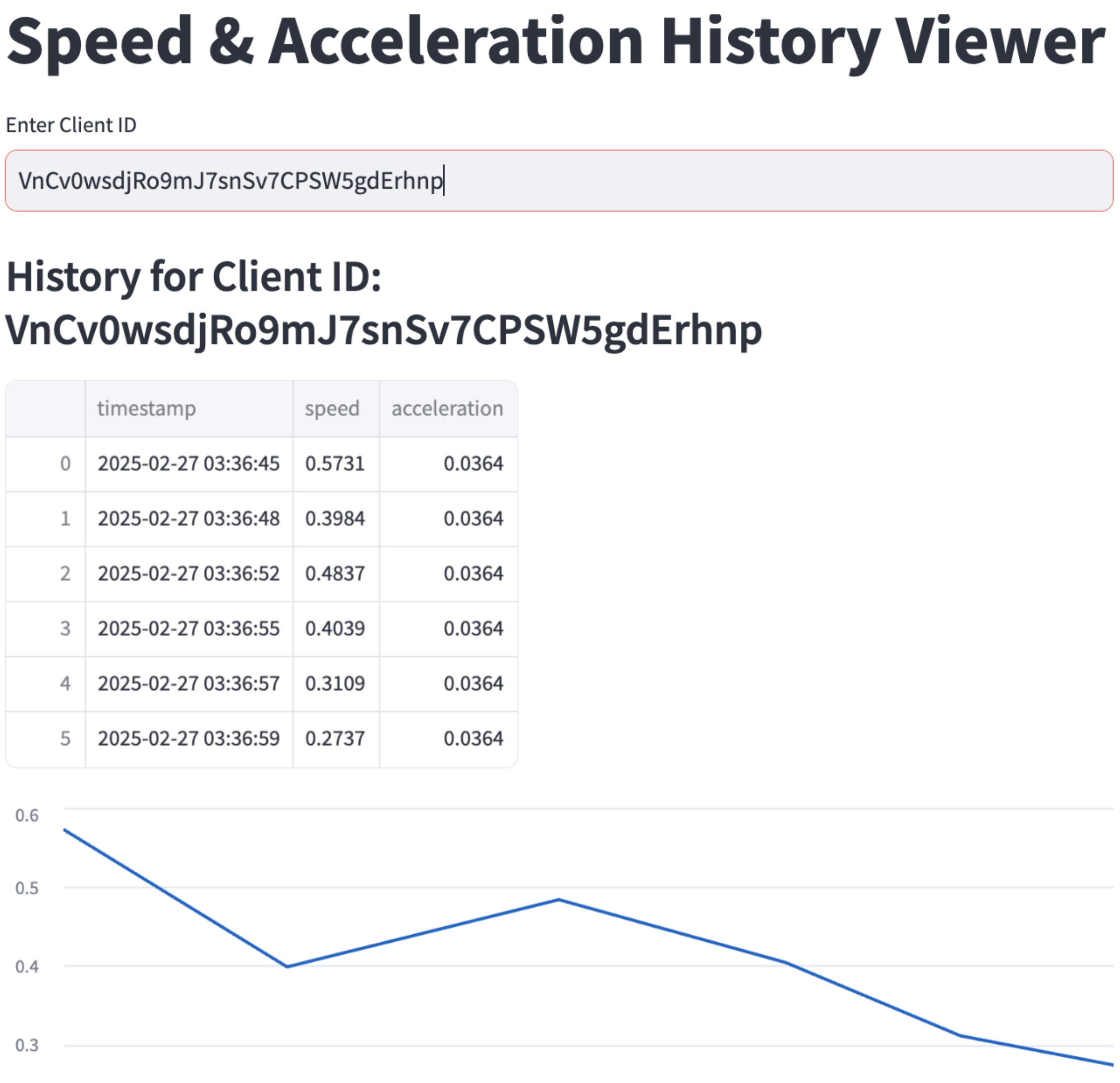
The project uses a simple yet effective database to store historical speed and acceleration records. The key aspects of the database design include:

- **Data Format:** A JSON file (database.json) is utilized to store records. Each record is associated with a unique CLIENT_ID, making it easier to track and query individual vehicle histories.
- **Data Structure:**
  - **Client ID:** Identifier for each user or vehicle.
  - **Timestamp:** The exact time when the record was logged.
  - **Speed and Acceleration Values:** The calculated speed and relevant sensor data that were recorded during each session.
- **Scalability Considerations:** Although a JSON-based solution is lightweight and suitable for prototyping or low-scale applications, the design can be extended to a more robust relational database (e.g., PostgreSQL) if the system needs to scale up.

III. UI Design

The user interface of the system is designed with two main aspects: real-time monitoring and historical data visualization.

- **Real-Time Video Overlay:**
    - **Visual Feedback:** During operation, the live video feed captured by the camera is augmented with overlaid information such as the current speed, the detected speed limit, and an alert message if the vehicle exceeds the limit.
    - **User Interaction:** The interface is simple and intuitive, ensuring that the driver can easily view the critical information without distraction.
- **Historical Data Website:**
    - **Technology Stack:** The project uses Streamlit to create a web-based interface for viewing historical speed data.
    - **Functionality:** Users can query past records, view trends, and analyze their driving performance. The website provides an accessible way to review logged data by `CLIENT_ID` along with graphical representations if needed.


Speed & Acceleration History Viewer

Enter Client ID

VnCv0wsdjRo9mJ7snSv7CPSW5gdErhnp

History for Client ID: VnCv0wsdjRo9mJ7snSv7CPSW5gdErhnp

	timestamp	speed	acceleration
0	2025-02-27 03:36:45	0.5731	0.0364
1	2025-02-27 03:36:48	0.3984	0.0364
2	2025-02-27 03:36:52	0.4837	0.0364
3	2025-02-27 03:36:55	0.4039	0.0364
4	2025-02-27 03:36:57	0.3109	0.0364
5	2025-02-27 03:36:59	0.2737	0.0364

Figure 2: Image of speed & acceleration history viewer.

CHAPTER 4 – SYSTEM IMPLEMENTATION

I. Implementation Overview

The system implementation combines sensor data acquisition, image processing, and alert generation within a multi-threaded Python application. The primary executable is the `main.py` file, which coordinates data retrieval from the Arduino IoT Cloud, real-time video processing with OpenCV, and object detection using a fine-tuned YOLO model. Additionally, a separate Jupyter Notebook (`Train/train.ipynb`) is used for training the YOLO model on a dataset of speed limit signs.

II. Fetching Data from Arduino Cloud

The system leverages the Arduino IoT Cloud to obtain real-time sensor data. Key implementation details include:

- **OAuth2 Authentication:**
  The function `fetch_oauth_token()` handles authentication by sending a request to the Arduino Cloud's token endpoint. This function uses the `oauthlib` and `requests_oauthlib` libraries to securely retrieve an access token, which is then used to authorize subsequent API requests.
- **Data Retrieval:**
  The `get_acceleration()` function retrieves sensor readings for the accelerometer (ACC_X, ACC_Y, ACC_Z) and magnetometer (MAG_X, MAG_Y, MAG_Z) from the Arduino Cloud by making GET requests to the respective endpoints.
  - **Local to Global Conversion:**
    Once the raw sensor data is obtained, the code converts the accelerometer readings from the sensor's local coordinate system to a global coordinate system. This is achieved by (Explain further in CHAPTER 4. VI):
    - Computing Euler angles (roll, pitch, yaw) using the function `compute_euler_angles()`.
    - Generating a rotation matrix with `euler_to_rot_matrix()`.
    - Transforming the local acceleration vector via the function `local_to_global()`.
  - **Speed Calculation:**
    The horizontal acceleration is then computed as the Euclidean norm of the first two components (global X and Y). This value is used in a sensor thread to integrate over time (using simple Euler integration) and estimate the vehicle's current speed in km/h.

III. Integration of YOLO for Speed Limit Detection

- **Model Loading and Detection:**
  The YOLO model (e.g., `30-70-120.pt`) is loaded at the start of the main function using the `ultralytics` package. The function `detect_speed_limit()` processes each video frame to detect speed limit signs. It iterates through the detection results, and for each

bounding box that meets a confidence threshold, it maps the detected class to a predefined speed limit value (e.g., 30, 70, 120 km/h).

- **Alert Mechanism:**
  When the system detects that the vehicle's current speed (calculated from sensor data) exceeds the detected speed limit, an audible alert is triggered by calling the `play_alarm_sound()` function. This alert mechanism ensures that the driver is warned immediately of any speed violations.

IV. Multi-Threaded System Architecture

To ensure real-time performance, the implementation uses a multi-threaded design:

- **Sensor Thread:**
  The `sensor_thread_func()` continuously fetches sensor data from the Arduino Cloud and integrates the acceleration over time to update the current speed. A lock (`sensor_lock`) is used to safely update shared variables across threads.
- **History Saving Thread:**
  The `history_saver_thread_func()` periodically saves the speed history data to a JSON file (`database.json`). This persistent logging enables later analysis and visualization of the vehicle's performance.
- **Main Thread (Video Processing):**
  The main thread handles capturing video frames from the camera using OpenCV, processing each frame through the YOLO model for speed limit sign detection, overlaying the speed and limit information on the video, and managing the warning display.

V. Training the YOLO Model

The system's detection accuracy relies on a YOLO model that has been fine-tuned for recognizing speed limit signs. The training process is performed using the `train.ipynb` Jupyter Notebook, which includes the following steps:

- **Dataset Preparation:**
  A dedicated dataset of speed limit signs (e.g., images labeled with speed limits like 30, 70, 120 km/h) is organized into a suitable directory structure. Data augmentation techniques (such as scaling, rotation, and flipping) may be applied to enhance model robustness.
- **Model Fine-Tuning:**
  The notebook leverages the Ultralytics YOLO framework to fine-tune a lightweight model (e.g., YOLOv5n or YOLO11n). The training process includes:
  - Setting hyperparameters such as learning rate, batch size, and the number of epochs.
  - Monitoring training performance through metrics such as loss and mean Average Precision (mAP).
  - Saving the best-performing model checkpoint as a `.pt` file (e.g., `30-70-120.pt`).
- **Integration into the Main Application:**
  Once training is complete, the generated model file is placed in the project directory. The main application then loads this model to perform real-time detection on incoming video frames.

VI. Sensor Data Conversion

In this system, sensor data is collected from both an accelerometer and a magnetometer. However, the acceleration values measured by the accelerometer are initially expressed in the sensor's local coordinate system. This local measurement does not accurately reflect the true movement direction in the global (real-world) coordinate system. Therefore, converting the data from the local to the global coordinate system is essential for accurately computing the vehicle's speed.

Steps for Data Conversion:

1. **Computing Euler Angles (Roll, Pitch, Yaw):**
   - **Roll and Pitch:**
     Using the accelerometer data, the roll and pitch angles are computed as follows:
     - Roll:

$$\text{Roll} = \arctan\left(\frac{a_y}{a_z}\right)$$

     - Pitch:

$$\text{Pitch} = \arctan\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right)$$

   - These angles describe the sensor's tilt along its axes.
   - **Yaw:**
     Based on the magnetometer data, and after correcting with the computed pitch and roll, the yaw angle is calculated using:

$$\text{Yaw} = \arctan\left(\frac{-m_y'}{m_x'}\right)$$

     where **m_x'** and **m_y'** are the adjusted magnetic field components after compensating for the pitch and roll.
2. **Constructing the Rotation Matrix:**
   Once the Euler angles are obtained, individual rotation matrices for each axis are constructed:

- Rotation matrix about the X-axis (Roll):

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\text{Roll}) & -\sin(\text{Roll}) \\ 0 & \sin(\text{Roll}) & \cos(\text{Roll}) \end{bmatrix}$$

- Rotation matrix about the Y-axis (Pitch):

$$R_y = \begin{bmatrix} \cos(\text{Pitch}) & 0 & \sin(\text{Pitch}) \\ 0 & 1 & 0 \\ -\sin(\text{Pitch}) & 0 & \cos(\text{Pitch}) \end{bmatrix}$$

- Rotation matrix about the Z-axis (Yaw):

$$R_z = \begin{bmatrix} \cos(\text{Yaw}) & -\sin(\text{Yaw}) & 0 \\ \sin(\text{Yaw}) & \cos(\text{Yaw}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. The composite rotation matrix RR is then obtained by multiplying these matrices in orde

$$R = R_z \cdot R_y \cdot R_x$$

4. **Transforming the Acceleration Vector from Local to Global Coordinates:**

   To convert the measured local acceleration vector to the global coordinate system, the following transformation is applied:

$$\vec{a}_{\text{global}} = R^{-1} \cdot \vec{a}_{\text{local}}$$

   Since **R** is an orthogonal matrix, its inverse is equal to its transpose, so this becomes:

$$\vec{a}_{\text{global}} = R^{T} \cdot \vec{a}_{\text{local}}$$

5. **Calculating the Horizontal Acceleration:**
   With the acceleration vector now expressed in the global coordinate system, the horizontal acceleration (considering the xx and yy components) is computed using:

$$a_{\text{horizontal}} = \sqrt{\left(a_x^{(\text{global})}\right)^2 + \left(a_y^{(\text{global})}\right)^2}$$

   This horizontal acceleration value is then integrated over time to estimate the vehicle's speed.

CHAPTER 5 - Testing and Video Demonstration

Since we did not have access to a car for testing, we adapted our approach by placing a mobile phone on the motorbike driver's seat to simulate sensor data collection. The dash cam functionality was provided by a laptop camera, which captured live video during the test. To evaluate the system’s speed limit sign recognition capabilities, we displayed images of speed limit signs to the camera.

Demo video on Youtube: https://www.youtube.com/watch?v=srjwEIAEaeA

During these field tests, we observed that:

- **Real-World Adaptation:**
  Despite the unconventional setup, the sensor data acquisition and real-time video processing worked reliably. The mobile phone successfully simulated the sensor environment, and the laptop camera provided a clear video feed for the YOLO-based detection module.
- **Speed Limit Sign Detection:**
  The system accurately recognized the displayed speed limit sign images, demonstrating that the YOLO model is robust enough to detect and classify speed limit signs even when tested in a controlled, yet dynamic, real-world scenario.
- **Alert Mechanism:**
  The visual overlays on the video feed and the audible alerts were correctly triggered when the simulated speed exceeded the displayed limit, confirming the functionality of the integrated warning system.

A demonstration video was recorded during these tests, showcasing the system's performance with the adapted setup. The video clearly shows the live camera feed with overlaid speed information, the detected speed limit, and the system’s response to simulated speed limit violations. This successful field test validates our approach and confirms that the system is effective even under non-traditional testing conditions.

CHAPTER 6 – SYSTEM DEPLOYMENT

I. Deployment Environment and Hardware Requirements

The Speed Violation Warning System was designed for real-time performance and ease of integration. The deployment environment includes:

- **Hardware:**
    - A laptop serving as the main processing unit with a built-in or external webcam (used as the dash cam).
    - A mobile phone positioned on the motorbike to simulate the sensor data input.
    - Internet connectivity to access the Arduino IoT Cloud for sensor data retrieval.
- **Software:**
    - Python 3.7 or higher.
    - Required Python libraries such as OpenCV, Ultralytics YOLO, playsound, requests, python-dotenv, oauthlib, requests_oauthlib, and numpy.
    - Streamlit (if a web interface for historical data visualization is used).

II. Software Configuration and Setup

Before deployment, the following configuration steps must be completed:

Environment Variables:
Create a `.env` file in the project directory containing all necessary credentials and sensor property IDs, such as:

```
CLIENT_ID=your_client_id
CLIENT_SECRET=your_client_secret
THING_ID=your_thing_id
ACC_LINEAR_ID=your_acc_linear_property_id
ACC_X_ID=your_acc_x_property_id
ACC_Y_ID=your_acc_y_property_id
ACC_Z_ID=your_acc_z_property_id
MAG_X_ID=your_mag_x_property_id
MAG_Y_ID=your_mag_y_property_id
MAG_Z_ID=your_mag_z_property_id
```

Model and Dependencies:
Ensure that the fine-tuned YOLO model file (e.g., `30-70-120.pt`) is in the project directory. Install the required Python packages using:

```
pip install opencv-python ultralytics playsound requests python-dotenv
oauthlib requests_oauthlib numpy streamlit
```

Historical Data Logging:
The system is configured to log speed history in a JSON file (`database.json`), which will be created or updated during system operation.

III. Deployment Procedure

The deployment of the system involves running the main application and, optionally, the web interface for historical data review:

Main Application:
Launch the main application to start sensor data collection, video processing, and speed violation detection:

```
python main.py
```

- The main application fetches sensor data from the Arduino IoT Cloud, processes real-time video frames from the webcam, and overlays the current speed and detected speed limit on the display.

- If a speed violation is detected (i.e., current speed exceeds the displayed limit), the system triggers an audible alert.

Web Interface (Optional):
To review historical speed data, deploy the Streamlit-based web interface by running:

```
streamlit run website.py
```

- This web application allows users to query past records, view performance trends, and analyze logged data based on `CLIENT_ID`.

CHAPTER 7 – CONCLUSIONS

In this project, we developed a Speed Violation Warning System that integrates IoT, machine learning, and signal processing techniques to enhance road safety. The system effectively detects speed limit signs using a fine-tuned YOLO model and computes vehicle speed by processing sensor data obtained from the Arduino IoT Cloud. Through a multi-threaded architecture, the system provides real-time visual overlays and audible warnings to alert drivers when they exceed the speed limit.

Key takeaways from this project include:

- **Integration of Diverse Technologies:**
  The project successfully combines sensor data acquisition, real-time video processing, and object detection, demonstrating the potential of merging IoT with advanced machine learning techniques for practical applications.
- **Real-Time Performance:**
  The multi-threaded design ensures timely and accurate data processing, which is critical for issuing alerts under real-world conditions. Despite challenges during testing—such as adapting our setup using a mobile phone and laptop camera—the system maintained reliable performance.
- **Scalability and Future Enhancements:**
  While a lightweight JSON file is used for historical data logging in the current prototype, the architecture allows for scaling to more robust data storage solutions if necessary. Future work may include integrating additional sensors, refining the detection model with more diverse datasets, and enhancing the user interface for broader applications.

Overall, this project demonstrates a viable approach to automating speed monitoring and warning mechanisms. The promising results obtained from both controlled and field tests provide a strong foundation for further development and eventual real-world deployment.

REFERENCES

Arduino cloud. (n.d.). Build, Control, Monitor Your IoT Projects. Retrieved March 30, 2025, from https://cloud.arduino.cc/

Contributors to Wikimedia projects. (2025a, January 9). *Orthogonal matrix*. Wikipedia. https://en.wikipedia.org/wiki/Orthogonal_matrix

Contributors to Wikimedia projects. (2025b, March 15). *Euler angles*. Wikipedia. https://en.wikipedia.org/wiki/Euler_angles

Contributors to Wikimedia projects. (2025c, March 28). *Rotation matrix*. Wikipedia. https://en.wikipedia.org/wiki/Rotation_matrix

hieupham1103. (n.d.). *GitHub - Hieupham1103/Speed-Limit-Detection*. GitHub. Retrieved March 30, 2025, from https://github.com/hieupham1103/Speed-Limit-Detection

OAuth 2.0 — oauth. (n.d.). Retrieved March 30, 2025, from https://oauth.net/2/

OpenCV - Open computer vision library. (2021, February 9). OpenCV. https://opencv.org/

Pham Dinh Trung Hieu, H. (2025, February 22). - *YouTube*. https://www.youtube.com/watch?v=srjwEIAEaeA

Streamlit • A faster way to build and share data apps. (n.d.). Retrieved March 30, 2025, from https://streamlit.io/

Ultralytics. (2024, September 30). YOLO11 🚀 NEW. *Ultralytics YOLO Docs*. https://docs.ultralytics.com/models/yolo11/

Welcome to python.org. (n.d.). Python.Org. Retrieved March 30, 2025, from https://www.python.org/