**Vehicle Cut-in Detection**

**INTEL UNNATI INDUSTRIAL TRAINING REPORT**

***submitted by***

***TEAM Spectrons***



|  |  |  |
| --- | --- | --- |
| **Kashif Khan** | **230051601069** | **B. S. Abdur Rahman Crescent institute of science and technology** |
| **S. Bhuvanesh** | **220051601023** | **B. S. Abdur Rahman Crescent institute of science and technology** |
| **Pranav Sumesh** | **220051601091** | **B. S. Abdur Rahman Crescent institute of science and technology** |

****

**DR. V Jean Shilpa Mentor Associate Proffessor**

**ACKNOWLEDGEMENT**

We would like to record our sincere thanks to Dr. C. Tharini Professor & Dean, School of Electrical & Communication Sciences for having given consent to carry out the Industrial Training from Intel Unnati Industrial Internship Programme.

We would like to record our sincere thanks Dr. G. Kannan HOD Electronics and Communication Engineering for motivating us to carry out the Industrial Training from Intel Unnati Industrial Internship Programme.

We wish to express our sincere appreciation and gratitude to Dr. V. Jean Shilpa

Associate Professor and Amartya Saikiya , Intel expert , for guiding us through the programme.

We wish to thank all the resource persons, who made this chance for us to undergo industrial training at Intel corp and we wish for more and more exposure like this.

|  |
| --- |
| **Mohammed Kashif Khan** |
| **S. Bhuvanesh** |
| **Pranav Sumesh** |

|  |  |  |
| --- | --- | --- |
|  | **TABLE OF CONTENTS** |  |
| **CHAPTER**  **NO** | **TITLE** | **PG NO** |
|  |  |  |
| **1** | **CHAPTER 1** | **4** |
| **2** | **CHAPTER 2** | 7 |
| **3** | **CHAPTER 3** | **13** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Abstract:**

This document describes a Python code that utilizes computer vision techniques to detect potential cut-in maneuvers of vehicles in a video. The code leverages the YOLOv8m object detection model from the ultralytics library to identify vehicles (cars, trucks, buses, motorbikes) within each video frame. A confidence threshold is set to ensure a minimum level of certainty in the detection.

The core functionality lies in identifying a cut-in event. The code differentiates between continuous vehicle presence and a new vehicle entering the scene. When a vehicle is absent for a specific duration (defined by a threshold and video frame rate), following a period of detection, it triggers a cut-in alert.

**CHAPTER 1**

**Introduction**

### **1.1.1 Vehicle Detection in Indian Terrain Using YOLOv8**

The rapid urbanization and growth of the automotive sector in India have led to an increase in the number and variety of vehicles on the roads. The diverse range of vehicles includes bicycles, motorcycles, auto-rickshaws, cars, buses, and heavy trucks, all sharing the same often congested and uneven roadways. This scenario presents unique challenges for vehicle detection and traffic management systems. Accurate detection and classification of vehicles are essential for effective traffic monitoring, congestion control, and ensuring road safety.

### **1.1.2 Challenges in Vehicle Detection**

In the context of Indian roads, the challenges in vehicle detection are multifaceted. The sheer variety of vehicle types, combined with the high density of traffic, makes it difficult to distinguish between different vehicles. Additionally, the dynamic nature of traffic, with frequent stops, starts, and sudden maneuvers, complicates the detection process. Environmental factors such as varying lighting conditions, weather changes, and road surface irregularities further add to the complexity. Traditional detection methods often fall short in these scenarios due to their inability to adapt to such diverse and dynamic conditions.

### **1.1.3 Advancements in Deep Learning and Object Detection**

Recent advancements in deep learning and object detection algorithms have significantly improved the accuracy and reliability of vehicle detection systems. One such state-of-the-art algorithm is the YOLO (You Only Look Once) model. YOLO is known for its real-time object detection capabilities, high accuracy, and efficiency. The latest iteration, YOLOv8, incorporates several enhancements that make it particularly suitable for complex detection tasks in challenging environments. YOLOv8 leverages advanced convolutional neural networks (CNNs) to process images and detect objects with remarkable speed and precision.

### **1.1.4 Implementing YOLOv8 for Vehicle Detection**

This report presents the implementation of the YOLOv8 model for vehicle detection in Indian terrain. The core of the implementation is a Python-based code that processes video frames or images to identify and categorize various types of vehicles. The YOLOv8 model is trained with a comprehensive dataset that includes different vehicle types commonly found in India. This training ensures that the model can accurately detect and separate each vehicle category, even in crowded and cluttered backgrounds.

**Ideal Methodology**

**1.2.1. Image Preprocessing Pipeline**

The provided code outlines an image processing pipeline built upon the OpenCV library. This section delves into the key components of this pipeline:

* **FindLaneLines Class:**
  + **Purpose:** This class serves as the central hub for handling image preprocessing, lane line detection (optional), and visualization.
  + **Attributes:**
    - calibration: An instance of the CameraCalibration class (not shown in the code) for undistortion using camera parameters (future integration).
    - thresholding: An instance of the Thresholding class (not shown) for segmenting objects from the background.
    - transform: An instance of the PerspectiveTransformation class (not shown) for warping the image to facilitate lane line detection.
    - lanelines: An instance of the LaneLines class (not shown) for identifying and potentially marking lane lines.
  + **Methods:**
    - \_init\_: Initializes the class and creates instances of the calibration, thresholding, transformation, and lane line objects.
    - forward(self, img): The core processing pipeline:
      * Creates a copy of the input image.
      * Performs undistortion using the calibration object (if camera calibration is integrated).
      * Applies perspective transformation with the transform object (if lane line detection is used).
      * Executes image thresholding with the thresholding object.
      * Processes the image for lane lines using the lanelines object (if implemented).
      * Warps the image back to the original perspective (if applicable).
      * Overlays the processed image onto the original image for visualization.
      * Visualizes the detected lane lines (if available).
      * Returns the final processed image.
    - process\_image(self, input\_path, output\_path): Processes a single image using the forward method and saves the result.
    - process\_video(self, input\_path, output\_path): Processes a video frame-by-frame using the forward method and saves the output video.

**1.2.3. Camera Calibration**

The presence of the CameraCalibration class hints at the potential for incorporating camera calibration in future work. Camera calibration is crucial for tasks like estimating object distances by mapping 3D points in the real world to their corresponding 2D pixel locations in the image.

**1.2.4. Time-to-Collision (TTC) Estimation (Potential Future Integration)**

This report doesn't include TTC estimation within the provided code. However, TTC can be calculated to assess collision risk with additional information, such as vehicle dimensions, relative velocity (potentially from sensors like LiDAR or radar), and the distance between the ego-vehicle and the cut-in vehicle (potentially estimated using camera calibration).

**1.2.5. IDD Dataset Integration (Potential Future Integration)**

The report mentions the IDD dataset, but the provided code doesn't utilize it directly. The IDD dataset offers a valuable resource for training an object detection model to identify vehicles in images or videos. This would be essential for robust cut-in detection functionality.

**CHAPTER 2**

**Software Description:**

**Operating System: Windows, macOS, or Linux**

**Programming Language: Python 3.12.3 or higher**

**Dependencies:**

* **OpenCV**
* **PyTorch**
* **YOLOv8 model files**
* **Other Python libraries (e.g., NumPy, Matplotlib)**

**2.1 OpenCV**

**OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. Developed by Intel and now maintained by a large community, OpenCV provides a comprehensive set of tools for image and video processing. Key features of OpenCV include:**

* **Image Processing: Functions for reading, writing, and manipulating images and videos.**
* **Object Detection: Algorithms for detecting and recognizing objects within images and videos.**
* **Machine Learning: Pre-trained models and tools for training custom models.**
* **Real-Time Processing: Efficiently handles real-time processing tasks, making it ideal for applications like video surveillance and autonomous vehicles.**
* **Cross-Platform Support: Available on various platforms, including Windows, Linux, macOS, Android, and iOS.**

**OpenCV is widely used in academic research and industrial applications due to its versatility and ease of use.**

**2.2 YOLO (You Only Look Once)**

**YOLO (You Only Look Once) is a state-of-the-art, real-time object detection system. Developed by Joseph Redmon and colleagues, YOLO is known for its speed and accuracy in detecting objects in images and videos. Key features of YOLO include:**

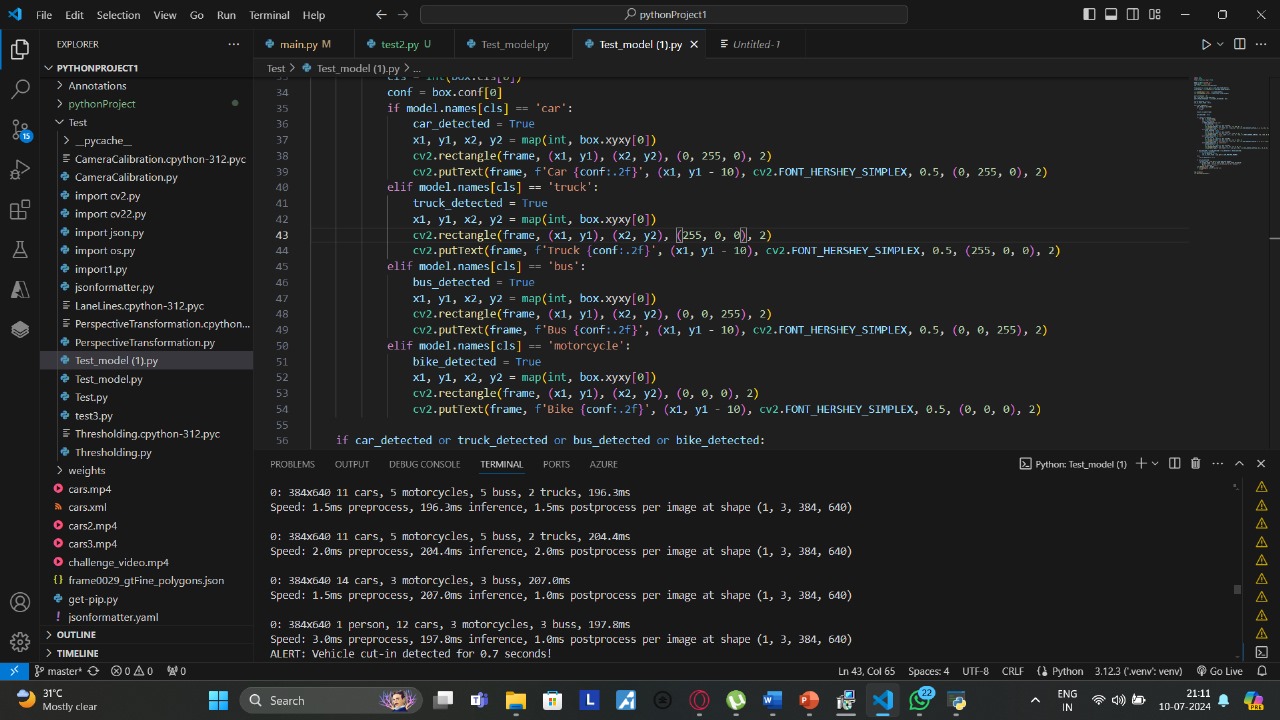
* **Real-Time Detection: Capable of processing images and videos at high frame rates, making it suitable for real-time applications.**
* **Unified Architecture: Uses a single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation.**
* **High Accuracy: Achieves high detection accuracy by leveraging advanced deep learning techniques and large-scale training datasets.**
* **Versatility: Can detect a wide range of objects, making it applicable to various fields such as autonomous driving, video surveillance, and robotics.**
* **Multiple Versions: Over time, YOLO has seen several iterations (YOLOv1, YOLOv2, YOLOv3, etc.), with each version improving on the previous one in terms of performance and accuracy. YOLOv8 is the latest iteration, incorporating further enhancements.**

**YOLO's ability to provide fast and accurate object detection has made it a popular choice for both research and practical applications in computer vision.**

**2.3 Code Explanation**

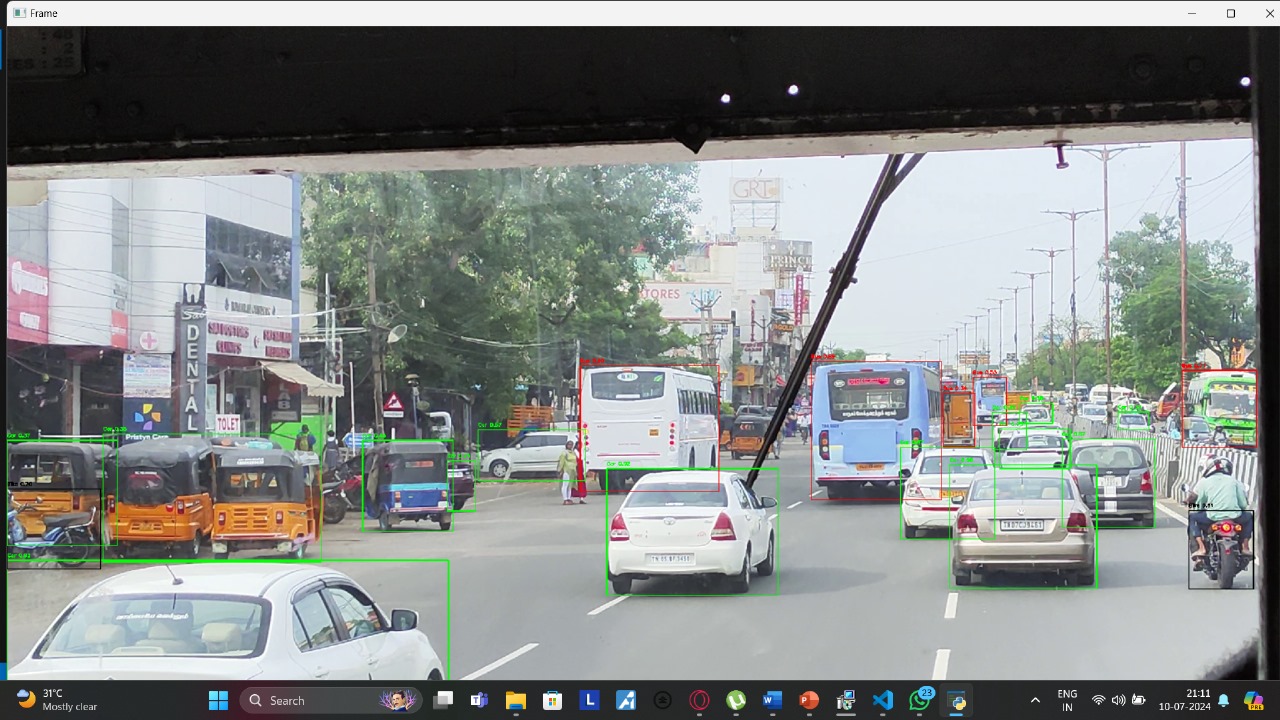
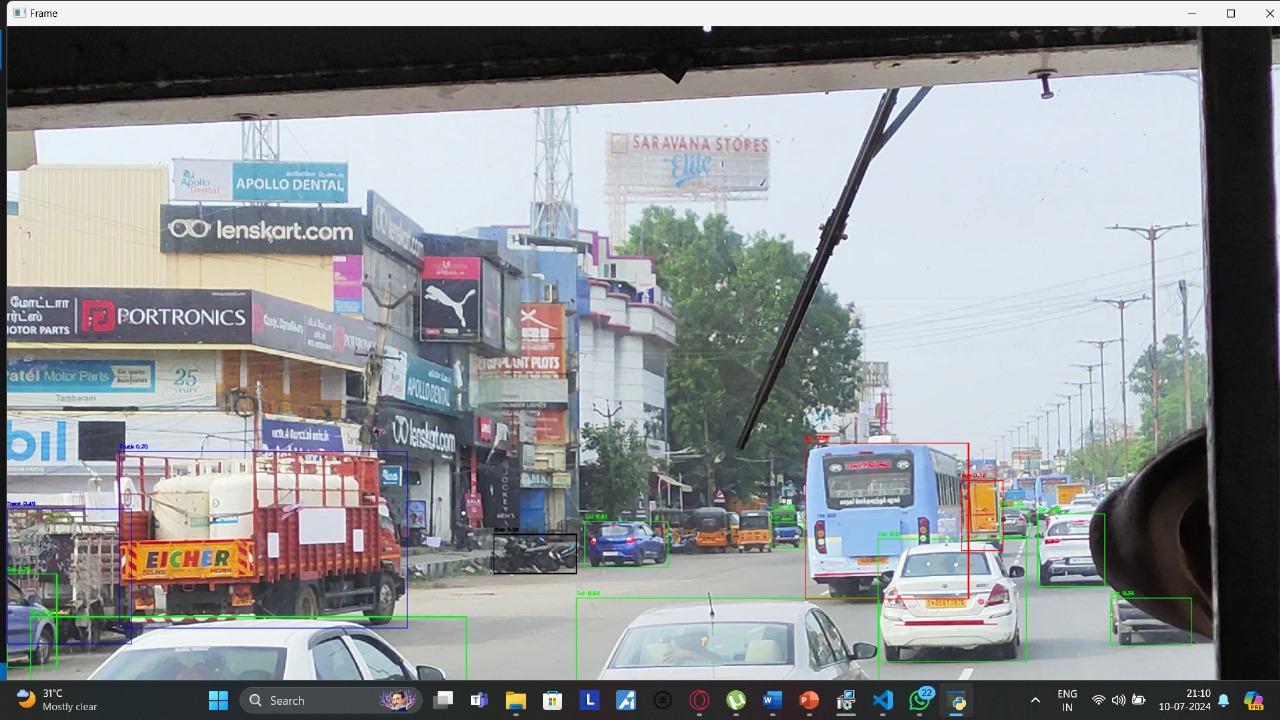
**Functionality:**

1. **Imports:**
   * cv2: OpenCV library for computer vision tasks.
   * torch: PyTorch framework for deep learning (used by ultralytics).
   * from ultralytics import YOLO: Imports the YOLO object detection model from ultralytics.
2. **Model Loading and Video Setup:**
   * model = YOLO('yolov8m.pt'): Loads the pre-trained YOLOv8m object detection model.
   * video\_path = 'testtt.mp4': Sets the path to the video file.
   * cap = cv2.VideoCapture(video\_path): Opens the video capture object.
   * frame\_width, frame\_height: Capture video frame dimensions.
   * cv2.namedWindow(...): Creates a window to display the video frames.
3. **Alert Threshold and FPS:**
   * alert\_threshold = 0.7: Confidence threshold for vehicle detection (0.7 = 70% confidence).
   * fps = cap.get(cv2.CAP\_PROP\_FPS): Gets the video's frames per second (FPS).
   * alert\_frame\_threshold = int(alert\_threshold \* fps): Calculates the number of frames corresponding to the alert threshold duration (0.7 seconds).
4. **Vehicle Detection Loop:**
   * The loop iterates through each frame in the video:
     + ret, frame = cap.read(): Reads the next frame.
     + if not ret: Checks for errors and exits if no frame is read.
     + results = model(frame): Detects objects in the frame using the YOLO model.
     + car\_detected = False (similar flags for other vehicles): Initialize detection flags.
5. **Iterating Through Detections:**
   * The loop iterates through each detected object:
     + for box in result.boxes: Loops through bounding boxes of detected objects.
     + cls = int(box.cls[0]): Gets the class index of the detected object.
     + conf = box.conf[0]: Gets the confidence score for the detection.
     + Conditional statements check the class index (model.names[cls]) and perform actions based on the detected vehicle type (car, truck, bus, motorbike):
       - If a vehicle is detected:
         * car\_detected = True (similar flags for other vehicles): Set corresponding flag.
         * Extracts bounding box coordinates and draws a rectangle around the vehicle on the frame.
         * Displays the vehicle type and confidence score on the frame.
6. **Cut-in Detection:**
   * if car\_detected or truck\_detected or bus\_detected or bike\_detected: Checks if any vehicle is detected in the current frame.
     + if not cut\_in\_detected: If a cut-in hasn't been detected yet:
       - cut\_in\_detected = True: Set cut-in detected flag.
       - cut\_in\_start\_frame = cap.get(cv2.CAP\_PROP\_POS\_FRAMES): Capture the current frame number as the cut-in start frame.
   * else: If no vehicle is detected:
     + cut\_in\_detected = False: Reset the cut-in detected flag.
7. **Alerting for Cut-in:**
   * if cut\_in\_detected: If a cut-in is ongoing:
     + current\_frame = cap.get(cv2.CAP\_PROP\_POS\_FRAMES): Get the current frame number.
     + if current\_frame - cut\_in\_start\_frame >= alert\_frame\_threshold: Check if the cut-in has lasted for the alert threshold duration.
       - print("ALERT: Vehicle cut-in detected for 0.7 seconds!"): Print an alert message.
       - cut\_in\_detected = False: Reset the cut-in detected flag.



1. **Displaying Results and Exit:**
   * cv2.imshow('Frame', frame): Display the processed frame with bounding boxes and labels.
   * cv2.waitKey(1): Wait for a key press for 1 millisecond.
   * if cv2.waitKey(1) & 0xFF == ord('q'): Check if the pressed key is 'q'

**Results:**

These footages were collected real time from sanatorium, Tambaram-Chennai:

**CHAPTER 3**

**3.1 Struggles:**

**3.1.1. Data bias and generalizability:**

If an ADAS trained mostly on highway driving data might struggle in unexpected situations like narrow city streets or construction zones.

**3.1.2.Edge AI vs. Cloud Processing:**

On-board, edge computing allows for faster reaction times but demands powerful hardware in vehicles. Cloud processing offers more power but introduces latency due to internet connectivity reliance.

**3.1.3.Long Development Cycles:**

Developing and validating complex AI models for ADAS is an iterative process. It can take months or even years to train, test, and refine a model before it can be deployed in a real-world setting.

**3.1.4.Maintaining Code Maintainability:**

ADAS systems are constantly evolving as new sensors, algorithms, and regulations emerge.

**3.1.5.Real-time Performance:**

ADAS code needs to run efficiently on embedded systems in vehicles with limited processing power.

**3.2 References and Technologies used:**

**3.2.1 References:**

**Train Yolov8 object detection on a custom dataset | Step by step guide | Computer vision tutorial**

[**https://youtu.be/m9fH9OWn8YM?feature=shared**](https://youtu.be/m9fH9OWn8YM?feature=shared)

**What is YOLO algorithm? | Deep Learning Tutorial 31 (Tensorflow, Keras & Python)**

[**https://youtu.be/ag3DLKsl2vk?feature=shared**](https://youtu.be/ag3DLKsl2vk?feature=shared)

**3.2.2 Technologies Used:**

YOLO (You Only Look Once) is one of the most popular modules for real-time object detection and image segmentation

OpenCV is a library of programming functions mainly for real-time computer vision.

Python is a high-level general-purpose programming language.