A

Project Report On

Automotive: License Plate Recognition, Vehicle Counting and Parking Space Detection

By

Laxita Sojitra[CE046] [21CEUEG128]
Mahipal Suchar[CE047] [21CEUBG134]
of

B.Tech CE , Semester VI

Subject: System Design Practices

Guided by:

Prof. Jatayu H. Baxi
Assistant Professor
CE Department

Faculty of Technology
Department of Computer Engineering
Dharmsinh Desai University
College Road
Nadiad – 387001

CERTIFICATE

This is to certify that the practical/term work carried out

in the subject of

System Design Practice and recorded in this report

is the bonafide work of

Laxita Sojitra[CE046] [21CEUEG128]
Mahipal Suchar[CE047] [21CEUBG134]

of B.Tech semester VI in the branch of Computer Engineering during the academic year **2023 2024**.

Prof. Jatayu H. Baxi,
Assistant Professor,
CE Department,
Dharmsinh Desai University, Nadiad

Dr. C. K. Bhensdadia,
Head,
CE Department,
Dharmsinh Desai University, Nadiad

ACKNOWLEDGMENT

It is indeed a great pleasure to express our thanks and gratitude to all those who helped us during this project. This project would have materialized without the help of many who asked us good questions and rescued us from various red tape crises.

Theoretical knowledge is of no importance if one doesn't know the way of its implementation. We are thankful to our institute which provided us with an opportunity to apply our theoretical knowledge through the project. We feel obliged to submit this project as part of our curriculum.

We would like to take the opportunity to express our humble gratitude to our guide **Prof. Jatayu Baxi**, under whom we undertook our project. His constant guidance and willingness to share his vast knowledge made us enhance our knowledge and helped us to complete the assigned tasks to perfection. Without his effort, support, and astonishing testing ability this project may not have succeeded.

With Sincere Regards,
Laxita
Mahipal

TABLE OF CONTENTS :

Figure table:

1. INTRODUCTION

In response to the persistent challenges of urban transportation and parking management, our project presents an innovative solution that integrates three pivotal functionalities: License Plate Recognition (LPR), Vehicle Counting, and Parking Space Detection. Leveraging cutting-edge technologies such as computer vision and machine learning, our system aims to revolutionize how cities handle traffic flow and parking issues.

License Plate Recognition (LPR):

- **Functionality**: Utilizing advanced computer vision techniques, our system can accurately detect and recognize vehicle license plates.

- **Application**: The LPR feature serves multiple purposes, including law enforcement, security surveillance, toll collection, and parking management.

- **Tools**: To achieve precise license plate recognition, we employ a combination of cutting-edge tools and technologies:

➔ **YOLOv8:** We utilize the YOLO (You Only Look Once) version 8 object detection framework for efficient and accurate vehicle detection in images and videos.
➔ **Custom License Plate Detection Model:** We have developed a specialized license plate detection model trained on a dataset sourced from platforms like Roboflow, tailored to accurately locate license plates within vehicle images.
➔ **Computer Vision**: Various computer vision techniques such as image preprocessing, edge detection, and contour analysis are employed to enhance the accuracy of license plate detection.
➔ **Optical Character Recognition (OCR) Technology:** OCR algorithms are utilized to interpret and convert the extracted characters from license plates into machine-readable text, enabling automated processing and analysis of license plate data.
➔ **Video Input Implementation:** Our system is designed to handle video input streams, allowing for real-time processing of license plate information from surveillance cameras or traffic monitoring systems.

Vehicle Counting:

- **Functionality**: Our vehicle counting model employs real-time video analysis to track and count vehicles passing through designated areas.This process involves multiple stages, including vehicle detection, tracking, and counting.

- **Application**: This functionality provides valuable insights into traffic flow patterns, congestion levels, and peak hours, aiding urban planners and transportation authorities in making data-driven decisions for optimizing road networks and traffic management strategies.

- **Tools**: Our vehicle counting system utilizes a combination of powerful tools and technologies to achieve accurate and efficient vehicle counting in real-time:

➔ **Motion Detection Algorithms**: Motion detection algorithms are employed to identify moving objects within the video frames, enabling the initial detection of vehicles in the scene.
➔ **Computer Vision Techniques:** Various computer vision techniques such as image preprocessing, feature extraction, and object segmentation are utilized to enhance the accuracy and robustness of vehicle detection and tracking.
➔ **YOLOv8 Object Detection:** The YOLOv8 (You Only Look Once) object detection framework is utilized for efficient and accurate vehicle detection in video streams. This deep learning-based approach enables real-time processing of video data, ensuring timely and accurate vehicle counting.
➔ **SORT (Simple Online and Realtime Tracking):** The SORT algorithm is employed for multi-object tracking, enabling the system to assign unique IDs to vehicles and track their movements across consecutive video frames.
➔ **Custom Implementation:** We have developed a custom vehicle counting solution that integrates the aforementioned tools and technologies into a cohesive system. This solution is tailored to meet the specific requirements of our vehicle counting application, ensuring high performance and reliability.

Parking Space Detection:

- **Functionality**: The parking space detection model utilizes computer vision algorithms to effectively detect and classify available and occupied parking spaces within a given area.

- **Application**: This feature offers immediate information to drivers regarding parking space availability, thereby enhancing the parking experience, reducing search time, and mitigating traffic congestion in urban settings. By streamlining the parking process, it contributes to improved traffic flow efficiency and overall urban mobility.

- **Tools**: Our parking space detection system relies on a variety of tools and techniques to achieve accurate and efficient detection of parking spaces:

➔ **Image Processing Techniques:** Various image processing techniques, including grayscale conversion, Gaussian blurring, adaptive thresholding, and median blurring, are employed to enhance image quality and extract relevant features for parking space detection.

➔ **Object Detection Algorithms**: Advanced object detection algorithms such as YOLO (You Only Look Once) is utilized to detect and localize parking spaces within the input video frames. These algorithms enable precise identification and classification of parking spaces in real-time.

➔ **Custom Implementation**: We have developed a custom parking space detection solution that integrates the aforementioned tools and techniques into a cohesive system. This solution is tailored to meet the specific requirements of parking space detection in urban environments, ensuring reliable and efficient performance.

2. SOFTWARE REQUIREMENT SPECIFICATION:

2.1 Introduction:

Our project aims to address urban transportation and parking management challenges by integrating three key functionalities: license plate recognition, vehicle counting, and parking space detection.

2.1.1 Purpose:

The purpose of our project is to develop a comprehensive solution for urban transportation and parking management, leveraging advanced technologies to optimize resource allocation, enhance safety, and improve user convenience. By automating key processes such as license plate recognition, vehicle counting, and parking space detection, we aim to alleviate congestion, reduce travel times, and create a more sustainable and enjoyable urban experience for all stakeholders.

2.1.2 User Interface:

- Design and development of intuitive user interfaces for accessing license plate recognition results, vehicle count insights, and parking space availability information.
- Integration with existing urban mobility apps and platforms for seamless user interaction and data visualization.

2.1.3 System Architecture:

- Design and implementation of a scalable and modular system architecture to support high-volume data processing and storage.
- Utilization of cloud-based services for scalability, reliability, and accessibility across multiple devices and platforms.

2.2. License Plate Recognition Module Using YOLOv8

Functional Requirements:

The software needs to support three categories of functionalities as described below:

R.1 Data Interpolation:

R.1.1 INTERPOLATE DATA:

Description: The system will interpolate missing data entries for bounding boxes of cars and license plates to ensure consistency in the dataset.

Input: CSV file containing data entries with frame numbers, car IDs, and bounding box coordinates.

Output: Updated CSV file with interpolated data entries for missing frames, maintaining the structure of the original dataset.

R.2 License Plate Recognition:

R.2.1 DETECT LICENSE PLATE:

Description: The system will utilize a pre trained license plate recognition model to detect and recognize license plates from captured frames of a video stream.

Input: Frames from the video stream captured by the camera.

Output: Detected license plate numbers along with confidence scores for the recognition results.

R.3 Data Storage:

Description: The system will store the results of license plate detection and recognition, along with the interpolated data, for further analysis and processing.

Input: Detected license plate numbers, associated metadata, and interpolated data.

Output: Recorded data stored in a structured format (CSV file) for analysis, reporting, and future reference.

Non Functional Requirements:

1. Accuracy: The system must ensure accurate interpolation of missing data entries to maintain the integrity and reliability of the dataset.

2. Efficiency: The system should efficiently interpolate missing data entries to minimize processing time and computational resources required for data preprocessing.

3. Robustness: The system should be robust against variations in data quality and missing information to handle diverse datasets effectively.

4. Scalability: The system should scale seamlessly with the size of the dataset, accommodating large volumes of data while maintaining performance and accuracy.

5. Usability: The system should provide an intuitive interface for data interpolation operations, allowing users to easily manage and process datasets with minimal training or expertise.

6. Compliance: The system should comply with data privacy regulations and ensure the security and confidentiality of sensitive information during data interpolation and storage operations.

2.3 Vehicle Counting Module Using YOLOv8

Functional Requirements:

R.1 Object Detection:

R.1.1 DETECT OBJECTS:

Description: The system will utilize a YOLO object detection model to detect various objects in each frame of the input video.

Input: Frames from the input video stream.

Output: Detected bounding boxes, confidence scores, and class labels for objects such as cars, trucks, buses, and motorbikes.

R.2 Object Tracking:

R.2.1 TRACK OBJECTS:

Description: The system will implement a SORT (Simple Online and Realtime Tracking) algorithm to track detected objects across consecutive frames.

Input: Detected bounding boxes from object detection.

Output: Tracked objects with unique IDs assigned to each object for consistent tracking.

R.3 Counting Objects:

R.3.1 COUNT OBJECTS:

Description: The system will count the number of tracked objects crossing a specified region of interest (ROI) in the video frame.

Input: Tracked object positions and IDs.

Output: Total count of objects that have crossed the predefined ROI.

R.4 Visualization:

R.4.1 VISUALIZE OBJECTS:

Description: The system will visually represent the detected and tracked objects, along with the count of objects within the ROI, on the output video frame.

Input: Detected and tracked object information.

Output: Output video frames with overlaid bounding boxes, object IDs, and count of objects.

Non Functional Requirements:

1. Accuracy: The system must achieve high accuracy in object detection and tracking to ensure reliable counting of objects.

2. Performance: The system should maintain a real time processing speed to handle input video streams efficiently and produce output videos without significant delays.

3. Usability: The system should provide a user friendly interface for configuring parameters such as confidence thresholds, ROI coordinates, and output video settings.

4. Scalability: The system should be scalable to handle videos of varying resolutions and durations, accommodating different traffic scenarios and environments.

5. Robustness: The system should be robust against variations in lighting conditions, camera perspectives, and object occlusions to maintain accurate object detection and tracking performance.

6. Resource Utilization: The system should optimize resource utilization, including CPU, memory, and GPU usage, to ensure efficient processing and minimize hardware requirements.

2.4 Parking Space Occupancy Monitoring System Using OpenCV

Functional Requirements:

R.1 Parking Space Detection:

R.1.1 DETECT PARKING SPACES:

Description: The system will analyze each frame of the input video to detect parking spaces using image processing techniques.

Input: Frames from the input video stream.

Output: Visualization of parking spaces marked with rectangles on the video frame, indicating whether each space is occupied or free.

R.2 Parking Space Counting:

R.2.1 COUNT FREE PARKING SPACES:

Description: The system will count the number of free parking spaces based on the analysis of parking space occupancy.

Input: Occupancy status of each parking space detected.

Output: Total count of free parking spaces displayed on the video frame.

R.3 User Interaction:

R.3.1 INTERACTIVE MARKING:

Description: The system will allow users to interactively mark parking spaces on a reference image.

Input: Mouse clicks indicating parking space positions on the reference image.

Output: Updated list of marked parking space positions stored for further processing.

Non Functional Requirements:

1. Accuracy: The system must accurately detect parking spaces and determine their occupancy status to provide reliable information about free parking spaces.

2. Efficiency: The system should process video frames efficiently to minimize processing time and ensure real time or near real time performance.

3. Usability: The system should provide a user friendly interface for marking parking spaces interactively, allowing users to easily add or remove parking space markers as needed.

4. Robustness: The system should be robust against variations in lighting conditions, camera perspectives, and occlusions to maintain accurate parking space detection and counting.

5. Persistence: The system should persistently store the positions of marked parking spaces to enable seamless interaction across multiple sessions without losing data.

6. Security: The system should implement appropriate security measures to protect stored data, such as encryption and access control, to prevent unauthorized access or tampering.

7. Compatibility: The system should be compatible with different video formats and resolutions to support a wide range of input video sources.

3. DATASET DESCRIPTION

YOLOv8 :

Source: ultralytics

The pre-trained models used in our system , including YOLOv8 for vehicle detection and a license plate detector, were obtained from the official repositories of the respective frameworks.

YOLOv8 was pre-trained on the **COCO** dataset, consisting of diverse images across 80 object categories.
(categories: https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names)

Dataset Structure:

- **Total Image:** 143000 Images
- **Train Images**: Contains 118,000 images for training object detection, segmentation, and captioning models.
- **Validation Images**: Comprises 5,000 images utilized for validation during model training.
- **Test Images**: Consists of 20,000 images designated for testing and benchmarking trained models. Ground truth annotations for this subset are not publicly available.

License plate detector model :

The license plate detector model was trained on a dataset specifically curated for license plate recognition tasks, comprising a variety of license plate images from different regions and under various lighting conditions.

Source: Roboflow

Link: https://universe.roboflow.com/roboflow-universe-projects/license-plate-recognition-rxg4e

Dataset Structure:

- **Total Images:** 24242
- **Training Images:** 21174
- **Test Images:** 2048
- **Validation Images:** 1020

Dataset Sample :

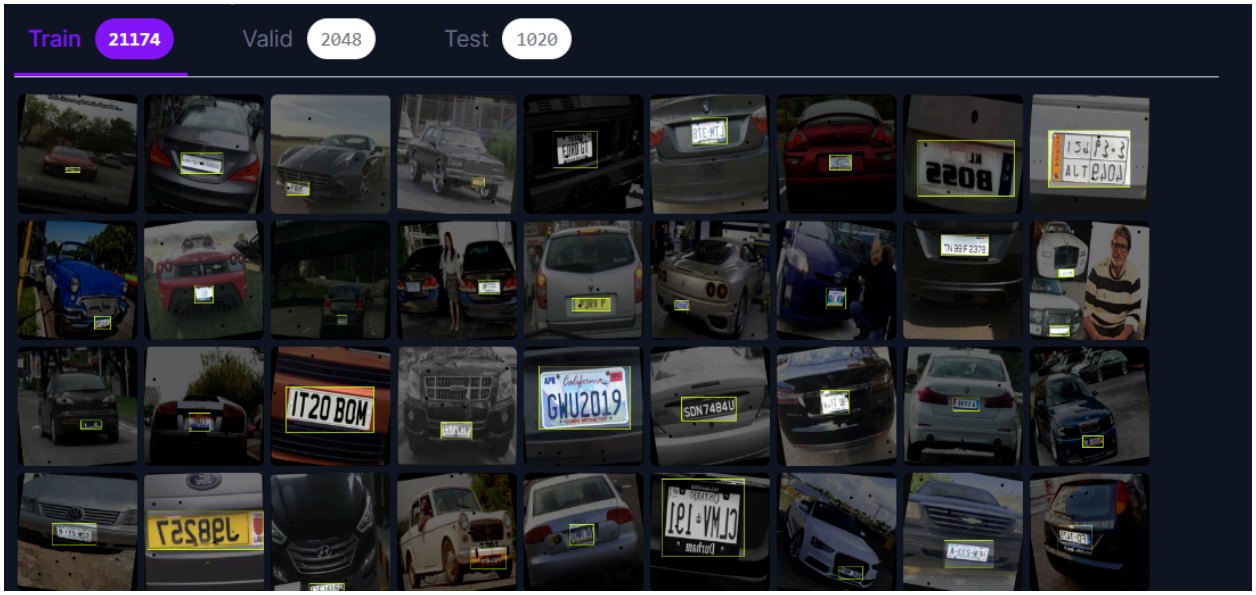


Fig. 3.1 Dataset of license plate detector

Target Variable :

Fig. 3.2 Target Variable of model

4. IMPLEMENTATION DETAILS

4.1 MODELS OF OUR SYSTEM:

We used 2 Pre-trained models for this system: Vehicle detection and License plate detection. For Vehicle detection we used YOLOv8 and for License plate detection we used a License plate detector.

4.1.1 Vehicle detection:

Introduction:

We detail the implementation process for vehicle detection using YOLOv8, integrated with the COCO (Common Objects in Context) dataset. YOLOv8 is renowned for its real-time object detection capabilities, while the COCO dataset provides a comprehensive resource for training and evaluating the model.

YOLOv8 Overview:

YOLOv8, the latest advancement in the YOLO series by Ultralytics, epitomizes continuous innovation in real-time object detection. It serves as a benchmark of state-of-the-art (SOTA) technology, enhancing performance, flexibility, and efficiency.

Key Features of YOLOv8:

- **Advanced Backbone and Neck Architectures**: YOLOv8 implements cutting-edge backbone and neck architectures, enhancing feature extraction and object detection capabilities.
- **Anchor-free Split Ultralytics Head**: Utilizes an anchor-free split Ultralytics head, improving accuracy and efficiency compared to traditional anchor-based approaches.
- **Optimized Accuracy Speed Tradeoff:** YOLOv8 maintains an optimal balance between accuracy and speed, making it suitable for real-time object detection tasks across diverse applications.
- **Variety of Pre-trained Models:** Offers a diverse range of pre-trained models tailored to different tasks and performance requirements, ensuring compatibility with specific use cases.

Supported Tasks and Modes:

The YOLOv8 series offers a diverse range of models, each specialized for specific tasks in computer vision. These models are designed to cater to various requirements, from object detection to more complex tasks like instance segmentation, pose/key points detection, oriented object detection, and classification. Additionally, these models are compatible with various operational modes including Inference, Validation, Training, and Export, facilitating their use in different stages of deployment and development.

COCO Dataset Overview:

The COCO dataset serves as a comprehensive resource for object detection, segmentation, and captioning tasks. It encourages research across a diverse range of object categories and is widely used for benchmarking computer vision models.

Key Features of COCO Dataset:

- **Image Count:** The dataset comprises 330,000 images, with annotations available for 200,000 images across various tasks.
- **Object Categories:** It covers 80 object categories, including common objects like cars, bicycles, and animals, as well as more specific items like umbrellas and handbags.
- **Annotations**: Annotations include bounding boxes for object detection, segmentation masks for instance segmentation, and captions for image understanding tasks.
- **Evaluation Metrics:** COCO provides standardized evaluation metrics such as mean Average Precision (mAP) for object detection and mean Average Recall (mAR) for segmentation, facilitating fair model performance comparison.

Applications:

The COCO dataset finds extensive usage in training and evaluating deep learning models across various computer vision tasks. It serves as a foundational dataset for object detection algorithms like YOLO, Faster R CNN, and SSD, as well as instance segmentation methods like Mask R CNN and keypoint detection models such as OpenPose.

4.1.2 License plate detection:

Base Dependencies:

- **ultralytics**: A library for deep learning and computer vision tasks.
- **hydra-core**: A framework for configuring complex applications, commonly used for managing configurations in deep learning experiments.
- **matplotlib**: A plotting library for creating visualizations in Python.
- **numpy**: A library for numerical computing in Python.
- **opencv-python**: A library for computer vision tasks, providing functionalities for image processing and manipulation.
- **Pillow**: A Python Imaging Library (PIL) fork for image processing tasks.
- **PyYAML**: A YAML parser and emitter for Python, commonly used for handling configuration files.
- **scipy**: A library for scientific computing and mathematical functions in Python.
- **torch**: The core PyTorch library for deep learning.
- **torchvision**: The PyTorch library containing datasets, models, and transformations for computer vision tasks.
- **tqdm**: A library for adding progress bars to Python code, commonly used for tracking the progress of tasks.

Logging:

- **tensorboard**: A library for logging and visualizing metrics during model training, commonly used in deep learning projects.

Plotting:

- **pandas**: A library for data manipulation and analysis in Python, providing data structures and functions for working with structured data.
- **seaborn**: A data visualization library based on matplotlib, providing a high-level interface for creating informative statistical graphics.

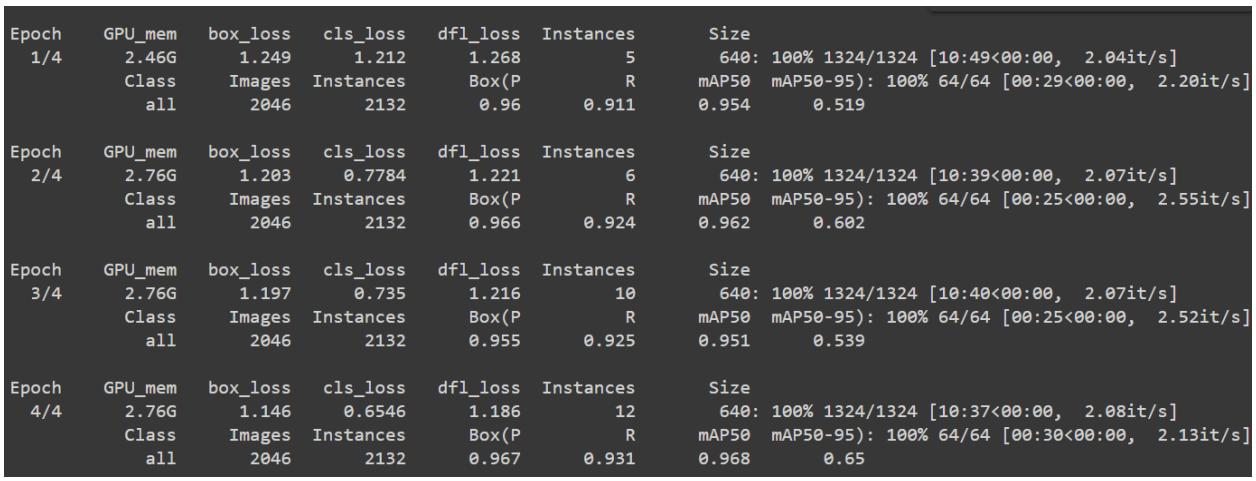
Extras:

- **ipython**: An interactive Python shell for interactive computing and data exploration.
- **psutil**: A library for retrieving information on running processes and system utilization.
- **thop**: THOP (Torch Operation) is a library for estimating FLOPs (floating-point operations) and number of parameters in PyTorch models.

HUB:

- **GitPython**: A library for interacting with Git repositories from Python code.

Epochs: Visualizing Deep Learning Model Training Progress



```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  1/4      2.46G      1.249      1.212      1.268          5        640: 100% 1324/1324 [10:49<00:00,  2.04it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 64/64 [00:29<00:00,  2.20it/s]
             all       2046       2132       0.96      0.911      0.954      0.519

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  2/4      2.76G      1.203     0.7784      1.221          6        640: 100% 1324/1324 [10:39<00:00,  2.07it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 64/64 [00:25<00:00,  2.55it/s]
             all       2046       2132      0.966      0.924      0.962      0.602

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  3/4      2.76G      1.197      0.735      1.216         10        640: 100% 1324/1324 [10:40<00:00,  2.07it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 64/64 [00:25<00:00,  2.52it/s]
             all       2046       2132      0.955      0.925      0.951      0.539

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  4/4      2.76G      1.146     0.6546      1.186         12        640: 100% 1324/1324 [10:37<00:00,  2.08it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 64/64 [00:30<00:00,  2.13it/s]
             all       2046       2132      0.967      0.931      0.968       0.65
```

Fig 4.1.2 Epochs of license plate detection model

This log tracks training progress over four epochs, detailing memory usage, loss values, instance counts, and mAP (mean Average Precision). Box and instance loss decline steadily, indicating improved bounding box prediction and instance segmentation. Meanwhile, mAP values consistently rise, reflecting enhanced object detection accuracy. These trends highlight the efficacy of the training strategy and model architecture, demonstrating continual performance enhancement.

We used the EasyOCR tool for license plate recognition in our system:

4.1.3 EasyOCR :

EasyOCR is actually a python package that holds PyTorch as a backend handler. It detects text from images but in my reference, while using it I found that it is the most straightforward way to detect text from images also when a high-end deep learning library(PyTorch) is supporting it in the backend which makes its accuracy more credible. EasyOCR supports 42+ languages for detection purposes. EasyOCR was created by a company named Jaided AI.

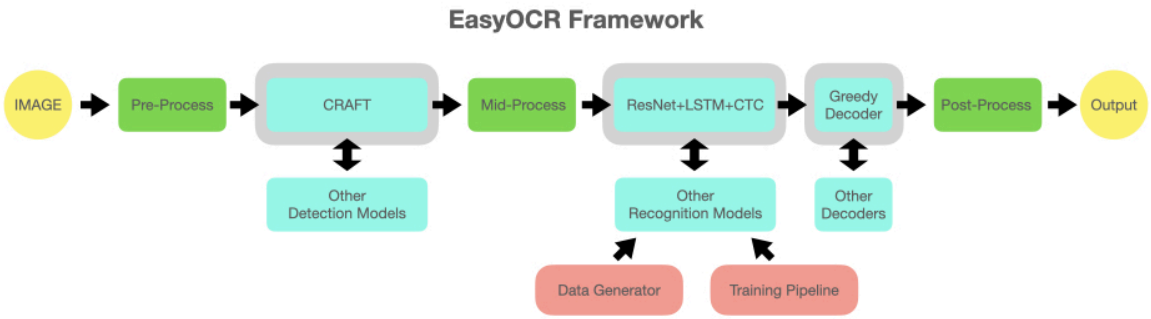


Fig 6.1.3 EasyOCR Framework

Key Features:

- **Pre-trained Models**: Utilizes pre-trained deep learning models.
- **Multi-Language Support**: Supports multiple languages for text extraction.
- **Flexibility**: Can be easily integrated into various applications.
- **Ease of Use**: Offers a straightforward API for OCR tasks.

Underlying Technology:

- **Deep Learning**: EasyOCR employs deep learning architectures, such as convolutional neural networks (CNNs), for character recognition.
- **Training Data**: The models used by EasyOCR are trained on large datasets of text images to ensure accuracy across different fonts, styles, and languages.

Usage:

- **Image Processing:** Extracts text from images in different formats.
- **Document Scanning**: Processes documents, including PDFs, for text extraction.
- **Integration**: Easily integrates into software applications or websites requiring OCR functionality.

Performance:

- **Accuracy**: The performance of EasyOCR is measured based on its accuracy in recognizing text from images or documents.
- **Speed**: It offers fast and efficient text extraction, suitable for real-time or batch-processing tasks.

Applications:

- **Document Digitization**: EasyOCR can be used for converting scanned documents into editable text formats.
- **Text Extraction**: It's useful for extracting text from images captured by cameras or smartphones.
- **Language Translation**: EasyOCR can aid in language translation tasks by extracting text for translation.

4.2 LIBRARIES USED IN PROJECT:

4.2.1 License Plate Detection :

main.py file of License plate recognition:

cv2 :

cv2 is a **Python library** widely used for **computer vision** tasks. It is an interface for **OpenCV (Open Source Computer Vision Library)**, which is a popular open source computer vision and machine learning software library.

With cv2, you can perform various image processing tasks such as reading and writing images, resizing, cropping, rotating, and transforming images. It also provides functionality for **object detection**, feature detection, image segmentation, and more. Additionally, cv2 allows for video processing, including video capture, manipulation, and analysis.

In our system, we utilized cv2 to perform specific image processing or computer vision tasks, such as:

- Loading and preprocessing images.
- Implementing algorithms for feature detection or object recognition.
- Performing image transformations or manipulations.
- Analyzing video streams or frames for real time applications.
- Extracting meaningful information from images or videos through segmentation or classification.

Here, cv2 is used for essential tasks such as loading, processing, and analyzing images and video frames in the context of object detection, tracking, and recognition within the file.

1. Loading and Processing Video Frames:

- cv2.VideoCapture() is used to open a video file ('./uploads/uploaded_video.mp4') for reading.
- cap.read() is utilized in a loop to read frames from the video file. Each frame is stored in the variable frame .

2. Image Manipulation and Processing :

- Image processing operations such as converting color spaces (cv2.cvtColor()), thresholding (cv2.threshold()), and cropping (frame[y1:y2, x1:x2, :]) are performed using cv2 functions.
- These operations are applied to manipulate and process the video frames for subsequent analysis.

3. Drawing Bounding Boxes :

- Bounding boxes are drawn around detected objects (vehicles and license plates) using cv2.rectangle() .
- These bounding boxes provide visual representations of the detected objects in the video frames.

4. Image Loading and Saving :

- Images are loaded using cv2.imread() (not shown in the provided code).
- Results, such as bounding boxes and text, are written to a CSV file ('./test.csv') using custom functions (write_csv()).

ultralytics :

Here, YOLO from **Ultralytics** is necessary to access the specific implementation of the **YOLO object detection model** provided by this library, ensuring compatibility, modularity, and potential access to additional functionalities or optimizations offered by Ultralytics.

1. Accessing YOLO Model : It utilizes the YOLO (You Only Look Once) object detection model for detecting objects (vehicles and license plates) in the video frames.

2. Modularization and Organization : Importing the YOLO model from Ultralytics modularizes the code and enhances its organization. Directly importing YOLO signifies the utilization of Ultralytics' YOLO model, simplifying comprehension and maintenance of the code.

3. Functionality Extension : Ultralytics provide additional functionalities, optimizations, or enhancements to the YOLO model that are not available in other implementations. By importing YOLO from Ultralytics, the code take advantage of these extended capabilities, potentially improving the performance or functionality of the object detection tasks.

SORT :

Here, **SORT**(Simple Online and Realtime Tracking) framework is used for **tracking multiple objects in real time video streams**. It provides a basic implementation of data association and state estimation techniques, making it suitable for online tracking applications. However, its performance heavily depends on the quality of the detections it receives.

Key points of SORT:

- **Purpose** : SORT is intended for online tracking applications where only past and current frames are accessible. It generates object identities dynamically as objects are tracked through the video.

- **Features** : SORT is a minimalistic tracker that focuses on simplicity. It doesn't handle complex scenarios like occlusion or re entering objects. Instead, it provides a basic framework that can serve as a foundation for the development of more advanced tracking algorithms.

- **Performance** : SORT was initially described in a paper and was ranked as the best open source multiple object tracker on the MOT (Multiple Object Tracking) benchmark at the time of its publication. However, it's important to note that a significant portion of SORT's accuracy depends on the quality of the detections it receives.

- **Dependencies** : SORT relies on object detections to perform its tracking tasks. For convenience, the repository containing SORT also includes Faster R CNN (Region based Convolutional Neural Network) detections for the MOT benchmark sequences in a standardized format.

util.py file of License plate recognition:

Below functions collectively facilitate the processing, validation, and extraction of license plate information from video frames, contributing to the overall functionality of the license plate recognition.

write_csv(results, output_path) :
Writes the tracking results to a CSV file.

- **Parameters**: results (dictionary) Contains the tracking results, output_path (string) Path to the output CSV file.

- **Purpose**: To store the tracking results in a structured format for analysis and further processing.

license_complies_format(text) :
Checks if the license plate text complies with a required format.

- **Parameter**: text (string) License plate text.
- **Purpose**: Validates the format of the license plate text to ensure consistency and accuracy.

format_license(text) :
Formats the license plate text by converting characters using mapping dictionaries.

- **Parameter**: text (string) License plate text.
- **Purpose**: Ensures uniformity in the representation of license plate text by converting characters according to predefined mappings.

read_license_plate(license_plate_crop) :
Reads the license plate text from a cropped image containing the license plate.

- **Parameter**: license_plate_crop (PIL.Image.Image) Cropped image containing the license plate.
- **Returns**: Tuple containing the formatted license plate text and its confidence score.
- **Purpose**: Extracts the license plate text from images for identification and analysis.

get_car(license_plate, vehicle_track_ids) :
Retrieves the vehicle coordinates and ID based on the license plate coordinates.

- **Parameters**: license_plate (tuple) Contains the coordinates of the license plate, vehicle_track_ids (list) List of vehicle track IDs and their corresponding coordinates.
- **Returns**: Tuple containing the vehicle coordinates (x1, y1, x2, y2) and ID.
- **Purpose**: Associates license plate information with vehicle coordinates for tracking and identification purposes.

4.2.2 Vehicle counting:

NumPy :

- **Description**: NumPy is a fundamental package for numerical computing with Python, providing support for large, multi dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

- **Usage:** NumPy is imported as np and used for various array manipulations and mathematical operations throughout the code, such as stacking arrays (np.vstack()), creating empty arrays (np.empty()), and resizing arrays (np.resize()).

Ultralytics :

- **Description**: Ultralytics is a computer vision library that provides implementations of deep learning models for object detection and tracking tasks. It offers pre-trained models and tools for training and deploying these models.

- **Usage:** Ultralytics' YOLO object detection model is imported and used for detecting objects in video frames. The model is initialized with pre-trained weights (yolov8l.pt) and used to detect various classes of objects in the input video frames.

OpenCV (cv2) :

- **Description**: OpenCV (Open Source Computer Vision Library) is a popular open source computer vision and machine learning software library. It provides various functions and algorithms for image and video processing, object detection, feature extraction, and more.

- **Usage:** OpenCV is imported as cv2 and utilized extensively for tasks such as reading and writing video files (cv2.VideoCapture() , cv2.VideoWriter()), image processing, drawing bounding boxes (cv2.rectangle()), overlaying images (cv2.bitwise_and()), and displaying images (cv2.imshow()).

cvzone :

- **Description**: cvzone is a Python library built on top of OpenCV, providing additional functionalities and utilities for computer vision applications. It offers various tools for image and video processing, object tracking, augmented reality, and more.

- **Usage**: cvzone is imported but not directly used in the provided code. It's possible that cvzone might be used in other parts of the project that are not included in the provided snippet.

Math :

- **Description**: The math module in Python provides a set of mathematical functions for performing mathematical operations. It includes functions for basic arithmetic, trigonometry, logarithms, exponentiation, and more.

- **Usage:** The math module is imported and used for performing mathematical calculations, particularly in rounding off confidence scores (math.ceil()).

SORT (github repo) :

- **Description**: SORT (Simple Online and Realtime Tracking) is a barebones implementation of a visual multiple object tracking framework based on rudimentary data association and state estimation techniques. It is designed for online tracking applications where only past and current frames are available and the method produces object identities on the fly.

- **Usage**: The SORT tracker is imported from a GitHub repository and used for tracking objects (e.g., vehicles) across video frames. It implements basic data association and state estimation techniques to assign object identities dynamically.

4.2.3 Parking Space Detection :

cv2 (OpenCV) :

- **Description** : OpenCV (Open Source Computer Vision Library) is a popular open source computer vision and machine learning software library. It provides various functions and algorithms for image and video processing, object detection, feature extraction, and more.

pickle :

- **Description**: The pickle module implements binary protocols for serializing and deserializing Python objects. It is used for saving and loading Python objects to and from files.

- **Usage**: In the code, pickle is used to save and load the list of parking space positions (posList) to and from a file named 'CarParkPos'. This allows the program to remember the positions of parking spaces between different runs.

cvzone :

- **Description** : cvzone is a Python library built on top of OpenCV, providing additional functionalities and utilities for computer vision applications. It offers various tools for image and video processing, object tracking, augmented reality, and more.

- **Usage**: In the code, cvzone is used for drawing text on images with a colored background. The cvzone.putTextRect function is used to draw the count of free parking spaces on the video frame with a colored background.

numpy :

- **Description** : NumPy is a fundamental package for numerical computing with Python, providing support for large, multi dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

- **Usage**: In the code, NumPy is used to handle arrays and perform mathematical operations. It is particularly used in the checkParkingSpace function to count the number of non-zero pixels in each parking space region.

4.3 ENVIRONMENT

- **Technology and tools used**
    - **Technology**
        - Python
        - OpenCV (cv2),
        - Ultralytics YOLO
        - Sort Algorithm
        - Easyocr
        - CVZone
        - React
        - Node and Express js
    - **Tools**
        - Git and GitHub
        - Visual Studio Code
        - MongoDB
        - Postman

5. EVALUATION

YOLOv8 :

Performance Metrics:

The performance of YOLOv8 models is evaluated using standardized metrics, including:

- **Size (pixels):** The model operates on images of size 50x50 to 95x95 pixels.

- **mAP (mean Average Precision) at IoU 50-95:** The model achieves a mean Average Precision (mAP) of 37.3% at IoU (Intersection over Union) threshold ranging from 50% to 95%. This metric indicates the accuracy of object detection.

- **Speed (CPU ONNX) - Inference Time:** The inference time for processing an image using the CPU ONNX implementation is 80.4 milliseconds. This metric reflects the speed of the model in making predictions on CPU hardware.

- **Speed (AIOO TensorRT) - Inference Time:** The inference time for processing an image using the AIOO TensorRT implementation is 0.99 milliseconds. This metric represents the speed of the model in making predictions on specialized hardware accelerated by TensorRT.

- **Parameters (M):** The model has 3.2 million parameters, which are the trainable weights learned during the training process.

- **FLOPs (B):** The model performs 8.7 billion Floating Point Operations (FLOPs) during inference, indicating the computational load required for processing each image.

License Plate Detection :

Performance evaluation of the license plate detection model involves quantifying its accuracy and robustness in detecting license plates across various scenarios.

Confusion Matrix:

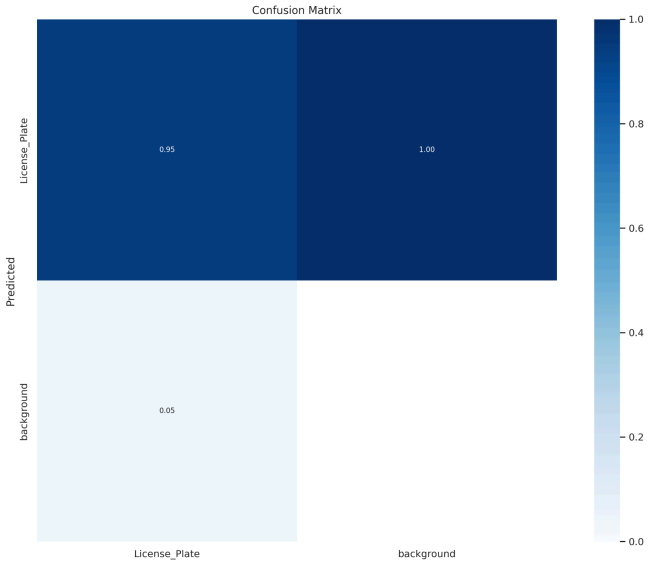


Fig 5.1 Confusion matrix

Precision-Confidence Curve :

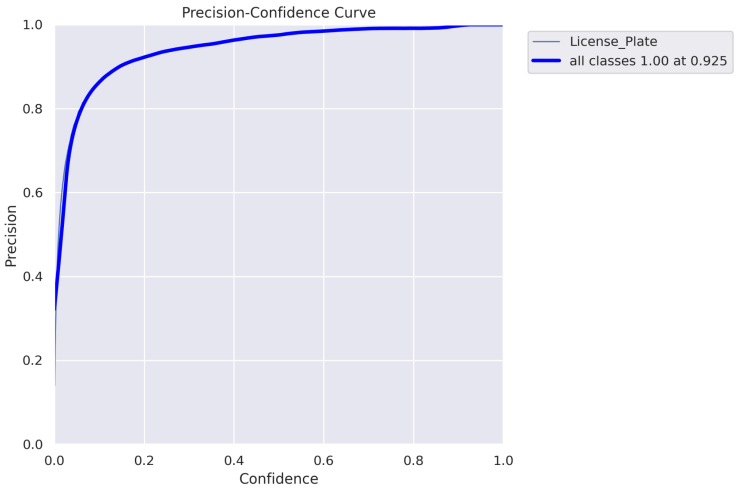


Fig 5.2 Precision-Confidence Curve

Precision-Recall Curve :

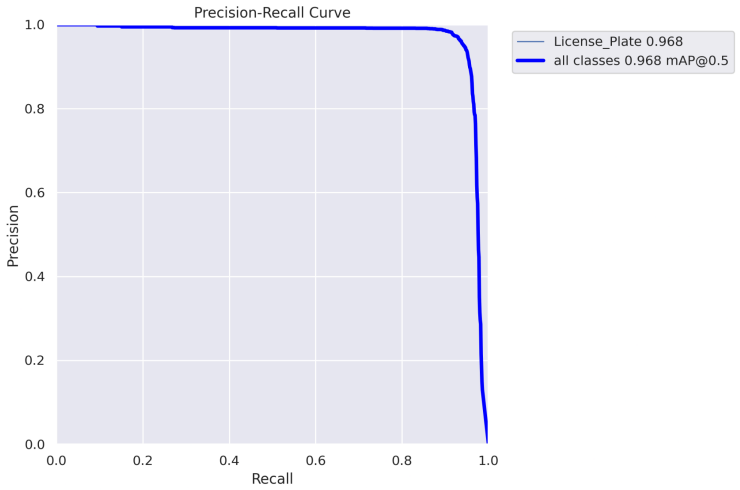


Fig 5.3 Precision-Recall Curve

F1-Confidence Curve :

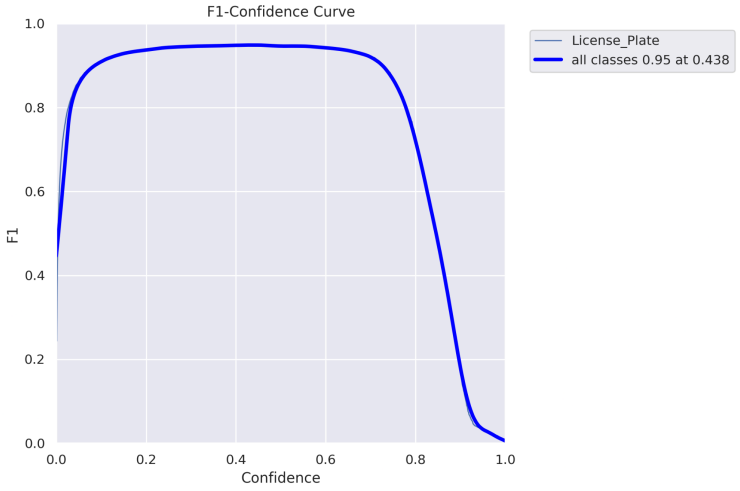


Fig 5.4 F1-Confidence Curve

6. TESTING

6.1 BLACKBOX TESTING

No.	Test Description	Test Case (Input)	Expected Results	Actual Result	Status
1.	Login	Correct username and correct password	Redirected to home page	Redirected to home page	success
		Incorrect Username and correct password	Invalid credentials message	Invalid credentials message	success
		Correct username and Incorrect password	Invalid credentials message	Invalid credentials message	success
2.	Sign Up	All details are correctly added	Redirected to login page	Redirected to login page	success
		Email is already registered	Email is already in use message	Email is already in use message	success
6.	Number Plate Detection	Upload a video	Give Processed video and generated CSV File	Give Processed video and generated CSV File	success
7.	Vehicle Counting	Upload a video	Give Processed video as an output	Give Processed video as an output	success
8.	Parking Space Detection	Upload a video	Give Processed video as an output	Give Processed video as an output	success

Fig 6.1 BlackBox Testing Table

6.2 SELENIUM TESTING

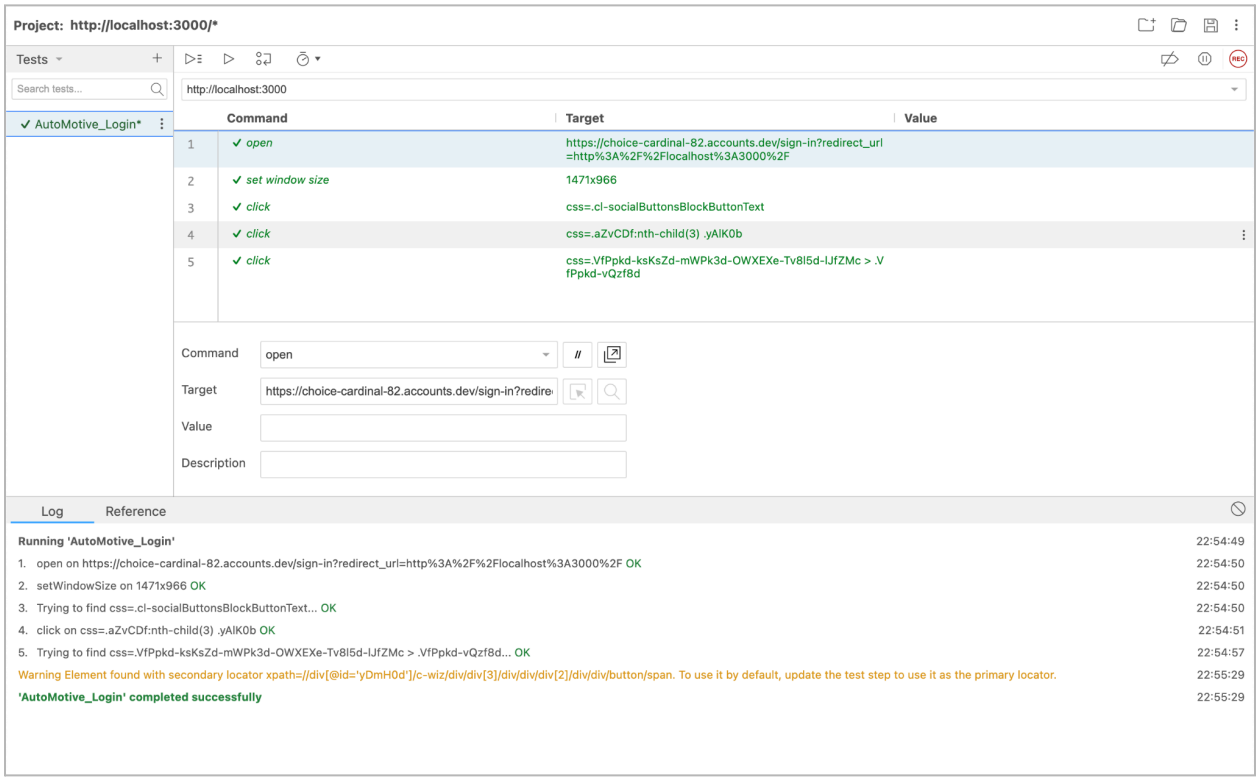


Fig 6.2.1 Selenium Testing of Login

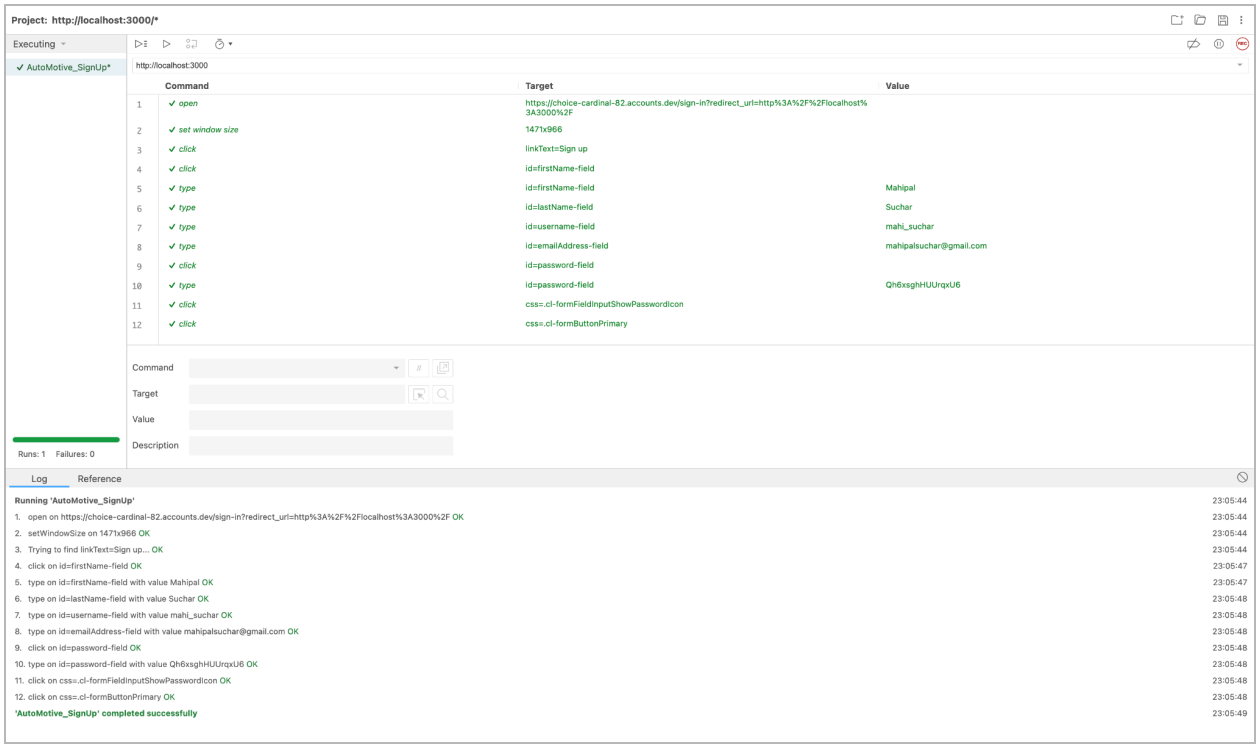


Fig 6.2.2 Selenium Testing of SignUp

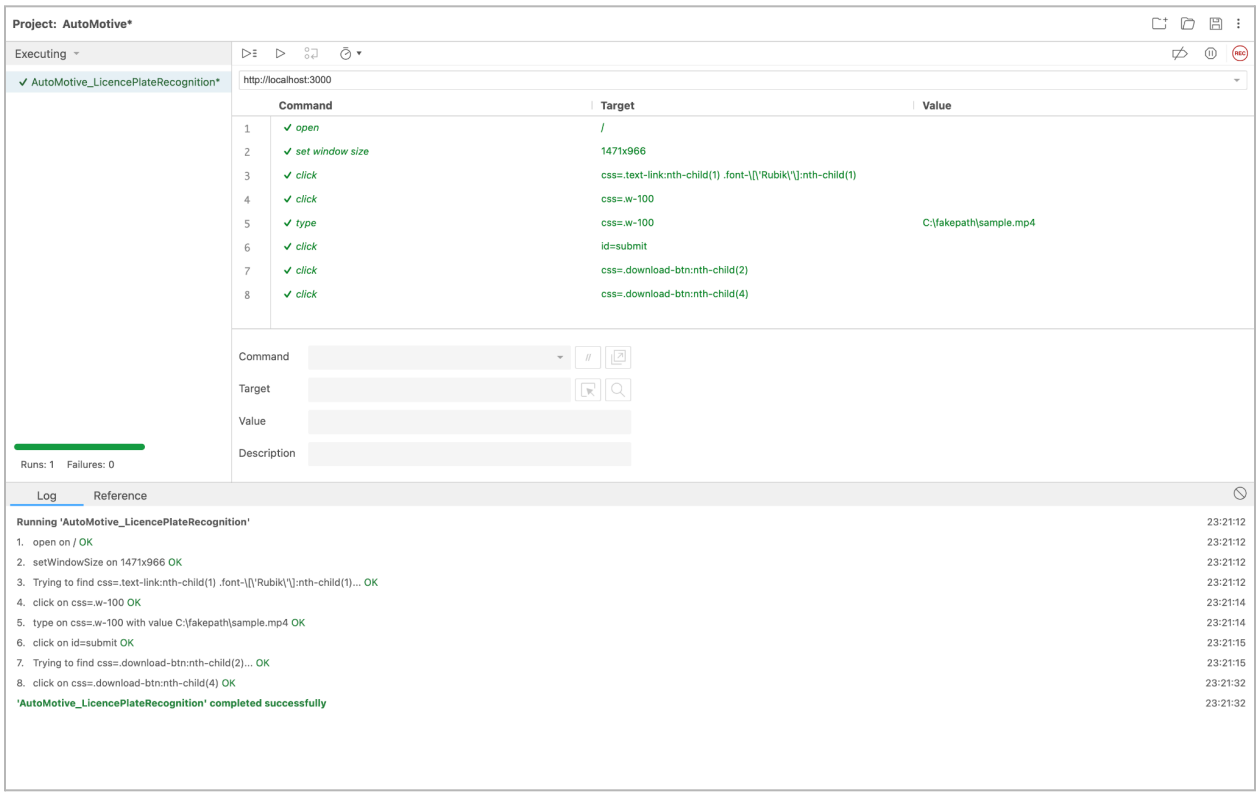


Fig 6.2.3 Selenium Testing of Licence Plate Recognition

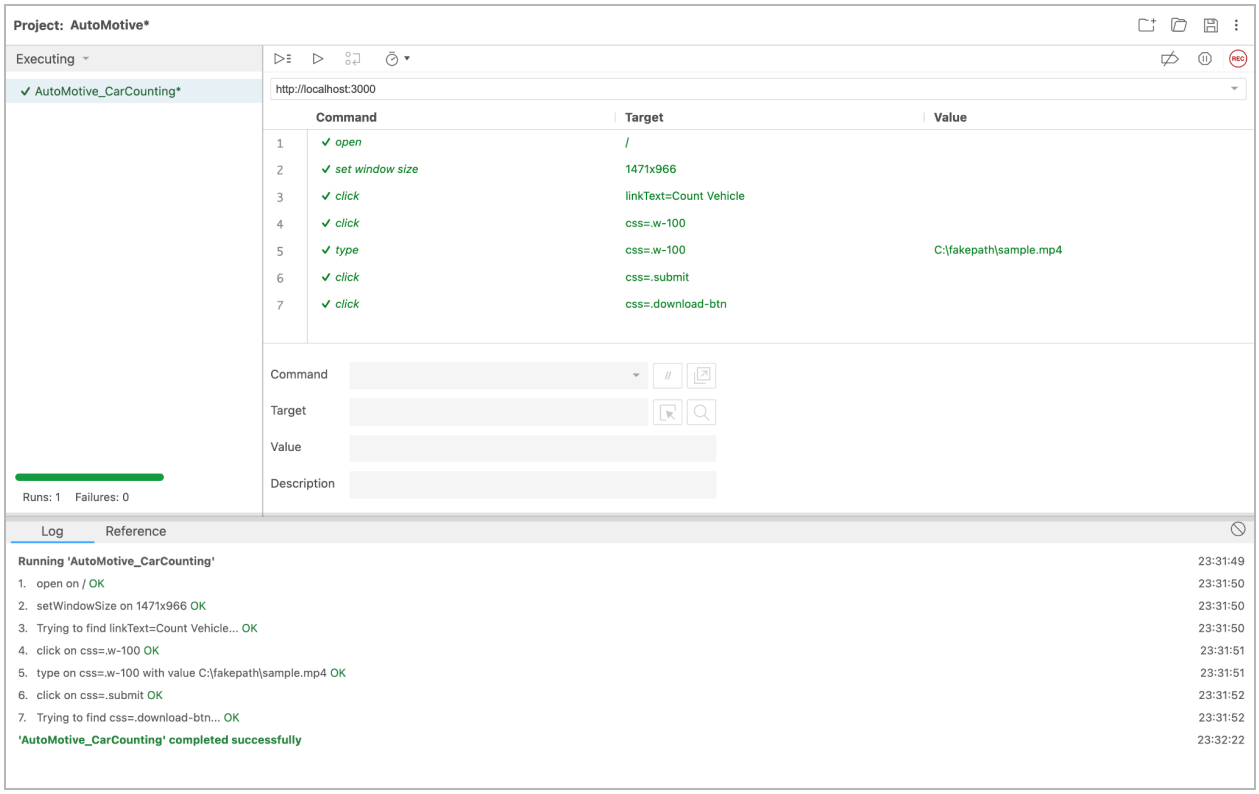


Fig 6.2.4 Selenium Testing of Car Counting

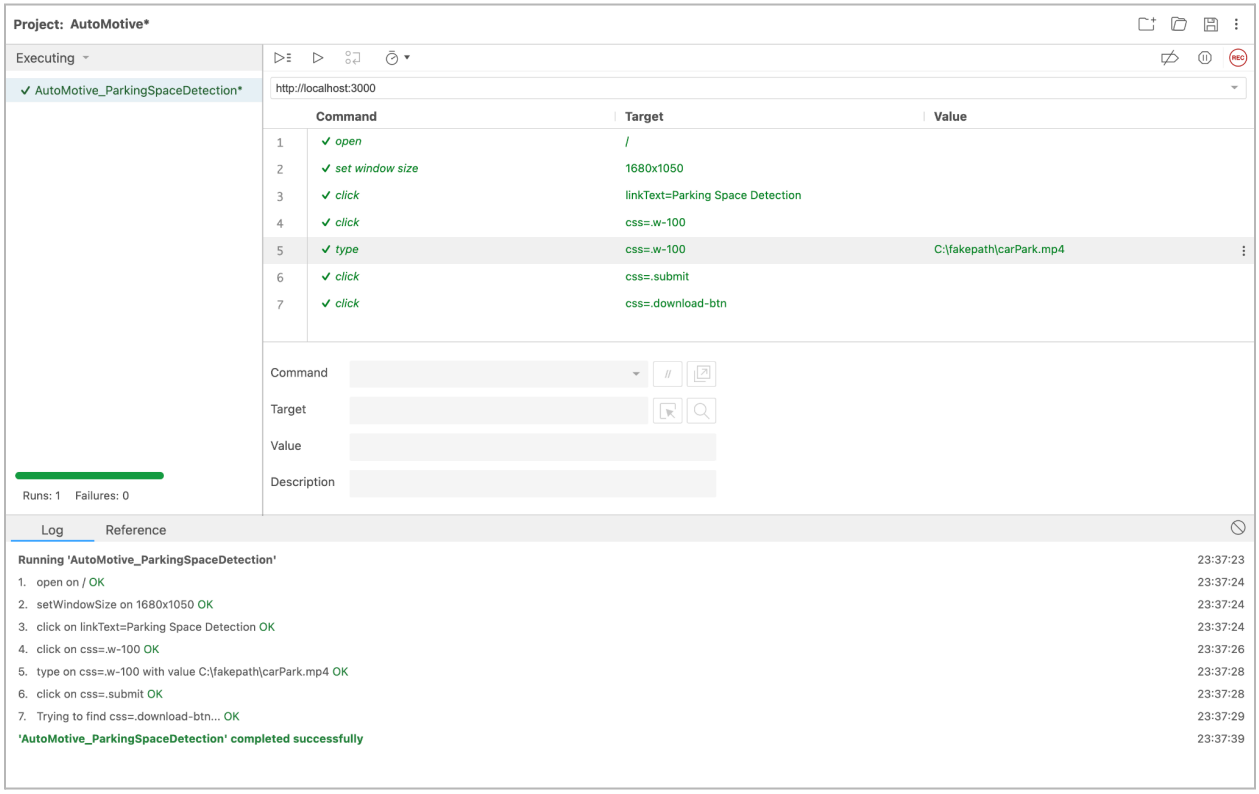


Fig 6.2.5 Selenium Testing of Parking Space Detection

7. LIMITATION AND FUTURE EXTENSION

Limitations:

- **Accuracy of Object Detection:** The accuracy of object detection using YOLO models affected by factors like lighting conditions, and presence of similar objects in the scene. This could lead to false positives or missed detections.

- **License Plate Recognition Accuracy:** The accuracy of license plate recognition impacted by factors such as variations in license plate sizes, fonts, and conditions like blur or distortion. This might lead to inaccuracies in reading license plate numbers, affecting the overall performance of the system.

- **Generalization to Different Environments:** The current implementation might be optimized for specific environments or datasets. Generalizing the system to operate effectively in diverse environments with varying lighting conditions, camera angles, and traffic densities could be challenging.

- **Excessive Processing Time:** Due to the complexity of the machine learning algorithms involved and the extensive computations required for video analysis, the time taken for processing and generating output videos is considerable, potentially hindering real-time application in certain scenarios.

Future Extensions:

- **Advanced License Plate Recognition Techniques:** Exploring advanced techniques for license plate recognition, such as deep learning-based approaches or ensemble methods, could improve the accuracy and robustness of license plate reading, especially in challenging conditions.

- **Advanced Integration for Smart Infrastructure:** Our system integrates data from multiple sensors for enhanced object detection, tracking, and scene understanding. With optimized real-time performance and scalability, it seamlessly deploys smart city infrastructures and transportation systems for efficient operations.

8. BIBLIOGRAPHY

- Yolov8: https://docs.ultralytics.com/
- Sort: https://github.com/abewley/sort
- License plate detection: https://universe.roboflow.com/
- Opencv: https://opencv.org/
- Stack Overflow: https://stackoverflow.com/
- CSS Loader (Spinner): https://10015.io/tools/css-loader-generator
- React JS : https://react.dev/