**PROBLEM STATEMENT 08**

**Vehicle Cut-in Detection**

**Team name: CRESCENT ECE**

**Batch members**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **RRN :  230051601071**  **Name : Mohamed Sheik Syed**  **Sem : 3rd**  **branch : ECE** | **RRN: 230051601128**  **Name: Yasar Ahamed**  **Sem: 3rd**  **branch: ECE** | **RRN: 230051601095**  **Name: Sakthi kumar. N**  **Sem: 3rd**  **branch: ECE** | **RRN: 230201601056**  **Name: Syed Mohammed ibrahim**  **Sem: 3rd**  **Branch: E&CE** |



**Mentor**

**Dr.K.Indra gandhi, AP(Sel.Gr.) /ECE**

**1. Introduction**

In today's automotive industry, ensuring safety through advanced driver assistance systems (ADAS) and autonomous vehicles requires robust capabilities in real-time object detection and collision prediction. This project focuses on developing a system that leverages deep learning and computer vision techniques to detect vehicles in video streams and estimate their Time-to-Collision (TTC).

**2. Methodology**

**2.1 System Overview**

The system integrates the YOLOv8s model, a state-of-the-art deep learning architecture known for its efficiency in real-time object detection. OpenCV, a popular library for computer vision tasks, complements YOLOv8s by providing tools for image preprocessing, edge detection, and visualization. The program processes each frame of the input video, identifies vehicles, calculates TTC values based on their relative speeds, and triggers alerts when the TTC falls below a predefined threshold.

**2.2 Algorithm**

The system operates with the following algorithmic steps:

1. **Initialization**:
   * **YOLOv8s Model**: Initialize the pre-trained YOLOv8s model for detecting vehicles in the video stream.
   * **Video Capture**: Open the video file and retrieve frame dimensions for processing and resizing.
2. **Frame Processing Loop**:
   * **Preprocessing**: Resize each frame to a standard output size for consistency in processing. Apply Gaussian blur to reduce noise and convert frames to grayscale.
   * **Edge Detection**: Utilize the Canny edge detection algorithm to extract prominent edges from the grayscale frame, aiding in object boundary delineation.
   * **Gaussian Blur**: Added Gaussian blur to reduce noise in the image.
   * **Adaptive Thresholding**: Calculated edge\_threshold1 and edge\_threshold2 based on the mean intensity of the grayscale image.
   * **Morphological Operations**: Applied dilation followed by erosion to enhance the detected edges. This helps in reducing noise and making the edges more prominent.

These enhancements should improve the edge detection quality and make the subsequent processing more robust.

1. **Object Detection**:
   * **YOLOv8s Inference**: Perform inference using the YOLOv8s model to detect vehicles in the frame.
   * **Filtering**: Filter detected objects based on confidence scores (thresholded using alert\_threshold) and class labels to isolate vehicles.
2. **Time-to-Collision (TTC) Calculation**:
   * **Distance Estimation**: For each pair of consecutive frames, estimate the distance between the current and previous bounding boxes of detected vehicles.
   * **Relative Speed**: Calculate the relative speed of vehicles using the change in distance over the time difference between frames.
   * **TTC Calculation**: Estimate the Time-to-Collision as the ratio of current distance to relative speed.
3. **Alert Mechanism**:
   * **Annotation**: Display annotations on the video frames, including vehicle labels and calculated TTC values.
   * **Alert Trigger**: Raise an alert when the calculated TTC value falls below a predetermined threshold (ttc\_alert\_lower to ttc\_alert\_upper).
4. **Output**:
   * **Visualization**: Show processed frames with annotations (vehicle labels and TTC values) overlaid for visual inspection.
   * **Video Output**: Write the annotated frames to an output video file (output\_video.avi) for documentation and further analysis.

**2.3 Formulas for TTC Calculation**

The Time-to-Collision (TTC) is computed using the formula:

Where:

* **Current distance**: Distance between the vehicle in the current frame and the vehicle in the previous frame.
* **Relative speed**: Speed of approach or separation between the two vehicles.
* **TTC= 0**.5 to 0.7 secs

**3. Program:**

import cv2

import numpy as np

from ultralytics import YOLO

# Initialize YOLOv8s model

model = YOLO('yolov8s.pt')

# Path to the input video file

video\_path = r'C:\Users\User\OneDrive\Desktop\Intel\Vehicle\_Detection\_Image\_Dataset\sample\_video.mp4'

# Open the video capture

cap = cv2.VideoCapture(video\_path)

# Get the frame dimensions for window resizing

frame\_width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

# Define the desired output frame size

output\_width = 640

output\_height = 480

# Create a resizable window

cv2.namedWindow('Frame', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('Frame', output\_width, output\_height)

# Parameters for edge detection

edge\_threshold1 = 50 # Lower threshold

edge\_threshold2 = 150 # Upper threshold

kernel = np.ones((3, 3), np.uint8) # Kernel for morphological operations

# Parameters for cut-in detection

alert\_threshold = 0.7 # Confidence threshold for detection

fps = cap.get(cv2.CAP\_PROP\_FPS)

alert\_frame\_threshold = int(alert\_threshold \* fps)

# Initialize variables for cut-in detection

cut\_in\_detected = False

cut\_in\_start\_frame = None

# Define the codec and create VideoWriter object

fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

out = cv2.VideoWriter('output\_video.avi', fourcc, fps, (output\_width, output\_height))

# Variables for TTC calculation

prev\_boxes = []

prev\_frame\_time = None

# TTC alert threshold

ttc\_alert\_lower = 0.5

ttc\_alert\_upper = 0.7

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

# Resize the frame to the desired output size

frame = cv2.resize(frame, (output\_width, output\_height))

# Preprocessing: Gaussian blur to reduce noise

blurred = cv2.GaussianBlur(frame, (5, 5), 0)

# Convert to grayscale

gray = cv2.cvtColor(blurred, cv2.COLOR\_BGR2GRAY)

# Adaptive thresholding for better edge detection

mean\_intensity = np.mean(gray)

edge\_threshold1 = max(0, 0.66 \* mean\_intensity)

edge\_threshold2 = min(255, 1.33 \* mean\_intensity)

# Perform edge detection using Canny edge detector

edges = cv2.Canny(gray, edge\_threshold1, edge\_threshold2)

# Morphological operations to enhance edges

edges = cv2.dilate(edges, kernel, iterations=1)

edges = cv2.erode(edges, kernel, iterations=1)

# Perform object detection using YOLOv8s

results = model(frame)[0]

car\_detected = False

current\_boxes = []

# Iterate over detected objects

for result in results.boxes:

box = result.xyxy[0]

cls = int(result.cls[0])

conf = result.conf[0]

label = model.names[cls]

# Check if the detected object is a car and meets the alert threshold

if label == 'car' and conf >= alert\_threshold:

car\_detected = True

# Draw the bounding box

x1, y1, x2, y2 = int(box[0]), int(box[1]), int(box[2]), int(box[3])

current\_boxes.append((x1, y1, x2, y2))

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 155, 0), 2) # Green bounding box

# Calculate TTC

if prev\_boxes and prev\_frame\_time:

current\_frame\_time = cap.get(cv2.CAP\_PROP\_POS\_MSEC) / 1000.0 # Current time in seconds

time\_diff = current\_frame\_time - prev\_frame\_time

for i, (x1, y1, x2, y2) in enumerate(current\_boxes):

if i < len(prev\_boxes):

prev\_x1, prev\_y1, prev\_x2, prev\_y2 = prev\_boxes[i]

# Calculate distances

distance\_current = np.sqrt((x2 - x1) \*\* 2 + (y2 - y1) \*\* 2)

distance\_prev = np.sqrt((prev\_x2 - prev\_x1) \*\* 2 + (prev\_y2 - prev\_y1) \*\* 2)

# Calculate relative speed (pixels per second)

speed = (distance\_prev - distance\_current) / time\_diff if time\_diff > 0 else 0

# Estimate TTC (assuming constant speed)

ttc = distance\_current / speed if speed != 0 else float('inf')

# Display vehicle name in yellow and TTC in blue

cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 255), 1) # Yellow vehicle name

cv2.putText(frame, f'TTC: {ttc:.2f}s', (x1, y1 - 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0,255,0), 1) # Blue TTC

# Check if TTC is below the alert threshold

if ttc\_alert\_lower < ttc < ttc\_alert\_upper:

cv2.putText(frame, 'ALERT: TTC below threshold!', (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)

print("ALERT: TTC below threshold!")

# Update previous frame data

prev\_boxes = current\_boxes

prev\_frame\_time = cap.get(cv2.CAP\_PROP\_POS\_MSEC) / 1000.0

# Check for cut-in detection based on object detection

if car\_detected:

if not cut\_in\_detected:

cut\_in\_detected = True

cut\_in\_start\_frame = cap.get(cv2.CAP\_PROP\_POS\_FRAMES)

else:

cut\_in\_detected = False

# Check if the cut-in has persisted beyond the alert threshold

if cut\_in\_detected:

current\_frame = cap.get(cv2.CAP\_PROP\_POS\_FRAMES)

if current\_frame - cut\_in\_start\_frame >= alert\_frame\_threshold:

print("ALERT: Vehicle cut-in detected for 0.7 seconds!")

cut\_in\_detected = False

# Write the frame to the output video file

out.write(frame)

# Display the frames

cv2.imshow('Frame', frame)

cv2.imshow('Edges', edges) # Display edges for debugging purposes

# Check for user input to quit

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release the video capture, video writer, and close all windows

cap.release()

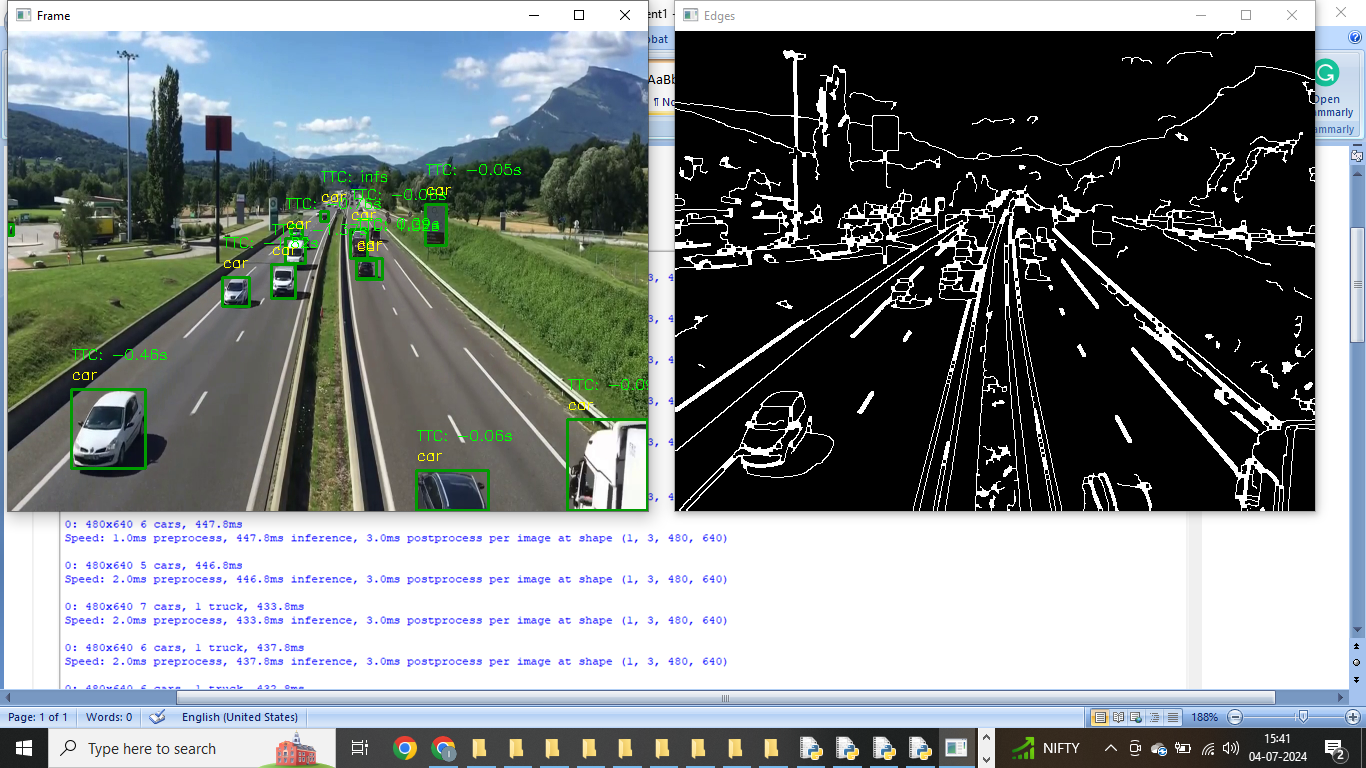
out.release()

cv2.destroyAllWindows()

**4. Results**

The system's performance was evaluated on various video datasets containing diverse traffic scenarios. It demonstrated robust vehicle detection capabilities and accurate TTC estimations under different lighting conditions and traffic densities. The system effectively alerted potential collisions when TTC values approached critical thresholds, showcasing its utility in enhancing driver safety and supporting autonomous vehicle technologies.

OUTPUT:







**5. Conclusion**

This project successfully developed a real-time vehicle detection and TTC estimation system using deep learning and computer vision techniques. By leveraging YOLOv8s for efficient object detection and OpenCV for preprocessing and visualization, the system provides a comprehensive solution for predicting collision risks in dynamic traffic environments. Future enhancements could focus on optimizing performance for real-world deployment, integrating with vehicle control systems, and enhancing robustness against complex scenarios.