

**A COMPUTER VISION-BASED METHOD
FOR VEHICLE SPEED DETECTION
USING VIDEO FOOTAGE**



A MINI PROJECT REPORT

Submitted by

SWEDHA M M	- (6176AC21UCS145)
THENMOZHI S	- (6176AC21UCS150)
THRISHA K	- (6176AC21UCS152)
VARSHINI V	- (6176AC21UCS158)

*in partial fulfilment for the award of the degree
of*

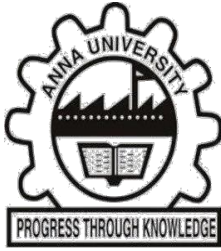
**BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING**

ADHIYAMAAN COLLEGE OF ENGINEERING (AUTONOMOUS)

DR.M.G.R NAGAR, HOSUR-635130

ANNA UNIVERSITY: : CHENNAI 600 025

NOVEMBER 2024



**A COMPUTER VISION-BASED METHOD
FOR VEHICLE SPEED DETECTION
USING VIDEO FOOTAGE**



A MINI PROJECT REPORT

Submitted by

SWEDHA M M	- (6176AC21UCS145)
THENMOZHI S	- (6176AC21UCS150)
THRISHA K	- (6176AC21UCS152)
VARSHINI V	- (6176AC21UCS158)

*in partial fulfilment for the award of the degree
of*

**BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING**

ADHIYAMAAN COLLEGE OF ENGINEERING (AUTONOMOUS)

DR.M.G.R NAGAR, HOSUR-635130

ANNA UNIVERSITY: : CHENNAI 600 025

NOVEMBER 2024

ANNA UNIVERSITY :: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this mini project report “**A COMPUTER VISION BASED METHOD FOR VEHICLE SPEED DETECTION USING VIDEO FOOTAGE**” is the Bonafide work of “**SWEDHA M M (6176AC21UCS145), THENMOZHI S (6176AC21UCS150), THRISHA K (6176AC21UCS152), VARSHINI V (6176AC21UCS158)**” who carried out the project under my supervision.

SIGNATURE

Dr. G. FATHIMA, M.E., Ph.D.,

HEAD OF THE DEPARTMENT

PROFESSOR,

Department of CSE,

Adhiyamaan College of Engineering,

(Autonomous),

Dr. M.G.R. Nagar,

Hosur – 635 130.

SIGNATURE

Mrs. M. MALATHI, M.E.,

SUPERVISOR

ASSISTANT PROFESSOR,

Department of CSE,

Adhiyamaan College of Engineering,

(Autonomous),

Dr. M.G.R. Nagar,

Hosur – 635 130.

Submitted for the Mini project VIVA-VOCE Examination held on_____at
Adhiyamaan College of Engineering (Autonomous), Hosur-635 130.

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

Expressing profound gratitude, I wish to acknowledge the invaluable support and guidance I've received during the execution of my mini project at Adhiyamaan College of Engineering.

We show our sincere thanks to **Dr. R. RADHAKRISHNAN, M.E., Ph.D.,** Principal. Adhiyamaan College of Engineering Hosur, for this valuable opportunity and environment given to us.

We extend our hearty gratitude to **Dr. G. FATHIMA , M.E., Ph.D.,** Professor and Head of the Department. Department of Computer Science and Engineering, Adhiyamaan College of Engineering Hosur, for her guidance and valuable suggestions and encouragement throughout this project.

We are highly indebted to our Supervisor **Mrs. M. MALATHI, M.E.,** Assistant Professor, Department of Computer Science and Engineering, Adhiyamaan College of Engineering, Hosur, whose immense support, encouragement and valuable guidance were responsible for completing the project successfully.

We extend our hearty thankfulness to our project coordinator and all our department faculty for their support in helping us complete this project successfully.

This project would not have been possible without the collective support and guidance of these individuals and the Adhiyamaan College of Engineering community.

ABSTRACT

This project presents a web-based application designed for vehicle speed detection using recorded video footage. The application leverages computer vision techniques, including the Lucas-Kanade optical flow algorithm for motion tracking and the Haar Cascade algorithm for vehicle detection, to accurately estimate vehicle speeds from uploaded videos. Built using the Django framework, the system provides a secure, user-friendly interface that supports multiple user types, including traffic authorities, law enforcement agencies, and individual users.

Users can upload video recordings of vehicles in motion, and the application processes the footage to detect and track vehicles, calculating their speeds based on movement across video frames. The processed video, with annotated speed data, is returned to the user, making it an effective tool for applications such as traffic monitoring, enforcement of speed limits, accident analysis, and insurance assessments.

The back end is powered by Django, handling user authentication, video data management, and communication with the video processing services. The combination of Lucas-Kanade and Haar Cascade algorithms ensures that the system provides accurate speed estimations even in diverse environmental conditions.

This application offers a scalable and reliable solution for vehicle speed detection without requiring specialized hardware, as it functions on recorded footage. By facilitating easy and accessible speed estimation, this tool serves as a valuable asset in traffic management, road safety, and legal enforcement efforts.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE.NO
	ABSTRACT	iv
	LIST OF FIGURES	vii
	LIST OF ABBREVIATIONS	viii
1.	INTRODUCTION	1
	1.1. OVERVIEW	1
	1.2. OBJECTIVES	3
2.	LITERATURE SURVEY	4
3.	SYSTEM ANALYSIS	7
	3.1. EXISTING SYSTEM	7
	3.2. PROPOSED SYSTEM	9
	3.3. PROPOSED SOLUTION	11
	3.4. IDEATION & BRAINSTORMING	13
	3.5. PROBLEM SOLUTION FIT	14
	3.6. ARCHITECTURE DESIGN	17
	3.7. DESCRIPTION OF MODULES	20
	3.7.1. USER INTERFACE MODULE	20
	3.7.2. AUTHENTICATION MODULE	20
	3.7.3. VIDEO UPLOAD & PROCESSING MODULE	20
	3.7.4. ERROR HANDLING & VIDEO FORMAT CONVERSION MODULE	21
	3.7.5. OPTICAL FLOW & SPEED CALCULATION MODULE	21
	3.7.6. RESULT VISUALIZATION MODULE	21
	3.8. DATAFLOW DIAGRAM	22
4.	SYSTEM REQUIREMENTS	23
	4.1. HARDWARE REQUIREMENTS	23
	4.2. SOFTWARE REQUIREMENTS	23

5.	IMPLEMENTATION	24
5.1.	USER INTERFACE MODULE	24
5.2.	USER AUTHENTICATION	25
5.3.	UPLOAD AND PROCESSING VIDEO	26
5.4.	RESULT VISUALIZATION MODULE	27
5.5.	SOFTWARE DESCRIPTION	28
5.6.	CODE IMPLEMENTATION	32
5.8.	RESULT	37
6.	CONCLUSION AND FUTURE ENHANCEMENT	41
6.1.	CONCLUSION	41
6.2.	FUTURE SCOPE	43
	APPENDICES	45
	SOURCE CODE	45
	REFERENCES	60

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
3.1	BRAINSTORM, IDEA LISTING AND GROUPING	13
3.2	MODEL ARCHITECTURE	17
3.3	DATAFLOW DIAGRAM	22
5.1	HOME PAGE	24
5.2	ABOUT PAGE	24
5.3	FEATURES PAGE	24
5.4	REGISTER PAGE	25
5.5	LOGIN PAGE	25
5.6	VIDEO UPLOAD PAGE	26
5.7	RESULT PAGE 1	27
5.8	RESULT PAGE 2	27

LIST OF ABBREVIATIONS

HTML	-	Hyper Text Markup Language
CSS	-	Cascading Style Sheet
JS	-	JavaScript
VS Code	-	Visual Studio Code
CV	-	Computer Vision

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The Vehicle Speed Detection Web Application is an innovative solution designed to estimate vehicle speeds accurately from recorded video footage, offering a valuable tool for traffic authorities, police departments, and individual users. The application provides a user-friendly platform for video-based speed detection without requiring specialized hardware or complex setup, making it accessible and practical for various stakeholders involved in road safety, law enforcement, and personal verification.

This project aims to develop a reliable system that can analyze vehicle speed directly from video files, helping improve road safety and assisting in investigations or insurance verification. The web app enables users to upload recorded videos and receive speed information for vehicles detected in the footage. This process is especially valuable for traffic monitoring, where accurate speed data can help enforce speed limits, analyze traffic patterns, and support accident investigations.

The application's architecture includes a front-end interface, built for ease of use, where users can upload videos and view the processed output. The back-end, powered by the Django framework, handles the core functionality of the application, including video processing and user authentication. Video processing relies on the Lucas-Kanade optical flow algorithm, which tracks vehicle movement across frames, and the Haar Cascade classifier, which detects vehicles within each frame. By using these algorithms together, the application can estimate vehicle speeds based on frame-by-frame movement analysis, providing a processed video that includes annotated speed data.

Once authenticated, users can securely upload footage, initiate the processing, and view or download the annotated video with estimated speeds. The processed video allows users to verify speeds for specific vehicles, making it useful for traffic authorities in monitoring and controlling speed limits. Law enforcement agencies can use this data to analyze accident footage, which can provide critical evidence in investigations.

Additionally, individual users and insurance companies may benefit from this application to verify vehicle speeds for insurance claims or legal cases, adding a layer of credibility and support for dispute resolution.

The application leverages Python's OpenCV library for image and video analysis, while Django provides a robust and scalable framework for web functionalities. By combining advanced computer vision algorithms with a straightforward web interface, the Vehicle Speed Detection Web Application serves as an effective and versatile tool, with potential applications in multiple domains that require accurate and accessible vehicle speed analysis. This project underscores the potential of integrating computer vision with web technologies to solve real-world challenges in an accessible and scalable manner.

1.2 OBJECTIVE

The main Objective of the project is

- **Accurate Vehicle Speed Estimation:** To provide precise vehicle speed calculations from recorded video footage using computer vision algorithms.
- **User-Friendly Interface:** To create an accessible web application where users can easily upload videos and view processed results with speed data annotations.
- **Automated Video Processing:** To utilize the Lucas-Kanade optical flow and Haar Cascade algorithms for efficient vehicle detection and speed estimation without manual intervention.
- **Support for Multiple Users:** To cater to various stakeholders, including traffic authorities, law enforcement, and individual users, providing tailored insights based on their needs
- **Enhanced Road Safety and Evidence Support:** To assist in monitoring speed compliance, aiding investigations, and supporting insurance claims by providing reliable speed data.
- **Scalable and Secure Web Application:** To ensure a scalable back-end using Django and secure user authentication for handling sensitive data.

CHAPTER 2

LITERATURE SURVEY

The literature survey provided references to multiple studies by Patel et al. [1], Zhang, Y. [2], Singh, A. [3], and Ruiz, M. [4], all of which are centered around the themes of vehicle speed estimation using optical flow techniques. These studies collectively delve into the complexities of optical flow algorithms in the context of traffic monitoring and vehicle speed detection.

Patel et al. [1] present a theory on improving vehicle speed estimation accuracy by using advanced optical flow techniques. They emphasize the importance of focusing on selected key points for motion estimation, underscoring that the success of real-time speed detection methods depends on computational efficiency, which makes Lucas-Kanade more suitable for traffic scenarios. This study highlights the computational advantages of selecting key points over the entire flow, making the system highly efficient for real-time processing.

Zhang, Y., Wang, X., & Liu, J. (2022). [2] take a broader view, focusing on real-time vehicle speed estimation using convolutional neural networks (CNNs) combined with optical flow. They explore the system's ability to avoid retraining dependencies by using pre-trained Haar Cascade classifiers, ensuring robustness across various traffic conditions. This study highlights how combining CNNs with optical flow methods can enhance vehicle detection and speed tracking without significant model adjustments, making the solution adaptable to diverse environments.

Singh, A., & Gupta, R. (2023). [3] shift the focus to vehicle detection and speed estimation using optical flow in traffic surveillance systems. By integrating Haar Cascade detection with Lucas-Kanade optical flow, the study explores handling occlusions more effectively. This is particularly relevant in urban settings where vehicles may be partially blocked by obstacles. The study underscores the importance of combining detection and tracking techniques to address real-world challenges, such as partial occlusions, for accurate speed detection.

Fernandez, J., & Ruiz, M. (2022). [4] examine the comparative evaluation of various optical flow methods for vehicle tracking in traffic scenarios. They discuss the classification of vehicles and how filtering out non-vehicle objects impacts the accuracy of speed estimation systems. The study provides insights into the importance of focusing on relevant traffic objects to ensure that the speed detection process remains accurate and relevant, particularly in dynamic and complex urban environments.

Kumar, R., & Sharma, N. (2023). [5] Vehicle Speed Detection Using Haar Cascade and Optical Flow: A Machine Learning Approach. This paper explores the integration of the Haar Cascade algorithm with Optical Flow techniques to detect and estimate vehicle speed. The authors present a machine learning approach that enhances the accuracy of vehicle detection in varied environmental conditions. The study demonstrates the adaptability of this combination in handling occlusions and diverse traffic scenarios, contributing to reliable speed detection in real-world applications.

Brown, P., & Lewis, K. (2023). [6] Computer Vision Applications in Traffic Monitoring and Speed Analysis: A Review. This review paper provides an extensive analysis of computer vision applications in traffic monitoring, with a specific focus on speed estimation methods. The authors compare various techniques, including optical flow, object tracking, and deep learning-based approaches, identifying their strengths and limitations. The study highlights recent advancements in computer vision for real-time traffic monitoring and outlines future directions in automated traffic systems.

Cheng, H., & Zhao, T. (2022). [7] Improving Accuracy in Vehicle Speed Detection Through Feature Optimization in Optical Flow Algorithms. This work addresses the challenges of accuracy in vehicle speed detection by optimizing feature selection in optical flow algorithms. The study introduces novel modifications to conventional optical flow computation, reducing noise and enhancing the reliability of speed estimation in complex traffic environments. The proposed method achieves significant improvements in precision compared to traditional approaches.

Rahman, M., & Lee, S. (2023). [8] Vehicle Detection and Speed Estimation in Real-World Scenarios Using Hybrid Machine Learning Models. The authors propose a hybrid machine learning model combining object detection algorithms with motion estimation techniques to estimate vehicle speed. This approach demonstrates robustness in diverse environmental and lighting conditions, enabling practical deployment in real-world scenarios. The study also emphasizes scalability and adaptability for large-scale traffic monitoring systems.

Wang, L., & Chen, Z. (2022). [9] Implementation of Speed Estimation Techniques Using Deep Learning and Optical Flow Analysis. This paper investigates the use of deep learning models in conjunction with optical flow analysis for vehicle speed estimation. The authors integrate convolutional neural networks (CNNs) with motion-based algorithms, achieving superior accuracy in speed detection. The work highlights the potential of deep learning in capturing complex motion patterns and improving traffic analysis systems.

Davis, R., & Patel, M. (2023). [10] A Scalable Framework for Automated Traffic Surveillance Using OpenCV and Machine Learning. This study introduces a scalable framework for automated traffic surveillance, utilizing OpenCV and machine learning. The framework includes vehicle detection, tracking, and speed estimation modules. The authors focus on efficiency and scalability, demonstrating the system's ability to process large volumes of video data while maintaining accuracy and low computational overhead.

CHAPTER 3

SYSTEM ANALYSIS

3.1. EXISTING SYSTEM

Existing speed measurement systems include radar-based, LIDAR-based, inductive loop sensors, and video-based systems. Radar-based systems utilize radio waves to detect vehicle speed from a distance, offering portability but requiring expensive installation and regular calibration. LIDAR systems use laser light to accurately measure speed, effective in various conditions, but also involve high costs and sensitivity to environmental factors. Inductive loop sensors, embedded in the road surface, detect vehicles through changes in inductance caused by metal, providing reliable traffic monitoring with minimal maintenance needs but requiring significant road modifications for installation.

Video-based systems capture footage analyzed through image processing algorithms, providing additional data like vehicle type and license plate recognition. While non-intrusive and adaptable, they depend on lighting and weather for accuracy and involve high initial setup costs. Overall, these systems each have distinct advantages and challenges, with common issues such as high costs, maintenance demands, and logistical complexities, prompting the exploration of newer technologies for more efficient speed monitoring and traffic management.

These systems have several limitations, including high installation and maintenance costs, complex installation procedures, environmental sensitivity, limited flexibility, and challenges in data processing. Traditional methods often require expensive hardware and significant modifications to road infrastructure, which can be disruptive and costly. To address these issues, the proposed optical flow-based speed estimation system utilizes existing CCTV cameras already in place for surveillance.

This system eliminates the need for additional hardware, simplifies deployment, and relies primarily on software configuration for speed estimation, resulting in minimal physical changes. It also allows for easy scalability by leveraging current camera infrastructure and reducing ongoing maintenance needs. Overall, this innovative approach enhances speed estimation while overcoming the limitations of traditional systems, making it an effective solution for traffic monitoring.

DISADVANTAGES

- **Hardware Dependency:** Existing systems like radar guns and LIDAR require specialized and expensive hardware, increasing costs and limiting scalability.
- **Limited Accessibility:** Many systems are designed for real-time use and cannot process pre-recorded footage, reducing flexibility in applications like accident analysis.
- **Inaccuracy in Challenging Conditions:** Environmental factors such as poor lighting, weather conditions, or occlusions can impact the accuracy of hardware-based or simplistic vision systems.
- **High Maintenance:** Hardware-based systems require regular calibration and maintenance to ensure reliability, increasing operational costs.
- **Lack of Automation:** Traditional methods often require manual intervention, which can lead to human error and inefficiency.

3.2. PROPOSED SYSTEM

The proposed system for vehicle speed detection using video footage aims to provide an effective solution for traffic monitoring and speed enforcement. This system leverages computer vision techniques, specifically the Haar Cascade algorithm for vehicle detection and the Lucas-Kanade optical flow method for tracking vehicle movement, using standard video cameras to ensure broad applicability.

The system begins with capturing video feeds from cameras positioned along roadways, which can be fixed at specific intersections or highways. These video feeds are processed to enhance image quality through resizing and video conversion. This preprocessing step is crucial for accurate detection and tracking. The Haar Cascade algorithm is then applied to detect vehicles in each frame of the video. It uses pre-trained classifiers to recognize features such as edges and shapes that are typical of vehicles. This allows the system to identify vehicles in various traffic conditions, creating bounding boxes around each detected vehicle.

Once the vehicles are identified, the Lucas-Kanade optical flow method tracks their movement across successive video frames. This method estimates the motion of objects by focusing on key feature points that can be reliably tracked, such as corners or textured regions on the vehicles. By analyzing how these feature points shift between frames, the system can calculate the movement path of each vehicle with precision. This is particularly useful for monitoring multiple vehicles simultaneously, ensuring that each detected vehicle is tracked individually throughout the video sequence.

The core of the system involves estimating the speed of each tracked vehicle by measuring its displacement over time. The displacement, calculated through the tracked motion of feature points, is converted into real-world distances using a calibrated pixel-to-meter conversion based on the camera's positioning. The frame rate of the video provides the time component needed to determine speed, allowing the system to compute accurate speed values for each vehicle.

By combining the strengths of the Haar Cascade algorithm and the Lucas-Kanade optical flow method, this system offers a robust solution for vehicle speed detection. It enhances traffic management capabilities by providing accurate, speed measurements, contributing to improved road safety and more efficient traffic flow. This approach ultimately aims to transform traditional speed detection methods, offering a modern solution that is both practical and adaptable to the needs of urban traffic management.

ADVANTAGES

- **Hardware Independence:** The proposed system uses readily available video footage, eliminating the need for costly and specialized hardware.
- **Flexibility with Pre-recorded Footage:** It can analyze both live and recorded videos, making it versatile for applications like post-accident investigations and insurance claim verification.
- **Robust Accuracy:** Advanced computer vision techniques, such as Haar Cascade and optical flow algorithms, improve accuracy even in challenging conditions like variable lighting or partial occlusions.
- **Low Maintenance:** The software-based solution reduces the need for physical maintenance and calibration, lowering operational costs.
- **Automation:** Automated detection and speed estimation reduce human intervention, minimizing errors and enhancing efficiency.

3.3. PROPOSED SOLUTION

The proposed solution for the vehicle speed detection project is a comprehensive web application that enables users to upload pre-recorded video footage to analyze and determine vehicle speeds accurately.

The application combines advanced computer vision techniques, namely the Haar Cascade for vehicle detection and the Lucas-Kanade optical flow algorithm for motion tracking, to provide reliable speed estimation results. By processing the footage through these algorithms, the system can identify moving vehicles, track their movement across frames, and calculate their speeds with high accuracy.

The back end of the application is built using the Django framework, which handles essential functions such as user authentication, database management, and request handling. This structure ensures a secure and efficient environment for managing user interactions and storing necessary data, including processed video results. A database stores user data and processed video metadata, enabling easy access to previous analyses and user preferences. Additionally, FFmpeg is integrated for video format conversion, making the application compatible with various video inputs and ensuring that footage is processed in a format optimized for analysis.

The front end of the application provides an intuitive and user-friendly interface where users can easily upload videos, view analysis results, and access other functionalities. The system is designed to accommodate different types of users, such as traffic authorities, police departments, and individual users.

Upon uploading a video, the application processes it by applying the Haar Cascade algorithm to detect vehicles and then uses the Lucas-Kanade optical flow algorithm to track these vehicles across frames. The speed is calculated based on the tracked movement between frames, with the processed footage then annotated to display speed data visually.

The output, a processed video showing calculated speeds, can be used as evidence in traffic enforcement or accident investigations, as well as in personal cases to assess speed data for insurance or personal review.

In addition to providing reliable speed detection, the application streamlines the process of speed enforcement and monitoring, reducing the need for manual speed checks or radar systems. This proposed solution addresses several real-world challenges in traffic safety, including the enforcement of speed limits, the analysis of accident footage, and the accessibility of accurate speed data for individuals and organizations alike.

3.4 IDEATION & BRAINSTORMING

Ideation is often closely related to the practice of brainstorming, a specific technique that is utilized to generate new ideas. A principal difference between ideation and brainstorming is that ideation is commonly more thought of as being an individual pursuit, while brainstorming is almost always a group activity.

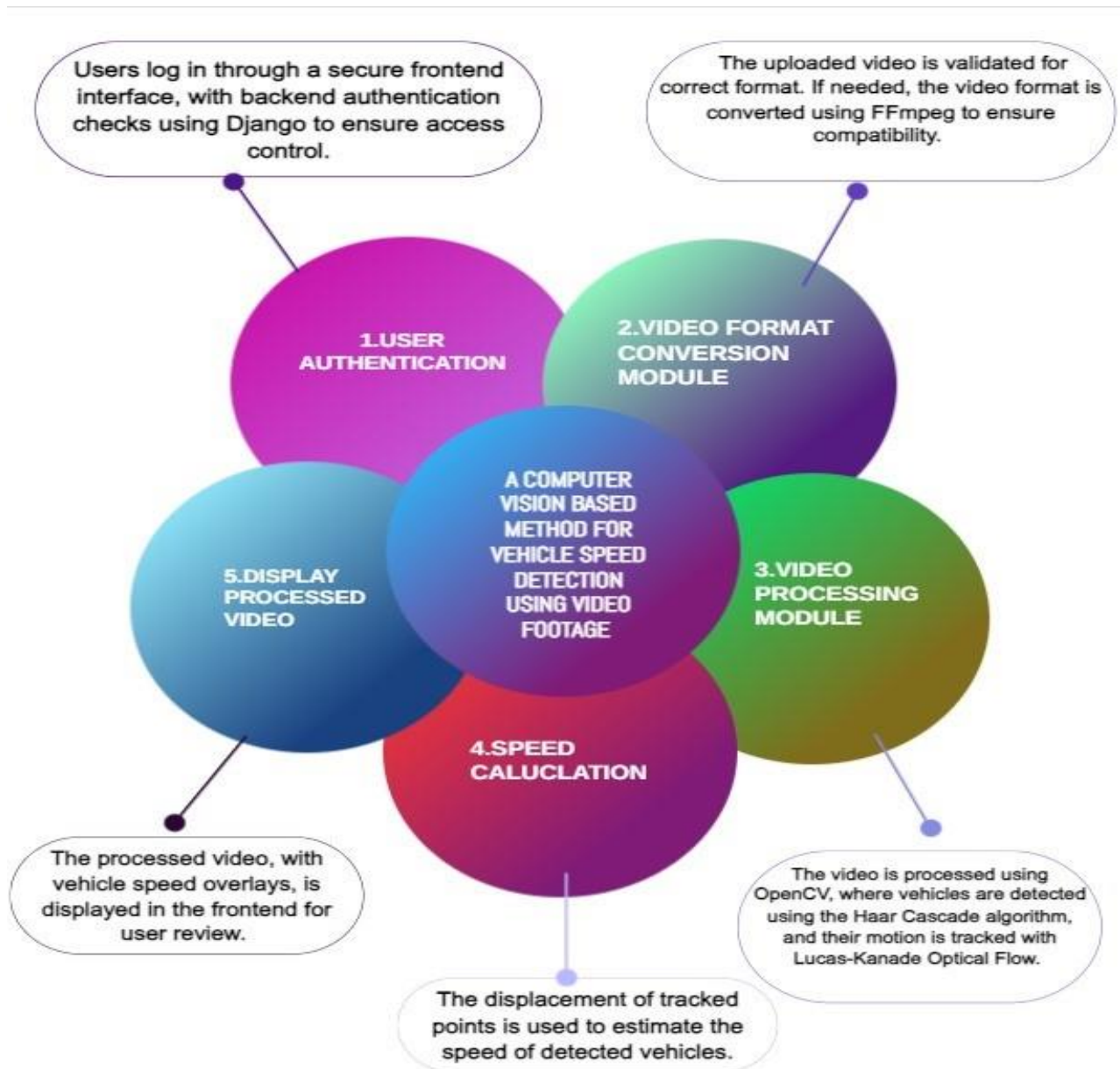


Figure 3.1: Brainstorm, Idea Listing and Grouping

3.5. PROBLEM SOLUTION FIT

This project addresses the need for an accessible, efficient, and cost-effective solution for detecting vehicle speeds from pre-recorded video footage. Conventional speed detection methods like radar systems are expensive and require specialized equipment, making it challenging to implement them in all locations, especially in regions with limited resources.

Additionally, manually analyzing recorded footage is time-consuming, prone to human error, and not feasible for large-scale traffic monitoring. This project proposes an automated approach using computer vision and machine learning techniques, integrating these into a user-friendly web application. This solution is designed to be easily accessible to both technical and non-technical users, providing a robust alternative to traditional speed-detection methods:

1. Automated Vehicle Detection with Haar Cascade

To accurately detect vehicles in video frames, the project uses the Haar Cascade algorithm, a machine learning-based approach trained to recognize specific vehicle features. Haar Cascade classifiers are well-known for their effectiveness in object detection, especially for shape-based recognition like cars. This algorithm scans the video frame for predefined vehicle features and provides location data for detected vehicles. The efficiency of Haar Cascade allows the system to process multiple frames quickly, enabling near-real-time detection while balancing computational resources. This step is essential as it accurately identifies vehicle locations within frames, laying the groundwork for precise tracking and speed measurement.

2. Vehicle Motion Tracking Using Lucas-Kanade Optical Flow

After detecting a vehicle, the system uses the Lucas-Kanade optical flow algorithm to track its movement across successive frames. Optical flow algorithms are ideal for tracking moving objects in video due to their ability to detect changes in pixel intensity. Lucas-Kanade, in particular, is a feature-tracking algorithm that focuses on tracking small pixel clusters across frames. This algorithm is robust, allowing the

system to track vehicles accurately even with changes in lighting or background, common in real-world traffic footage. The use of optical flow ensures consistent and precise tracking, which is crucial for calculating the distance traveled by each vehicle over time, directly impacting the accuracy of the speed estimation.

3. Speed Calculation Using Pixel Distance and Frame Rate

To estimate the speed of detected vehicles, the project calculates the distance a vehicle travels in pixel units across frames, then converts this to a real-world measurement. The scale for converting pixel distance to real distance (pixels-per-meter) is based on assumptions specific to the video. The frame rate (frames per second) of the video is also factored in to measure how quickly each vehicle moves across frames. By combining these measurements, the system calculates speed in km/h with reasonable accuracy. This calculation is essential as it transforms tracking data into meaningful, real-world information that can be used in law enforcement and insurance assessments.

4. Video Processing with FFmpeg for Format Consistency

The project incorporates FFmpeg to ensure all uploaded videos are compatible with the processing pipeline by converting them to the H.264 format if needed. FFmpeg is a widely-used, powerful tool for video format conversion, allowing the application to handle different video formats users may upload. By standardizing the format, FFmpeg minimizes compatibility issues and ensures that every video uploaded to the application can be processed seamlessly. This step is crucial for usability as it allows users to upload footage directly from various devices without additional pre-processing steps.

5. Django Framework for Web Application and User Interface

The entire project is built on the Django web framework, which provides a reliable and scalable backend for managing user interactions and video processing requests. Django's structure supports multiple simultaneous users, enabling the system to be used by various stakeholders without performance degradation. The web application interface guides users through uploading their video, initiating the

processing, and viewing the final results. Django's powerful ORM, session management, and authentication mechanisms further enhance the usability, allowing for a secure, organized, and efficient application experience.

6. Result Presentation with Annotated Video Output

After processing, the final output is an annotated video displaying the speed of each detected vehicle directly on the footage. This visual annotation provides a clear and immediate representation of speed data, valuable for users who require documented proof, such as law enforcement or insurance investigators. The annotated video is stored on the server and accessible through the application interface, making it easy for users to review and download the processed results.

3.6. ARCHITECTURE DESIGN

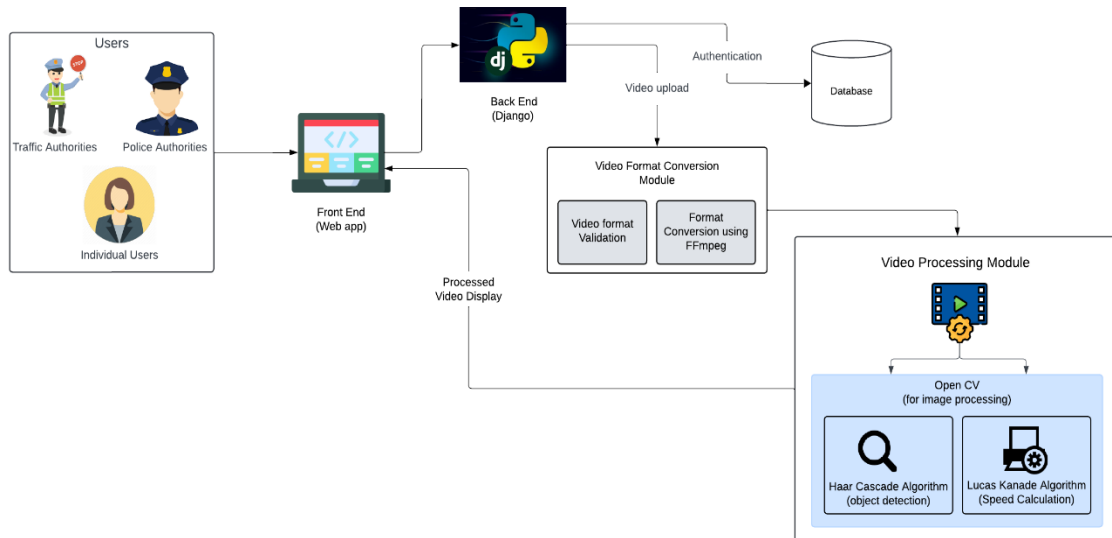


Figure 3.2. Model Architecture

The figure shown above represents the Solution Architecture that we made use of in our Project.

1. Presentation Layer

This layer provides the user interface, enabling interaction between the users and the system. It is responsible for gathering input from users (such as video uploads) and displaying outputs (processed videos). It ensures an intuitive flow for the user, from signing in to viewing the results.

- **Landing Page:** The homepage where users get introduced to the application.
- **Login/Signup Page:** Allows users to authenticate and register for personalized access.
- **Video Upload Page:** Users upload recorded videos of vehicle footage for speed analysis.
- **Result Page:** Displays the processed video with vehicle speed annotations to users.

2. Application Layer (Backend)

- **Views:** Handles requests and defines the responses for each URL, linking the presentation layer with backend processing.
- **URL Routing:** Directs users to the correct view based on their requests, organizing the paths in the application.
- **Static & Media Management:** Manages the storage and retrieval of media files, such as uploaded videos and static assets like CSS or images.
- **Authentication & Authorization:** Ensures that only authorized users can access certain features, securing the application.
- **Data Handling:** Manages data flow and ensures that all data inputs and outputs are correctly processed and delivered to the respective layers.

3. Service Layer (Video Processing with OpenCV)

This layer is responsible for the core functionality of vehicle detection and speed calculation. It performs the actual computational tasks. After receiving the video, it processes each frame to detect and track vehicles, calculate their speeds, and annotate the video with the results.

- **Haar Cascade Algorithm:** Detects vehicles in each frame of the uploaded video. Haar cascades are machine learning-based classifiers that identify objects in images.
- **Lucas-Kanade Algorithm:** Uses optical flow to track detected vehicles across frames, enabling calculation of speed based on movement.
- **Processed Video:** Generates an annotated output video where the speed of each detected vehicle is displayed.

4. Data Access Layer (Database and File Storage)

This layer manages data storage, ensuring that uploaded videos, user credentials, and other data are securely stored and accessible. It provides data persistence, allowing the application to store user information and videos. It ensures that videos and user details are accessible as needed and securely stored.

- **SQLite Database (User Credentials):** Stores user information, such as login details and history of processed videos.
- **Media Folder (Video Storage):** Saves uploaded videos and processed outputs so they can be retrieved and displayed.

3.7. DESCRIPTION OF MODULES

3.7.1. USER INTERFACE MODULE

This module provides a visually appealing and easy-to-navigate front-end interface for users to interact with the application. Built with HTML, CSS, and JavaScript, it includes essential pages such as the landing page, login/signup page, video upload page, and result display page. This module focuses on user experience, ensuring that the interface is intuitive and user-friendly, with clear instructions and messages guiding users through the application. Each page is designed to provide a seamless user journey, from uploading a video to viewing

3.7.2. AUTHENTICATION MODULE

This module serves as the security backbone of the application, handling user registration, login, and logout functionality. It uses Django's built-in authentication system, which includes secure password hashing and session management, to protect user accounts and control access to the platform. This module ensures that only authorized users can access the video upload and processing features, thereby maintaining privacy and security. Users can create accounts or log in to access their dashboards, and this module manages these interactions seamlessly while securely maintaining user session data.

3.7.3. VIDEO UPLOAD & PROCESSING MODULE

This module allows users to upload recorded video footage for analysis. This module performs file validation to confirm that the video meets quality and format standards suitable for processing. Once a video is uploaded, it is stored in a dedicated media folder within the application's file structure. This module is integrated with Django's file handling system to facilitate efficient storage and retrieval of videos, setting up the videos for further analysis by the speed detection algorithms.

3.7.4. ERROR HANDLING & VIDEO FORMAT CONVERSION MODULE

This module addresses any issues that may arise during the video upload or processing stages. It handles various error scenarios, such as unsupported video formats, resolution problems, and processing errors, by converting videos to a compatible format and resolution if needed. This module uses FFmpeg for format conversion, ensuring that the system can accept and process a wide range of video inputs. Additionally, it provides informative error messages to guide users in troubleshooting issues, making the application more robust and user-friendly.

3.7.5. OPTICAL FLOW & SPEED CALCULATION MODULE

This module is responsible for analyzing the uploaded videos to detect vehicles and calculate their speed. It employs computer vision techniques using OpenCV, specifically leveraging the Haar Cascade algorithm for initial vehicle detection. Once detected, the module applies the Lucas-Kanade optical flow algorithm to track vehicle movement across frames, calculating the speed based on positional changes over time.

3.7.6. RESULT VISUALIZATION MODULE

This module displays the processed video with the detected vehicle's speed in a clear and informative manner. Once the speed is calculated, this module overlays the speed data onto the video, allowing users to view the analysis results in real-time as the video plays. The module enhances the user experience by providing visual feedback on the vehicle's speed, making it easy for users to interpret the data and understand the results of the analysis.

3.8. DATAFLOW DIAGRAM

- Start the process by collecting video footage of traffic or moving vehicles.
- Preprocess the video to enhance its quality, adjust the frame rate, and convert it to a compatible format.
- Apply the Haar Cascade classifier to detect vehicles in each video frame.
- Extract the regions containing the detected vehicles to focus on specific areas of interest. Implement the Lucas-Kanade Optical Flow algorithm to track vehicle movements across frames by analyzing feature points.
- Calculate the vehicle speed by using the displacement of these tracked points and the video's frame rate.
- Display the results by overlaying the speed data on the processed video and presenting it to the user.
- Finally, conclude the process and provide the results for further analysis or download.

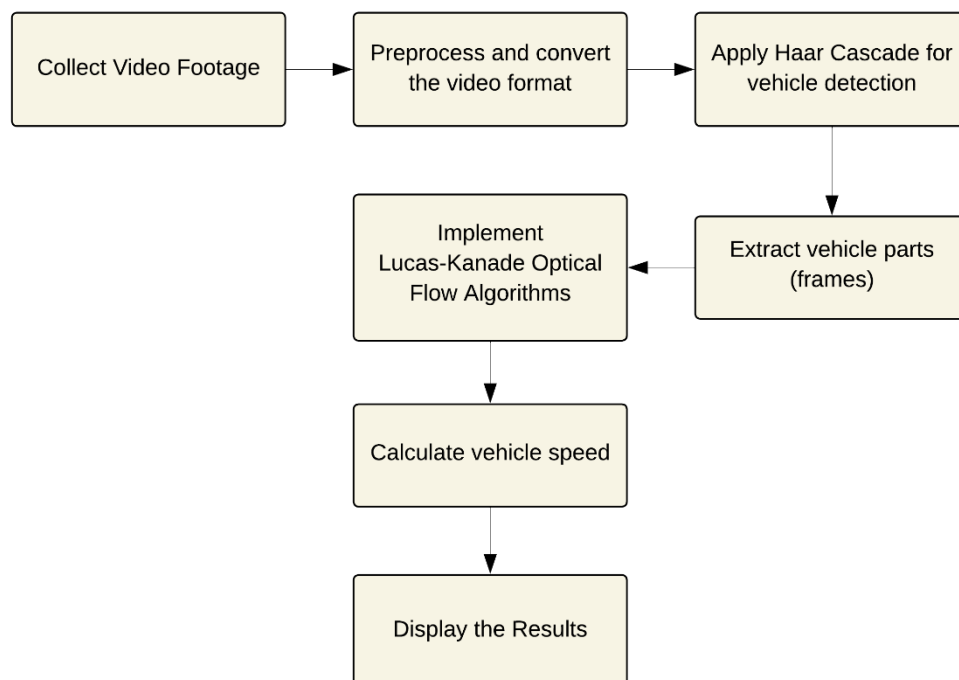


Figure 3.3: Data Flow Diagram

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENT

- CPU
- GPU
- RAM
- Storage
- Camera
- Monitor
- Internet Connection

4.2 SOFTWARE REQUIREMENTS

- Python Idle 3.9.0
- Web Framework – Django Framework
- Machine Learning Libraries
- Database - Sqlite
- Development Tools – Vs Code, Anaconda Navigator
- Ide Extensions
- Browser
- Version Control Tools
- Testing Tools

CHAPTER 5

IMPLEMENTATION

5.1 USER INTERFACE MODULE

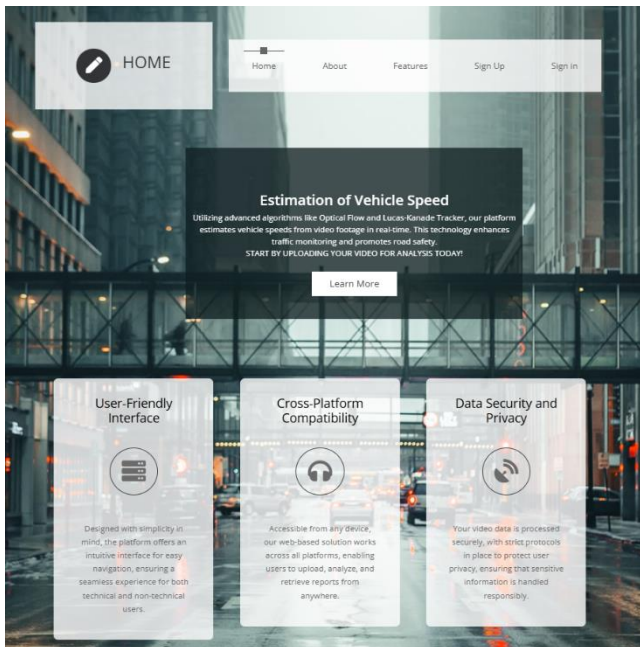


Figure:5.1. Home page

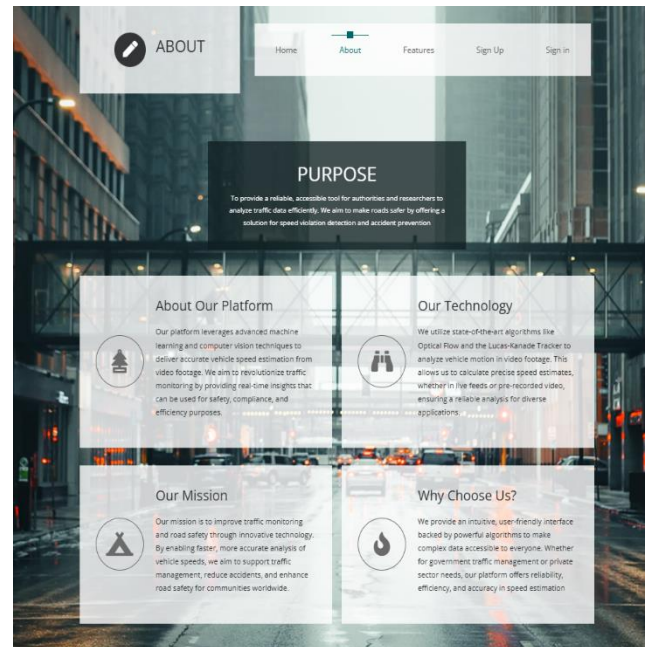


Figure:5.2. About Page

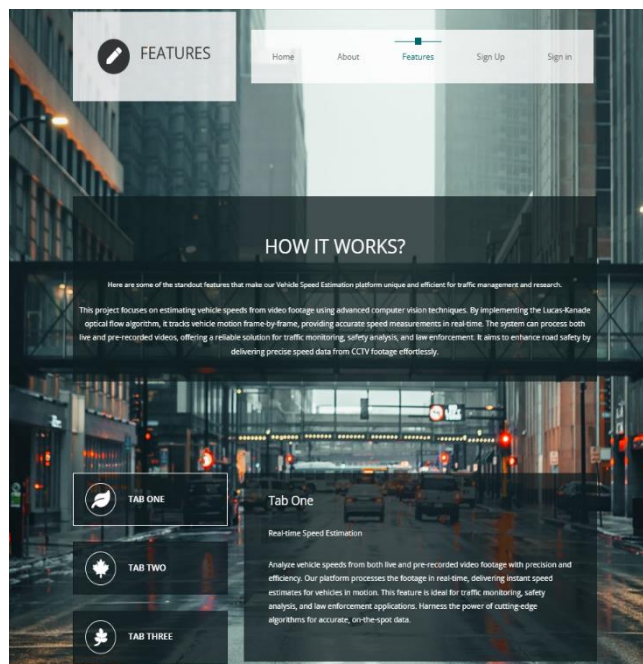


Figure:5.3. Features Page

5.2. USER AUTHENTICATION

The Authentication Module in the vehicle speed detection system ensures secure access and user management. It allows users to register, log in, and access the platform using their credentials. The module is responsible for verifying user identities and managing user sessions using Django's built-in authentication features. It ensures that only authorized users can upload video footage for processing, preventing unauthorized access to the system's features.

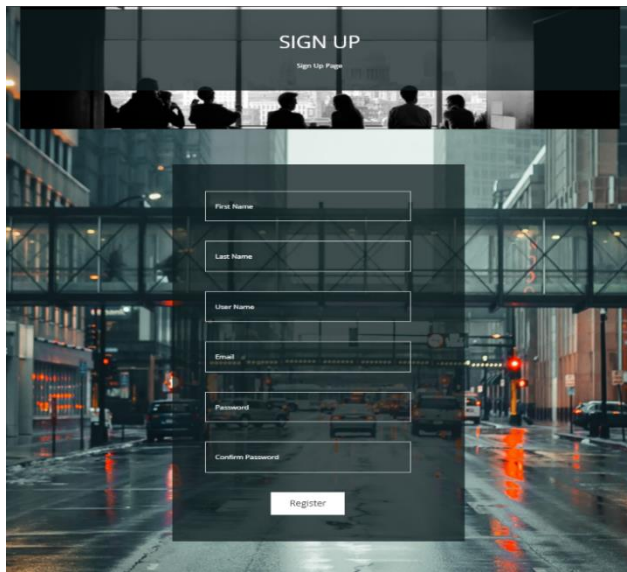


Figure:5.4. Register Page

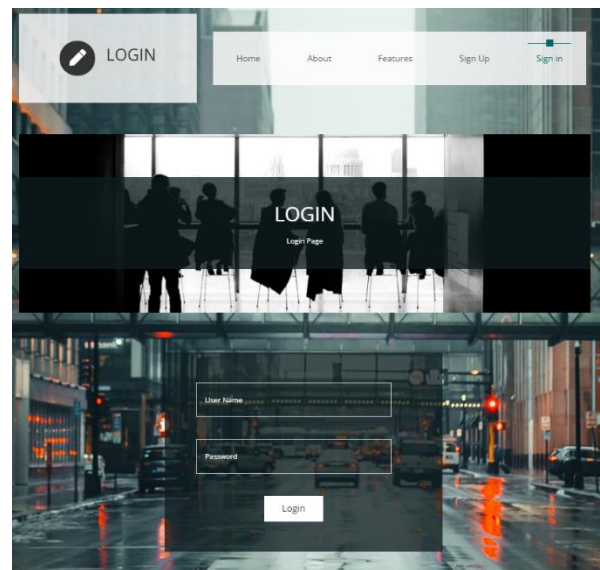


Figure:5.5. Login Page

5.3 UPLOAD AND PROCESSING VIDEO

The Uploading and Processing Video Module allows users to upload video footage through the web interface for speed detection analysis. Once a video is uploaded, the module verifies the video format and size to ensure compatibility with the processing algorithms. After validation, the video is sent to the processing pipeline, where the Haar Cascade algorithm detects vehicles in each frame, and the Lucas-Kanade algorithm tracks their movement to estimate speeds. The module manages the entire processing flow, providing users with real-time status updates on the progress of the video analysis. Upon completion, the processed video, with detected vehicles and calculated speeds, is ready for user review and download.

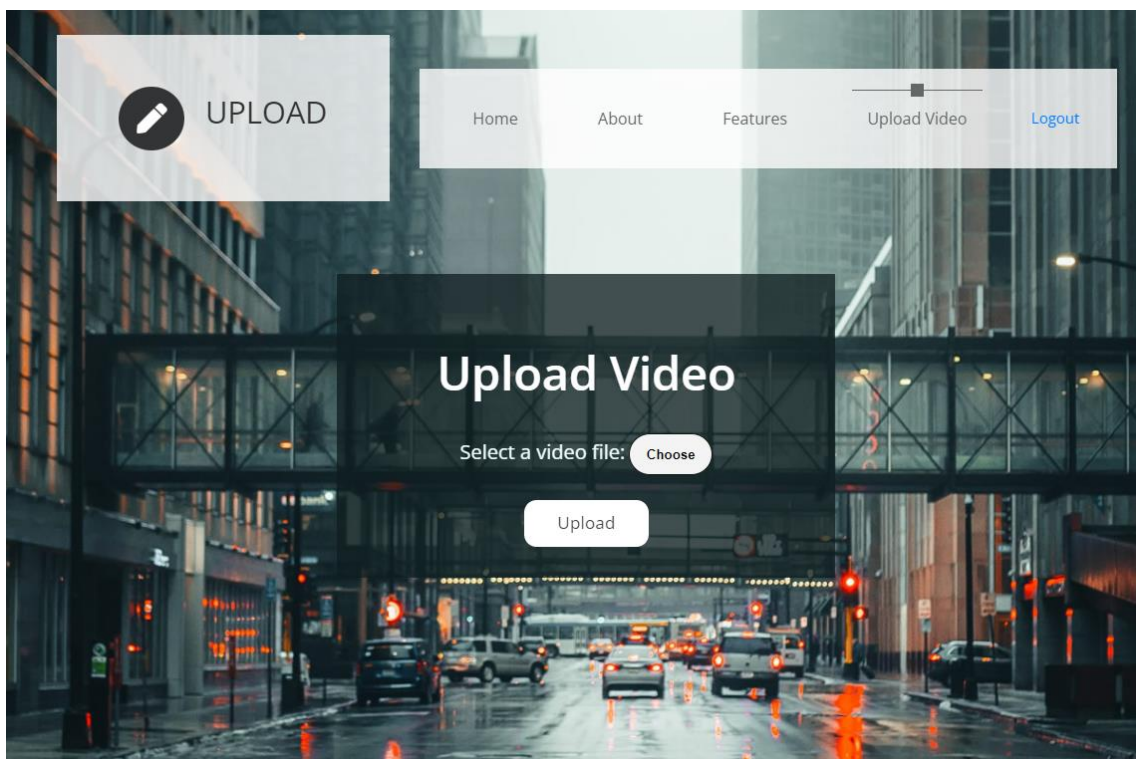


Figure:5.6.. Video Upload Page

5.4. RESULT VISUALIZATION MODULE

The Result Visualization Module is responsible for presenting the output of the video analysis to the user in an intuitive manner. After processing the uploaded video with the Lucas-Kanade and Haar Cascade algorithms, this module overlays the calculated speeds directly onto the video footage, showing each detected vehicle with its corresponding speed. Users can view the processed video through the web interface, where speed data is displayed as annotations on the moving vehicles. The module also provides options for users to download the processed video or review a summary of the speed data in a tabular format. This helps users understand and interpret the results clearly, making it suitable for further analysis or evidence.

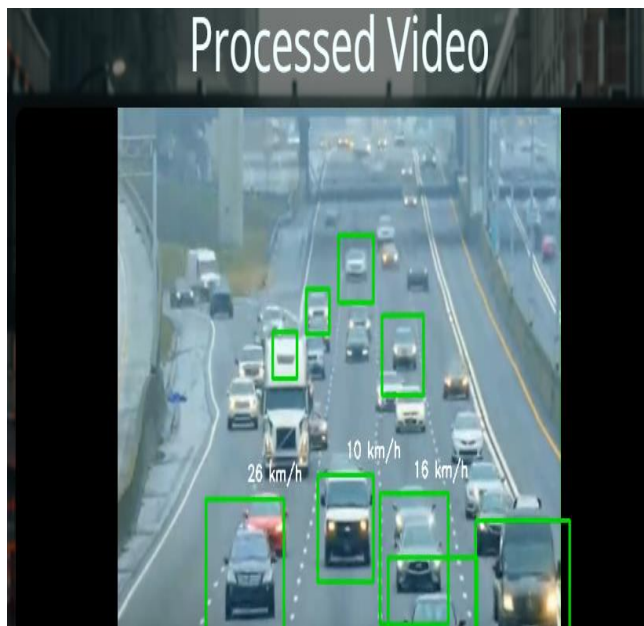


Figure:5.7. Result Page 1

(Vehicle speed will be detected on top of each vehicle)

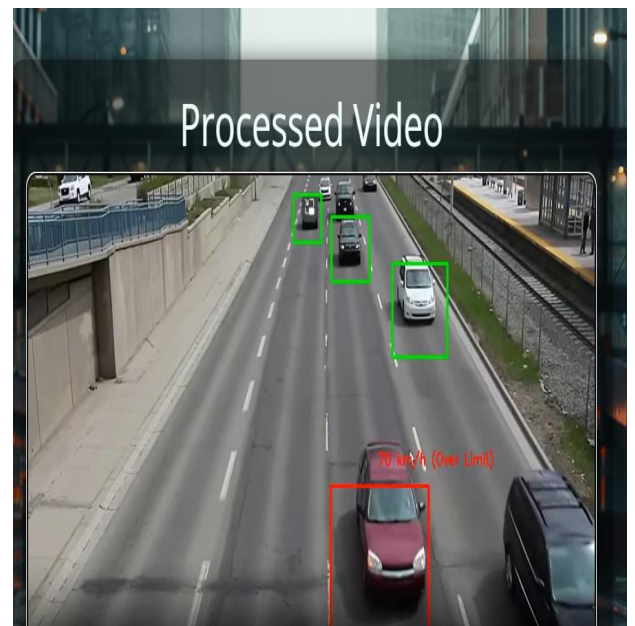


Figure:5.8. Result Page 2

(Over speeding vehicles will be highlighted in red color)

5.5. SOFTWARE DESCRIPTION

1. Visual Studio Code (VS Code)

A Visual Studio Code (VS Code) is a lightweight yet powerful source code editor developed by Microsoft, highly favored by developers for its versatility and support for various programming languages. It is designed for building and debugging modern web and cloud applications, making it an excellent choice for projects involving web development, such as the crop and fertilizer recommendation system. VS Code offers cross-platform support, enabling users to work on Windows, macOS, or Linux. One of its most notable features is the integrated terminal, which allows developers to run commands and scripts directly from the editor.

The extensive extension marketplace enhances functionality, enabling the installation of tools for Python, Django, and other languages, which provides features like code linting, syntax highlighting, and debugging. Version control integration with Git allows users to manage their source code repositories seamlessly, facilitating collaboration. Additionally, IntelliSense offers intelligent code completion, providing context-aware suggestions that accelerate coding and reduce errors.

In the context of your project, VS Code served as the primary Integrated Development Environment (IDE), enabling efficient coding and debugging of Python scripts, Django views, and templates. The editor's version control integration allowed multiple developers to work on the codebase concurrently. By leveraging extensions specific to Django, developers enhanced productivity through rapid development, testing, and deployment. The integrated terminal simplified management of the Django server, database migrations, and running unit tests, resulting in a streamlined development process.

Overall, VS Code was a robust and flexible code editor that significantly contributed to the development of the crop and fertilizer recommendation project, improving code quality and developer productivity.

2. Django Framework

Django is a high-level web framework for Python that facilitates rapid development of secure and maintainable web applications. Following the Model-View-Template (MVT) architectural pattern, Django promotes a clean separation of business logic, presentation, and data management. Its batteries-included philosophy provides a rich set of tools and libraries essential for building complex web applications.

Django's features include a robust admin interface for managing application data, user authentication systems for secure registration and login processes, and an Object-Relational Mapping (ORM) system that allows developers to interact with the database using Python objects. The framework includes built-in protection against common web vulnerabilities, ensuring the security of applications.

In your project, Django served as the backbone of the web application, providing the necessary framework for both frontend and backend development. The MVT architecture enabled organized code development, allowing for effective implementation of features such as user authentication, data processing, and integration of machine learning models. The built-in admin interface facilitated management of user data and inputs for crop and fertilizer recommendations, while the ORM simplified database interactions with SQLite, ensuring data integrity and reducing complexity.

Django's authentication features provided a secure login process, which was crucial for protecting sensitive user data. This robust framework allowed the development team to build a feature-rich, user-friendly application while maintaining high standards of security and maintainability throughout the project.

3. SQLite Database

SQLite is a self-contained, serverless, zero-configuration, transactional SQL database engine widely used for small to medium-sized applications due to its simplicity and lightweight nature. It operates directly on disk files, making it easy to

integrate into applications without the overhead of a separate database server. SQLite is known for its speed and cross-platform compatibility, ensuring that databases can be used seamlessly across different environments.

In your project, SQLite was used as the database backend to store user data, including login credentials and input parameters for crop and fertilizer predictions. Its lightweight nature made it an excellent choice for a web application that needed to handle moderate data volumes without the complexity of managing a full-fledged database server. The integration with Django's ORM facilitated seamless interaction with the SQLite database, allowing for efficient database migrations, model creation, and queries.

Using SQLite also contributed to faster development cycles, as the team could quickly set up the database schema and focus on implementing application features without worrying about server configuration and management. The choice of SQLite was instrumental in maintaining a reliable data storage solution, allowing for quick access and manipulation of user data necessary for providing accurate crop and fertilizer recommendations.

4. Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) that simplifies the management of packages, environments, and applications in the Anaconda distribution, which is widely used for data science and machine learning projects. Anaconda provides tools for package management and deployment of Python and R applications, making it easier for developers to manage dependencies and environments effectively.

With Anaconda Navigator, users can easily create and manage different environments, ensuring that projects have their dependencies isolated. The interface allows for straightforward installation, updating, and removal of packages from the Anaconda repository, ensuring access to the latest versions of libraries. The user-friendly GUI is designed for intuitive navigation, making it accessible for users of all

experience levels.

In the context of your project, Anaconda Navigator was used to manage the development environment and dependencies required for building the crop and fertilizer recommendation system. By creating a dedicated environment for the project, the development team could isolate it from other Python projects on their systems, minimizing potential conflicts. The ease of installing necessary packages such as Django and machine learning libraries streamlined the setup process, ensuring that all team members worked with consistent versions and reducing compatibility issues.

Additionally, Anaconda Navigator's integration with Jupyter Notebook provided an interactive environment for exploratory data analysis and model evaluation, allowing team members to visualize data and experiment with different algorithms before finalizing the implementation in Django. Overall, Anaconda Navigator significantly enhanced the efficiency of the development process, enabling the team to focus on building a robust web application without the overhead of managing dependencies manually.

5.6. CODE IMPLEMENTATION

Step 1: Set up the home page

The home page in our application vehicle speed estimation, featuring navigation options (Home, About, Features), and highlights user-friendliness, cross- platform compatibility, and data security. Here's the code for setting up the home page:

Code Snippet:

```
<!-- About -->
    <h3 class="mb-4 tm-app-feature-title">About Our Platform</h3>
    <p class="tm-app-feature-description">Our platform leverages
advanced machine learning and computer vision techniques to deliver accurate
vehicle speed estimation from video footage. We aim to revolutionize traffic
monitoring by providing real-time insights that can be used for safety,
compliance, and efficiency purposes.</p></div>
<!-- Features -->
    <div class="row" id="tmFeatures">
    <div class="col-lg-4">
        <p class="text-center">Designed with simplicity in mind, the platform
offers an intuitive interface for easy navigation,
        ensuring a seamless experience for both technical and non-technical
users.</p>
        <h3 class="tm-feature-name">Cross-Platform Compatibility</h3>
        </div>
        <p class="text-center">Accessible from any device, our web-based
solution works across all platforms,
        enabling users to upload, analyze, and retrieve reports from
anywhere.
        <div class="tm-bg-white-transparent tm-feature-box">
        <h3 class="tm-feature-name">Data Security and Privacy</h3>
        <p class="text-center">Your video data is processed securely, with
strict protocols in place to protect user privacy,
        ensuring that sensitive information is handled responsibly.</p>
    </div>
    </div>
    <div class="col-12 text-white text-center tm-copyright-text">
    Copyright &copy; 2024 Page.
    </div>
</div>
```

```

<nav class="navbar navbar-expand-lg navbar-light tm-bg-white-transparent
tm-navbar">
    <span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav">
        <li class="nav-item active">
            <div class="tm-nav-link-highlight"></div>
            <a class="nav-link" href="#">Home <span class="sr-
only">(current)</span></a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{ % url 'about' % }">About</a>
        </li>
        <li class="nav-item">
            <div class="tm-nav-link-highlight"></div>
            <a class="nav-link" href="{ % url 'features' % }">Features</a>
        </li>
        { % if user.is_authenticated % }
        <li class="nav-item">
            <div class="tm-nav-link-highlight"></div>
            <a class="nav-link" href="{ % url 'upload' % }">Upload
Video</a>
        </li>
        <a href="{ % url 'logout' % }">Logout</a>
        { % else % }
        <li class="nav-item">
            <div class="tm-nav-link-highlight"></div>
            <a class="nav-link" href="{ % url 'register' % }">Sign Up</a>
        </li>
        <li class="nav-item">
            <div class="tm-nav-link-highlight"></div>
            <a class="nav-link" href="{ % url 'login' % }">Sign in</a>
        </li>
    </ul>
</nav>

```

Step 2: Set up the registration and login page

The user **registration page** in our application is a critical component that facilitates the users to create an account by entering their first name, last name, username, email, and password. After confirming the password, they can register by clicking the Register button. The **login page** allows users to access their accounts by entering their username and password. Once the credentials are provided, the user can click the "Login" button to authenticate and proceed to the platform.

Code Snippet:

```
< --Sign up -->
<form action="#" method="POST" id="tmContactForm" > { % csrf_token % }
    <input type="text" id="contact_name" name="fname" class="form-
control rounded-0" placeholder="First Name" required=""></div>
    <input type="text" id="contact_name" name="lname" class="form-
control rounded-0" placeholder="Last Name" required=""></div>
    <input type="text" id="contact_name" name="uname" class="form-
control rounded-0" placeholder="User Name" required=""> </div>
    <input type="email" id="contact_email" name="Email" class="form-
control rounded-0" placeholder="Email" required=""> </div>
    <input type="password" id=" password" name="pass" class="form-
control rounded-0" placeholder="Password" required=""></div>
    <input type="password" id="confirm_password" name="cpass"
class="form-control rounded-0" placeholder="Confirm Password" required="">
    <button type="submit" class="btn btn-primary tm-btn-submit rounded-
0"> Register</button>
< --Sign in -->
    <input type="text" id="contact_name" name="uname" class="form-
control rounded-0" placeholder="User Name" required=""></div>
    <input type="password" id="password " name="pass" class="form-
control rounded-0" placeholder="Password" required=""></div>
    <div class="text-center">
        <button type="submit" class="btn btn-primary tm-btn-submit rounded-
0"> Login </button> </div></form></div></section>
```

Step 3: Set up the upload page

The user interface where users can upload video files for processing. The interface features a simple design with an upload button that allows the selection of video files. This part of the system might also involve handling various video file formats and processing them on the server side, such as compression or format conversion.

Code Snippet:

```
<!--Upload -->
<section class="row" id="tmHome">
  <div class="col-12 tm-home-container">
    <div class="text-white tm-home-left">
      <form action="/upload/" method="post" enctype="multipart/form-data"
        class="tm-bg-black-transparent text-center tm-services-header"
style="width: 600px;">
        { % csrf_token % }
        <h2 class="tm-home-title " style="white-space: nowrap; text-align:
center;"><b>Upload Video </b> </h2>
        <label for="video" style="color: azure;">
          <h4>Select a video file:</h4>
        </label>
        <div class="file-upload">
          <input type="file" id="video" name="video" accept="video/*"
required> <span>Choose</span>
          <input type="submit" value="Upload" class="btn btn-primary"
style="border-radius: 15px;"
          onclick="showLoading();">
        </div>
      <!-- Loading overlay -->
      <div class="loading-container" id="loading-container">
        <div class="loader"></div>
        <p class="loading-text">Processing your video, please wait...</p>
      </div>
    </div>
  </div>
</section>
```

Step 4: Set up the video capture and haar cascade

The video capture process begins by initializing `cv2.VideoCapture()` to read frames from a video source, and loading a pre-trained Haar Cascade classifier with `cv2.CascadeClassifier()` for vehicle detection. Each frame is analyzed to identify vehicle patterns, enabling real-time tracking as part of the speed detection workflow

Code Snippet :

```
import pandas as pd
import numpy as np
from django.conf import settings
from pathlib import Path
import os
cascade_path =
r'C:\Users\ADMIN\Swe.Project\vehicle_speeddetection\haarCascade_cars.xml'
if not os.path.exists(cascade_path):
    raise FileNotFoundError(f"Cascade file not found: {cascade_path}")
carCascade = cv2.CascadeClassifier(cascade_path)
if carCascade.empty():
    raise ValueError("Failed to load cascade classifier.")
def estimate_speed(location1, location2):
    d_pixels = math.sqrt(math.pow(location2[0] - location1[0], 2) +
def process_video(full_video_path):
    matchCarID = None
    [x1, y1, w1, h1] = carLocation1[carID]
    [x2, y2, w2, h2] = carLocation2[carID]
    if speed[carID] is not None and speed[carID] > speed_limit:
        color = (0, 0, 255) # Red for vehicles exceeding the speed limit
        cv2.putText(resultImage, f"{int(speed[carID])} km/h (Over Limit)",
                    (int(x1 + w1 / 2), int(y1 - 5)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)
    elif speed[carID] is not None:
        cv2.putText(resultImage, f"{int(speed[carID])} km/h",
                    (int(x1 + w1 / 2), int(y1 - 5)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255, 255, 255), 2)
    return output_path, unique_over_limit_vehicles
```

Step 5: Set up the resultant page

The result page displays the processed video and calculated vehicle speed data. In Django, a `result.html` template is created to render this information for the user. The view function in Django retrieves the processed video path and speed data. This setup allows users to see the speed analysis directly on the result page in an organized and visually accessible way:

Code Snippet:

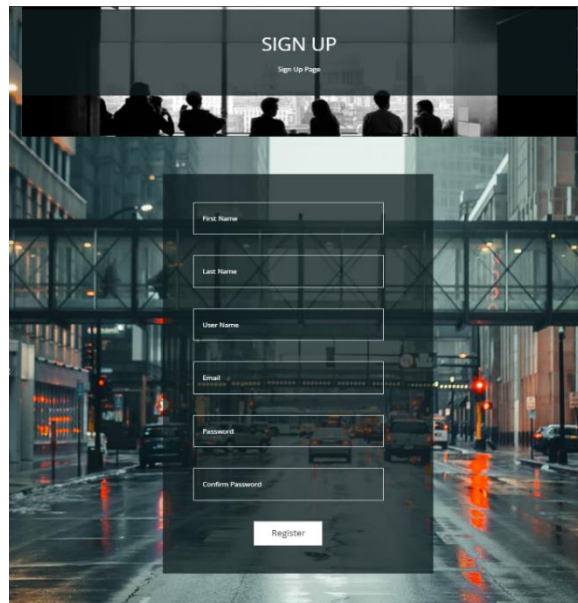
```
<section class="row" id="tmHome">
    <!-- Loading animation -->
    <div class="loading-container" id="loading-container">
        <div class="loader"></div>
        <h1>Processed Video</h1>
        <video id="processed-video" controls autoplay loop
style="display: none;">
            <source src="{{ processed_video_path }}"
type="video/mp4">
            Your browser does not support the video tag.
        </video>
    </div>
    <!-- Display the over-speeding vehicles -->
    {% if over_limit_vehicles %}
    <h2>Over Speeding Vehicles</h2>
    <ul>
        {% for vehicle_id, speed in over_limit_vehicles %}
        <li>Vehicle {{ vehicle_id }}: {{ speed }} km/h</li>
        {% endfor %}
    </ul>
    {% else %}
    <p>No speeding vehicles detected.</p>
    {% endif %}
</section>
```

5.8. RESULT

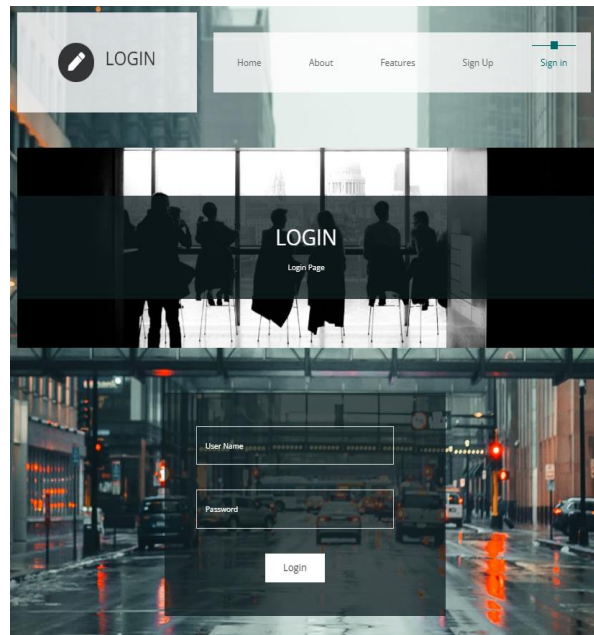
The vehicle speed detection project combines computer vision techniques with a Django web application to analyze video footage of vehicles. Users can upload videos through an intuitive interface, receiving processed outputs with annotated speed overlays, making it a valuable tool for traffic monitoring and enforcement.

App Flow:

Register Page



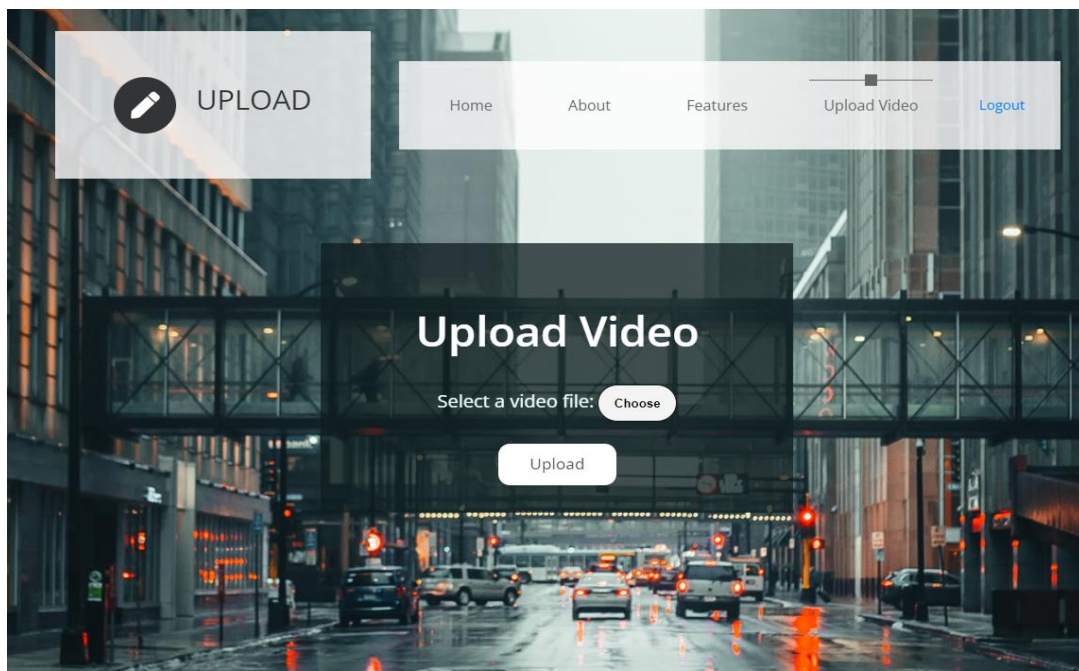
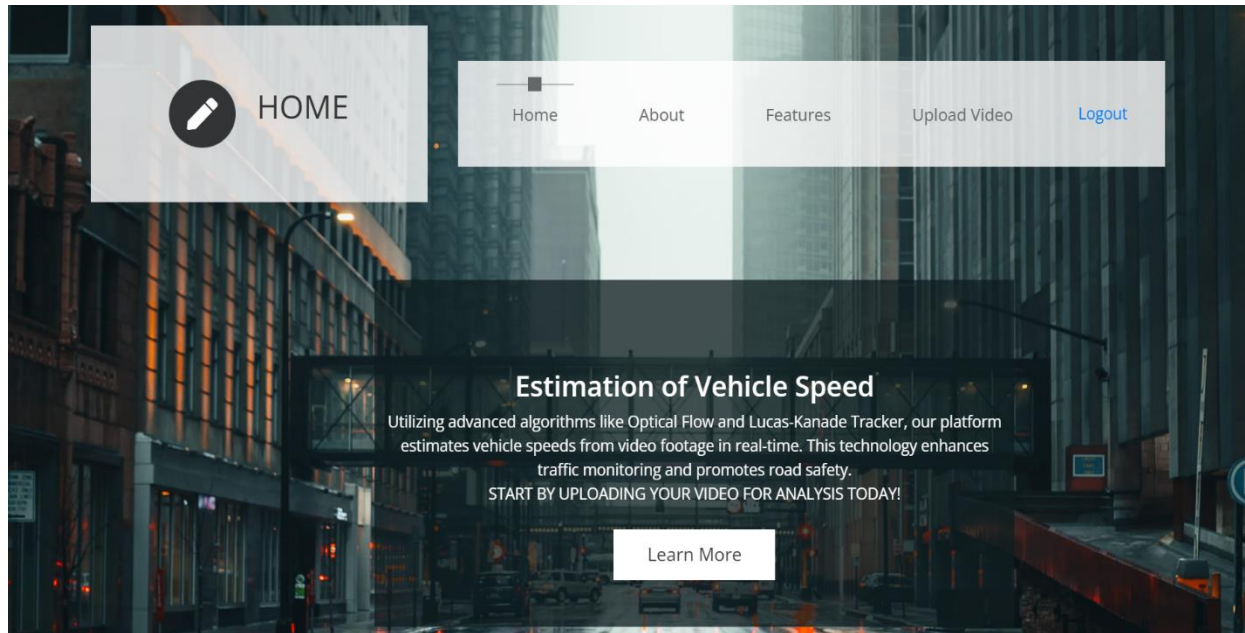
A screenshot of the 'SIGN UP' page. The page has a dark background with a city street scene. At the top, it says 'SIGN UP' and 'Sign Up Page'. Below this, there are input fields for 'First Name', 'Last Name', 'User Name', 'Email', 'Password', and 'Confirm Password'. At the bottom, there is a 'Register' button.



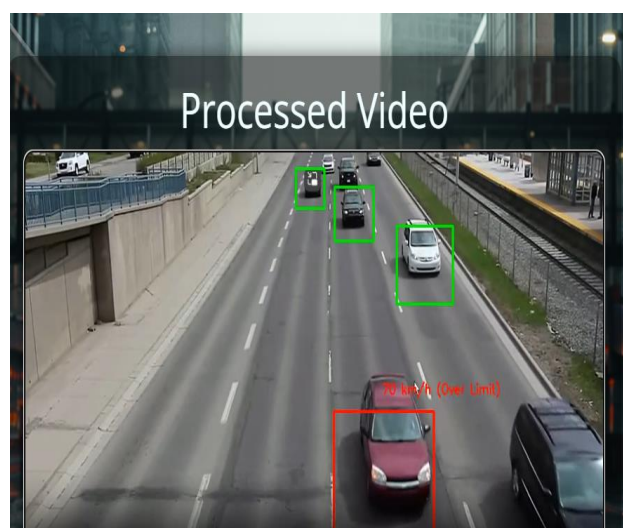
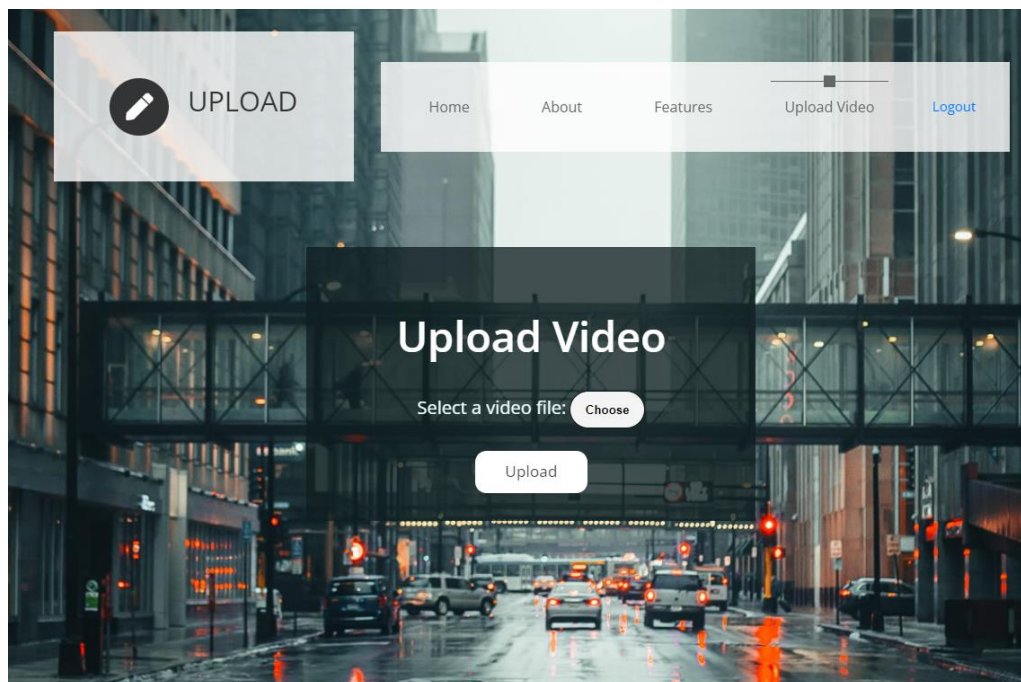
A screenshot of the 'LOGIN' page. The page has a dark background with a city street scene. At the top, there is a 'LOGIN' button with a key icon. Below this, there are input fields for 'User Name' and 'Password'. At the bottom, there is a 'Login' button. The page also has a navigation bar with links for 'Home', 'About', 'Features', 'Sign Up', and 'Sign In'.

Login Page

Initially, the user will have to register if they already have an account or they will have to sign up with the essential details like Username, email and password. Once the user signs in, the user authentication takes place and all the user data will be stored securely in the SQLite for future access. Then the user will be navigated to the home page of our application.



In the home page, after clicking the Upload video it redirect to the Upload page where user can upload there input video.



Once the user clicks the logout option, the account will be logged out from the application. If the user tries to use the app again after logging out, he/she will have to log in using the registered email and password.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

The vehicle speed detection project has successfully integrated advanced computer vision techniques with a Django web application to create an effective and efficient solution for monitoring vehicle speeds through video analysis. This project serves as a significant contribution to the field of traffic management and enforcement, addressing the growing need for reliable methods to analyze vehicular behavior on the roads.

By employing the Haar Cascade algorithm for vehicle detection, the system demonstrates the ability to accurately identify vehicles in a variety of conditions, including differing lighting, weather patterns, and diverse vehicle types. The selection of this algorithm was crucial, as its performance directly impacts the overall effectiveness of the speed detection system. Coupled with the Lucas-Kanade optical flow method, the project accurately estimates vehicle speeds in real-time by tracking the motion of vehicles across frames. This combination of algorithms ensures that the system can provide precise speed calculations, which are essential for traffic enforcement applications.

The project outcomes are particularly noteworthy, showcasing high accuracy in both vehicle detection and speed estimation. Through extensive testing in various scenarios, the system has shown a low error rate in speed calculations compared to established ground truth data. The capability to handle multiple vehicle detections within a single frame enhances the robustness of the application, making it suitable for busy urban environments where numerous vehicles may be present simultaneously.

The user interface of the Django web application was designed with the end user in mind, facilitating easy interaction and making the technology accessible to a wider audience. Users can effortlessly upload their recorded video footage for analysis, and the processed outputs, which include annotated video

files with speed overlays, provide clear and actionable information. This user-centric approach not only broadens the potential user base, including law enforcement agencies, traffic management departments, and private individuals, but also encourages the adoption of technology for improved road safety and traffic monitoring.

Moreover, the project's impact extends beyond mere vehicle speed detection. By providing law enforcement and traffic management authorities with a reliable tool for speed analysis, the system contributes to enhanced public safety measures. Accurate speed detection can lead to more effective enforcement of speed limits, which is a critical factor in reducing traffic accidents and improving overall road safety. The ability to present clear, annotated video evidence supports enforcement actions and helps to educate the public about safe driving practices.

In conclusion, this vehicle speed detection project not only demonstrates the technical feasibility of applying computer vision techniques for real-world traffic monitoring. The successful integration of computer vision with a web-based platform paves the way for future innovations in traffic management solutions. Future work could involve incorporating machine learning algorithms for improved detection accuracy, developing features for vehicle classification, and analyzing driving behavior patterns. These enhancements could further elevate the application's capabilities, making it an even more powerful tool for traffic analysis and management.

Overall, the vehicle speed detection project stands as a robust example of how technological advancements can be harnessed to address critical challenges in transportation safety and efficiency. By providing an accessible, accurate, and efficient means of speed analysis, the project contributes significantly to the ongoing efforts to improve road safety and promote responsible driving behavior.

6.2 FUTURE SCOPE

The vehicle speed detection project opens up several promising avenues for future enhancements and applications. Here are some potential future scopes:

1. Integration of Machine Learning Algorithms

Future iterations of the project could incorporate machine learning techniques to improve the accuracy of vehicle detection and speed estimation. By training models on diverse datasets, the system could learn to recognize a wider variety of vehicles and adapt to different environmental conditions.

2. Vehicle Classification

Expanding the system to include vehicle classification would allow it to differentiate between types of vehicles (e.g., cars, trucks, motorcycles). This could provide valuable insights into traffic patterns and contribute to more tailored traffic management strategies.

3. Behavior Analysis

The project could evolve to analyze driving behavior, such as lane changes, acceleration patterns, and braking. This feature could help identify unsafe driving practices, enabling targeted interventions to enhance road safety.

4. Real-Time Speed Monitoring

Developing capabilities for real-time speed monitoring could be beneficial for applications in traffic management. Integrating the system with live camera feeds would allow for immediate alerts and interventions when vehicles exceed speed limits.

5. Data Analytics and Reporting

Implementing data analytics features could allow users to generate reports on traffic patterns, average speeds, and peak traffic times. This data could be invaluable for urban planners and traffic management authorities in making informed decisions.

6. Integration with Other Traffic Management Systems

Future developments could include integrating the speed detection system with existing traffic management infrastructures, such as traffic lights and automated ticketing systems, to create a more cohesive traffic management environment.

7. Collaboration with IoT Devices

Exploring the use of Internet of Things (IoT) devices, such as smart traffic cameras and sensors, could enhance the system's capabilities. These devices could provide additional data points for analysis and improve the overall accuracy of speed detection.

8. Research and Development

Continued research into new algorithms and techniques in computer vision and machine learning could lead to innovations that further enhance the system's performance and applicability in diverse contexts

By pursuing these future scopes, the vehicle speed detection project can evolve into a comprehensive traffic monitoring solution, offering valuable insights and tools to improve road safety, traffic management, and urban planning.

APPENDICES

SOURCE CODE

Index.html

```
{ % load static % }
{ % block content % }
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta http-equiv="X-UA-Compatible" content="ie=edge" />
<title>Home Page</title>
<link rel="stylesheet" href="{ % static 'css/bootstrap.min.css' % }" />
<link rel="stylesheet" href="{ % static 'css/templatemo-style.css' % }" />
</head>
<body>
<div class="parallax-window" data-parallax="scroll" data-image-src="{ % static
'img/bg-01.jpg' % }">
<div class="container-fluid">
<div class="row tm-brand-row">
<div class="col-lg-4 col-11">
<div class="tm-brand-container tm-bg-white-transparent">
<i class="fas fa-2x fa-pen tm-brand-icon"></i>
<div class="tm-brand-texts">
<h1 class="text-uppercase tm-brand-name">HOME</h1>
<p class="small"></p>
</div>
</div>
</div>
<div class="col-lg-8 col-1">
<div class="tm-nav">
```

```

<nav class="navbar navbar-expand-lg navbar-light tm-bg-white-transparent tm-
navbar">
<button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarNav"
aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
<span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarNav">
<ul class="navbar-nav">
<li class="nav-item active">
<div class="tm-nav-link-highlight"></div>
<a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
</li>
<li class="nav-item">
<div class="tm-nav-link-highlight"></div>
<a class="nav-link" href="{ % url 'about' % }">About</a>
</li>
<li class="nav-item">
<div class="tm-nav-link-highlight"></div>
<a class="nav-link" href="{ % url 'features' % }">Features</a>
</li>
{ % if user.is_authenticated % }
<li class="nav-item">
<div class="tm-nav-link-highlight"></div>
<a class="nav-link" href="{ % url 'upload' % }">Upload Video</a>
</li>
<a href="{ % url 'logout' % }">Logout</a>
{ % else % }
<li class="nav-item">
<div class="tm-nav-link-highlight"></div>
<a class="nav-link" href="{ % url 'register' % }">Sign Up</a>
</li>
<li class="nav-item">

```

```

<div class="tm-nav-link-highlight"></div>
<a class="nav-link" href="{ % url 'login' % }">Sign in</a>
</li>
{ % endif % }
</ul>
<section class="row" id="tmHome">
<div class="col-12 tm-home-container">
<div class="text-white tm-home-left">
<p class="text-uppercase tm-slogan">
</p>
<div class="tm-bg-black-transparent text-center tm-services-header" style=" width:
760px;">
<p style="color: white;">
<h2 style="white-space: nowrap;"><b>Estimation of Vehicle Speed</b></h2>
Utilizing advanced algorithms like Optical Flow and Lucas-Kanade Tracker, our
platform estimates vehicle speeds from video footage in real-time.
This technology enhances traffic monitoring and promotes road safety.<br>START
BY UPLOADING YOUR VIDEO FOR ANALYSIS TODAY!
<br> <br><a href="#tmFeatures" class="btn btn-primary">Learn More</a>
</p>
</div></section>
<!-- Features -->
<div class="row" id="tmFeatures">
<div class="col-lg-4">
<div class="tm-bg-white-transparent tm-feature-box">
<h3 class="tm-feature-name">User-Friendly Interface</h3>
<div class="tm-feature-icon-container">
<i class="fas fa-3x fa-server"></i>
</div>
<p class="text-center">Designed with simplicity in mind, the platform offers an
intuitive interface for easy navigation,
ensuring a seamless experience for both technical and non-technical users.</p>
</div>
</div>

```



```

<div class="col-lg-4">
<div class="tm-bg-white-transparent tm-feature-box">
<h3 class="tm-feature-name">Cross-Platform Compatibility</h3>
<div class="tm-feature-icon-container">
<i class="fas fa-3x fa-headphones"></i>
</div>
<p class="text-center">Accessible from any device, our web-based solution works
across all platforms,
enabling users to upload, analyze, and retrieve reports from anywhere.
</p>
<div class="col-lg-4">
<div class="tm-bg-white-transparent tm-feature-box">
<h3 class="tm-feature-name">Data Security and Privacy</h3>
<div class="tm-feature-icon-container">
<i class="fas fa-3x fa-satellite-dish"></i>
</div>
<p class="text-center">Your video data is processed securely, with strict protocols in
place to protect user privacy,
ensuring that sensitive information is handled responsibly.
</p>
<footer class="row">
<p class="col-12 text-white text-center tm-copyright-text">
Copyright &copy; 2024 Page.
</p>
</footer>
</div>
<!-- .container-fluid -->
</div>
<script src="{ % static 'js/jquery.min.js' % }"></script>
<script src="{ % static 'js/parallax.min.js' % }"></script>
<script src="{ % static 'js/bootstrap.min.js' % }"></script>
{ % endblock % }
</body></html>

```

Sigup.html

```
<body>
<div class="page-container">
<!-- Navigation Bar -->
<nav>
<ul>
<div class="center-nav" style="padding-left: 20px;">
<li><a href="index#home">Home</a></li>
<li><a href="index#about">About</a></li>
<li><a href="index#recommendation">Recommendation</a></li>
</div>
<div class="right-nav">
</ul>
</nav>
<!-- Signup Section -->
<div class="main-content">
<section id="signup"><br><br>
<h2>Signup</h2>
<form method="post" action="{ % url 'signup' % }">
{ % csrf_token % }
<label for="username">Username:</label>
<input type="text" id="username" name="username" required>
<label for="email">Email ID:</label>
<input type="email" id="email" name="email" required>
<label for="password">Password:</label>
<input type="password" id="password" name="password" required>
<label for="confirm-password">Confirm Password:</label>
<input type="password" id="confirm_password" name="confirm_password" required>
<button type="submit">Signup</button>
</form>
<!-- Already signed up section -->
<div class="login-redirect">
```

```

<p>Already have an account? <a href="login">Login here</a></p>
</section>
</div>
{% for i in messages %}
<script>
alert("{i}");
</script>
{% endfor %}
</div>
</body>

```

Login.html

```

<section class="row" id="tmServices">
<div class="col-12">
<div class="parallax-window tm-services-parallax-header tm-testimonials-parallax-
header"
data-parallax="scroll"
data-image-src="{% static 'img/people.jpg' %}">

<div class="tm-bg-black-transparent text-center tm-services-header tm-testimonials-
header">
<h2 class="text-uppercase tm-services-page-title tm-testimonials-page-title "
style="text-align: center;">Login</h2>
<p class="tm-services-description mb-0 small">
Login Page
</p>
</section>
<section class="row tm-contact-row justify-content-center">
<div class="col-lg-6 col-md-8">
<form action="#" method="POST" id="tmContactForm" class="tm-bg-black-
transparent tm-contact-form">
{% csrf_token %}
<div class="form-group">

```

```

<input type="text" id="contact_name" name="uname" class="form-control rounded-0"
placeholder="User Name" required="">
</div>
<div class="form-group">
<input type="password" id="contact_email" name="pass" class="form-control
rounded-0" placeholder="Password" required="">
<div class="text-center">
<button type="submit" class="btn btn-primary tm-btn-submit rounded-0">
Login
</form>
</section>
{% for i in messages %}
<script>alert("{ {i} }")</script>
{% endfor %}

```

Upload.html

```

<section class="row" id="tmHome">
<div class="col-12 tm-home-container">
<div class="text-white tm-home-left">
<form action="/upload/" method="post" enctype="multipart/form-data"
class="tm-bg-black-transparent text-center tm-services-header" style="width: 600px;">
{% csrf_token %}
<h2 class="tm-home-title " style="white-space: nowrap; text-align:
center;"><b>Upload Video</b></h2>
<label for="video" style="color: azure;">
<h4>Select a video file:</h4>
</label>
<div class="file-upload">
<input type="file" id="video" name="video" accept="video/*" required>
<span>Choose</span>
</div>
<br><br>
<input type="submit" value="Upload" class="btn btn-primary" style="border-radius:

```

```

15px;"
onclick="showLoading();">
</form>
<!-- Loading overlay -->
<div class="loading-container" id="loading-container">
<div class="loader"></div>
<p class="loading-text">Processing your video, please wait...</p>
</section>

```

Upload_success.html

```

<section class="row" id="tmHome">
<!-- Loading animation -->
<div class="loading-container" id="loading-container">
<div class="loader"></div>
</div>
<div class="video-container tm-bg-black-transparent text-center tm-services-header">
<h1>Processed Video</h1>
<video id="processed-video" controls autoplay loop style="display: none;">
<source src="{{ processed_video_path }}" type="video/mp4">
Your browser does not support the video tag.
</video>
</div>
<!-- Display the over-speeding vehicles -->
<div class="video-container tm-bg-black-transparent text-center tm-services-header">
<div class="over-limit-vehicles">
{% if over_limit_vehicles %}
<h2>Over Speeding Vehicles</h2>
<ul>
{% for vehicle_id, speed in over_limit_vehicles %}
<li>Vehicle {{ vehicle_id }}: {{ speed }} km/h</li>
{% endfor %}
</ul>

```

```
{% else %}  
<p>No speeding vehicles detected.</p>  
{% endif %}</div>  
</section>
```

Views.py

```
from pathlib import Path  
import cv2  
from django.conf import settings  
from django.core.files.storage import default_storage  
from django.http import JsonResponse  
from django.shortcuts import render, redirect  
from django.contrib import messages  
from django.contrib.auth.models import User, auth  
from .forms import VideoUploadForm  
  
def index(request):  
    return render(request, "index.html")  
  
def about(request):  
    return render(request, "about.html")  
  
def features(request):  
    return render(request, "features.html")  
  
def upload(request):  
    return render(request, "upload.html")  
  
def logout(request):  
    auth.logout(request)  
    return redirect('index')
```

```

def register(request):
    if request.method == "POST":
        first = request.POST['fname']
        last = request.POST['lname']
        uname = request.POST['uname']
        email = request.POST['Email']
        p1 = request.POST['pass']
        p2 = request.POST['cpass']

    if p1 == p2:
        if User.objects.filter(email=email).exists():
            messages.info(request, "Email Exists")
        elif User.objects.filter(username=uname).exists():
            messages.info(request, "Username available")
        else:
            user = User.objects.create_user(first_name=first, last_name=last, username=uname,
            email=email, password=p1)
            user.save()
            return redirect('login')
        else:
            messages.info(request, "Password not matched")
            return render(request, "register.html")
            return render(request, "register.html")

def login(request):
    if request.method == "POST":
        uname = request.POST['uname']
        ps = request.POST['pass']
        user = auth.authenticate(username=uname, password=ps)
        if user is not None:
            auth.login(request, user)
            return redirect('index')
        else:

```

```
messages.info(request, "Invalid Credentials")
return render(request, "login.html")
```

```
import cv2
import numpy as np
from django.shortcuts import render
from django.conf import settings
from .forms import VideoUploadForm
from pathlib import Path
import subprocess
import os
```

```
def convert_video_to_mp4(input_video_path):
    output_video_path = str(input_video_path).replace(input_video_path.suffix, '.mp4')
    if input_video_path.suffix != '.mp4':
        try:
            command = [
                'ffmpeg', '-i', str(input_video_path),
                '-c:v', 'libx264', '-c:a', 'aac', '-strict', 'experimental',
                str(output_video_path)
            ]
            subprocess.run(command, check=True)
        except subprocess.CalledProcessError as e:
            print(f"Error in video conversion: {e}")
            raise ValueError("Video conversion failed. Please ensure the file is valid.")
```

```
return Path(output_video_path)
import cv2
import dlib
from django.conf import settings
from pathlib import Path
import os
```



```

cascade_path =
r'C:\Users\ADMIN\Downloads\final\Swe.Project\vehicle_speed_detection\haarCascad
e_cars.xml'
if not os.path.exists(cascade_path):
raise FileNotFoundError(f"Cascade file not found: {cascade_path}")

carCascade = cv2.CascadeClassifier(cascade_path)

if carCascade.empty():
raise ValueError("Failed to load cascade classifier.")

WIDTH = 1280
HEIGHT = 720

def estimate_speed(location1, location2):
d_pixels = math.sqrt(math.pow(location2[0] - location1[0], 2) + math.pow(location2[1]
- location1[1], 2))
ppm = 8.8 # Pixels per meter
d_meters = d_pixels / ppm
fps = 18
speed = d_meters * fps * 3.6 # Convert m/s to km/h
return speed

def process_video(full_video_path):
cap = cv2.VideoCapture(str(full_video_path))
frame_rate = cap.get(cv2.CAP_PROP_FPS)
currentCarID = 0
carTracker = {}
carLocation1 = {}
carLocation2 = {}
speed = [None] * 1000
output_path = Path(settings.MEDIA_ROOT) / 'processed_video.mp4'
fourcc = cv2.VideoWriter_fourcc(*'mp4v')

```

```

out = cv2.VideoWriter(str(output_path), fourcc, frame_rate, (WIDTH, HEIGHT))
frameCounter = 0
speed_limit = 60 # Define the speed limit (in km/h)
unique_over_limit_vehicles = { }
while cap.isOpened():
    ret, image = cap.read()
    if not ret:
        break

    image = cv2.resize(image, (WIDTH, HEIGHT))
    resultImage = image.copy()
    frameCounter += 1

    carIDtoDelete = []
    for carID in carTracker.keys():
        trackingQuality = carTracker[carID].update(image)
        if trackingQuality < 7:
            carIDtoDelete.append(carID)

    for carID in carIDtoDelete:
        carTracker.pop(carID, None)
        carLocation1.pop(carID, None)
        carLocation2.pop(carID, None)

    if frameCounter % 5 == 0:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        cars = carCascade.detectMultiScale(gray, 1.1, 13, 18, (24, 24))
        for (_x, _y, _w, _h) in cars:
            x, y, w, h = int(_x), int(_y), int(_w), int(_h)
            x_bar = x + 0.5 * w
            y_bar = y + 0.5 * h
            matchCarID = None
            for carID in carTracker.keys():

```

```

trackedPosition = carTracker[carID].get_position()
t_x = int(trackedPosition.left())
t_y = int(trackedPosition.top())
t_w = int(trackedPosition.width())
t_h = int(trackedPosition.height())
t_x_bar = t_x + 0.5 * t_w
t_y_bar = t_y + 0.5 * t_h
if ((t_x <= x_bar <= (t_x + t_w)) and (t_y <= y_bar <= (t_y + t_h)) and
(x <= t_x_bar <= (x + w)) and (y <= t_y_bar <= (y + h))):
    matchCarID = carID
    if matchCarID is None:
        tracker = dlib.correlation_tracker()
        tracker.start_track(image, dlib.rectangle(x, y, x + w, y + h))
        carTracker[currentCarID] = tracker
        carLocation1[currentCarID] = [x, y, w, h]
        currentCarID += 1

for carID in carTracker.keys():
    trackedPosition = carTracker[carID].get_position()
    t_x, t_y, t_w, t_h = int(trackedPosition.left()), int(trackedPosition.top()),
    int(trackedPosition.width()), int(trackedPosition.height())
    carLocation2[carID] = [t_x, t_y, t_w, t_h]

color = (0, 255, 0) # Default color green
[x1, y1, w1, h1] = carLocation1[carID]
[x2, y2, w2, h2] = carLocation2[carID]
carLocation1[carID] = [x2, y2, w2, h2]

if [x1, y1, w1, h1] != [x2, y2, w2, h2]:
    if (speed[carID] is None or speed[carID] == 0) and y1 >= 275 and y1 <= 285:
        speed[carID] = estimate_speed([x1, y1, w1, h1], [x2, y2, w2, h2])

if speed[carID] is not None and speed[carID] > speed_limit:

```

```

color = (0, 0, 255) # Red for vehicles exceeding the speed limit
if carID not in unique_over_limit_vehicles: # Check if vehicle already recorded
    unique_over_limit_vehicles[carID] = int(speed[carID])

cv2.putText(resultImage, f"{int(speed[carID])} km/h (Over Limit)",
(int(x1 + w1 / 2), int(y1 - 5)),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)
elif speed[carID] is not None:
cv2.putText(resultImage, f"{int(speed[carID])} km/h",
(int(x1 + w1 / 2), int(y1 - 5)),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255, 255, 255), 2)
cv2.rectangle(resultImage, (t_x, t_y), (t_x + t_w, t_y + t_h), color, 4)
out.write(resultImage)
cap.release()
out.release()
return output_path, unique_over_limit_vehicles

def upload_video(request):
    if request.method == 'POST':
        form = VideoUploadForm(request.POST, request.FILES)
        if form.is_valid():
            video = form.cleaned_data['video']
            unique_video_name =
            f"{Path(video.name).stem}_{uuid.uuid4().hex}_{Path(video.name).suffix}"
            video_path = default_storage.save(unique_video_name, video)
            full_video_path = Path(settings.MEDIA_ROOT) / video_path

            try:
                processed_video_path, unique_over_limit_vehicles = process_video(full_video_path)
            except Exception as e:
                return render(request, 'upload_error.html', {'error': str(e)})

            try:
                h264_video_path = convert_video_to_h264(processed_video_path)
            except subprocess.CalledProcessError as e:

```

```

return render(request, 'upload_error.html', {'error': 'Video conversion failed: ' + str(e)})
video_url = f"{settings.MEDIA_URL}{os.path.basename(video_path)}"
h264_video_url = f"{settings.MEDIA_URL}{os.path.basename(h264_video_path)}"
return render(request, 'upload_success.html', {
    'video_path': video_url,
    'processed_video_path': h264_video_url,
    'over_limit_vehicles': unique_over_limit_vehicles.items() # Pass unique vehicles and
    their speeds
})
else:
form = VideoUploadForm()
return render(request, 'upload.html', {'form': form})
def convert_video_to_h264(input_video_path):
unique_filename = f"{input_video_path.stem}_{uuid.uuid4().hex}.mp4"
output_video_path = input_video_path.with_name(unique_filename)
command = [
'ffmpeg', '-i', str(input_video_path),
'-c:v', 'libx264', '-c:a', 'aac', '-strict', 'experimental',
str(output_video_path)
]
subprocess.run(command, check=True)
return output_video_path

```

REFERENCES

1. Patel, S., & Jha, R. (2023). Improving Vehicle Speed Estimation Accuracy Using Advanced Optical Flow Techniques. [Journal of Real-Time Image Processing]
2. Zhang, Y., Wang, X., & Liu, J. (2022). Real-time Vehicle Detection and Speed Estimation Using Haar Cascade and Optical Flow. [IEEE Transactions on Intelligent Transportation Systems]
3. Singh, A., & Gupta, R. (2023). Comparative Analysis of Optical Flow Methods for Vehicle Speed Estimation in Urban Environments. [International Journal of Computer Applications]
4. Fernandez, J., & Ruiz, M. (2022). An Enhanced Framework for Real-time Traffic Analysis Using OpenCV and Optical Flow. [Journal of Computer Vision and Image Processing]
5. Kumar, R., & Sharma, N. (2023). Vehicle Speed Detection Using Haar Cascade and Optical Flow: A Machine Learning Approach. [Journal of Image and Video Processing]
6. Brown, P., & Lewis, K. (2023). "Computer Vision Applications in Traffic Monitoring and Speed Analysis: A Review." Journal of Transportation and Visual Computing.
7. Cheng, H., & Zhao, T. (2022). "Improving Accuracy in Vehicle Speed Detection Through Feature Optimization in Optical Flow Algorithms." IEEE Access.
8. Rahman, M., & Lee, S. (2023). "Vehicle Detection and Speed Estimation in Real-World Scenarios Using Hybrid Machine Learning Models." Advances in Intelligent Transportation Systems.
9. Wang, L., & Chen, Z. (2022). "Implementation of Speed Estimation Techniques Using Deep Learning and Optical Flow Analysis."
10. Davis, R., & Patel, M. (2023). "A Scalable Framework for Automated Traffic Surveillance Using OpenCV and Machine Learning." International Journal of Intelligent Traffic Systems.

