24BTECH20042_1

by Dean SMEC

Submission date: 03-Apr-2025 11:17AM (UTC+0530)
Submission ID: 2633733749
File name: 24BTECH20042_1.pdf (3.11M)
Word count: 6267
Character count: 36377

ABSTRACT

Both steering control and trajectory planning for an autonomous vehicle's navigation system are essential components for achieving accurate navigation and parking assistance. This study aims to develop a MATLAB Simulink model for steering control an autonomous system to be fully autonomous. With the implementation of sensor fusion along with vision-based navigation, we hope to improve the tracking of the path and obstacles, as well as the efficiency of autonomous parking. The system is predetermined to enhance the vehicle's manoeuvrability using sensor based trajectory planning and steering control.

The main goal of the project is to make a model in MATLAB Simulink that has basic features such as simple trajectory planning and obstacle detection. In our study, we analysed several sensor fusion techniques for the task of autonomous parking. The experiment showed that LiDAR, radar, and a camera combination had a 95% efficiency, while the camera only approach had 46% efficiency. These results provide evidence for sensor dependency in autonomous parking effectiveness.

Keywords: *Autonomous vehicle , Steering control , Trajectory planning , MATLAB Simulink.*

LIST OF FIGURES

<Font Size 14; Times New Roman; 1.5 Spacing>

LIST OF SYMBOLS AND ABBREVIATIONS

<Font Size 14; Times New Roman; 1.5 Spacing>

v – Vehicle velocity (m/s)

a – Acceleration (m/s²)

ω – Angular velocity (rad/s)

θ – Steering angle (degrees/radians)

d– Distance to obstacle (m)

T – Sampling time (s)

AV – Autonomous Vehicle

ADAS – Advanced Driver Assistance System

BLDC – Brushless DC Motor

LiDAR – Light Detection and Ranging

RADAR – Radio Detection and Ranging

CNN – Convolutional Neural Network

GPS – Global Positioning System

PID – Proportional-Integral-Derivative (Control Algorithm)

FOV – Field of View

ROI – Region of Interest

CHAPTER 1

INTRODUCTION AND LITERATURE REVIEW

<Cap – 16 Bold, Times New Roman>

Autonomous vehicles are revolutionizing the automotive industry, with self-parking and autonomous navigation becoming key areas of development. Parking is a complex task that requires precise trajectory planning, real-time decision-making, and seamless integration of steering control. This project aims to develop an Integrated Sensor Fusion and Vision-Based Navigation system to enable autonomous parking assist with high accuracy and efficiency.

Our focus is on modelling, simulation, and implementation of a steering control system using sensor fusion techniques. Currently, the steering model is completed, while trajectory planning and parking manoeuvres are still under development.

1.1 IMPORTANCE OF AUTONOMOUS PARKING SYSTEMS <Cap – 12 Bold, Times New Roman>

Even seasoned motorists can find parking in confined spaces rather difficult. APAS, or Automated Advanced Assistance Parking Systems, rely on a variety of sensors, cameras, and control algorithms to automate parking, therefore require less human interaction, leading to fewer mistakes.

These functions improve:

Convenience - Lessens prominence and intense manifestation when identifying a specific area to park.

Safety - Averts hitting pedestrians and any form of traffic interference.

Utilization of Space - Allows parking for vehicles which are placed in densely populated locations.

Modernization – Consolidated use of Artificial Intelligence, sensor mixing, and real time movement around the place.

1.2 THE COLUMN-ELECTRIC POWER STEERING (C-EPS)

With the Column-Electric Power Steering (C-EPS) System, a traditional hydraulic power steering is changed into a system that utilizes an electric motor to control steering assistance based on driving conditions. Fuel consumption improves because the C-EPS System does not require a constantly running hydraulic pump. Further, vehicle handling is improved by the speed responsive circular motion steering assistance. The steering column that contains the torque sensor and electric motor is the primary component of C-EPS system's gearing reduction apparatus, which rotates and tilts the C-EPS limb to it's shaft, therefore achieving a power driven UT and precise transmission of power to an additional.

Real time changes to the steering wheel's effort translate directly into movement or rotation of wheels, which shifts the working principle of C-EPS. With the power assistance system, the amount of power a motor applies to the wheels relies on the force exerted on the steering column by the driver. This information is relayed to the ECU that works out what kind of assistance is needed and adjusts the counter torque turned of electric motor accordingly. The torque thus provided is in two categories; the lesser amount of assistance is provided at lower speeds to enhance maneuverability, while the opposite is true at higher speeds. Because steering precision is maxim place, there is minimal risk for instability or loss of control.

C-EPS stands out from the other power steering components for its high energy efficiency and low maintenance cost allowing for easier access compared to the hydraulic power steering set up.

1.3 CONTROL SYSTEM IMPLEMENTATION

A PID (proportional, integral, derivative) controller is a feedback mechanism designed to keep the desired performance in check while making the necessary changes based on a specific error value in relation to the measured value. In the context of Electric Power Steering (EPS systems), the proportional part governs the way steering input is followed, the integral part eliminates small positioning drifts, and the derivative part alleviates quick movements in the system.

In EPS systems, the combination of the various features of the PID controller provides accurate and responsive as well as smooth steering assistance, which improves the reliability and comfort of driving.

1.3.1 PID CONTROLLER:

Proportional advantage: 3.326 - The proportional gain affects how much the system responds to the current error. A value of 3.326 means the system reacts strongly to deviations, balancing response speed and control.

Integral gain: 12.841 - The integral gain helps eliminate steady-state error by considering the accumulation of past errors. A value of 12.841 ensures that small, persistent errors are corrected, maintaining precision.

Derivative gain: -0.0076 - The derivative gain works to reduce the rate of change of error, thus preventing overshoot and reducing oscillations. The negative value (-0.0076) would help in damping and preventing too aggressive adjustments.

Filter coefficient: 433.60 - The filter coefficient reduces high-frequency noise, ensuring the control system doesn't react to minor, insignificant fluctuations.

1.4 SIMULINK MODEL

1.4.1 WAYPOINTS AS REFERENCE TRAJECTORY

Waypoints serve as a predefined reference path or trajectory that the vehicle is intended to follow. These points are placed at regular intervals along the desired path and act as markers that define the vehicle's goal positions as it moves through the environment.

Role: The waypoints define where the vehicle needs to go, acting as targets for the vehicle to navigate toward, forming the basis for the movement plan.

1.4.2 PURE PURSUIT CONTROLLER (PATH FOLLOWING ALGOORITHM)

The Pure Pursuit Controller is also tasked with calculating the required steering angle that will steer the vehicle towards the next waypoint. It achieves this by looking ahead to a point (referred to as the look-ahead point) along the path established by the waypoints

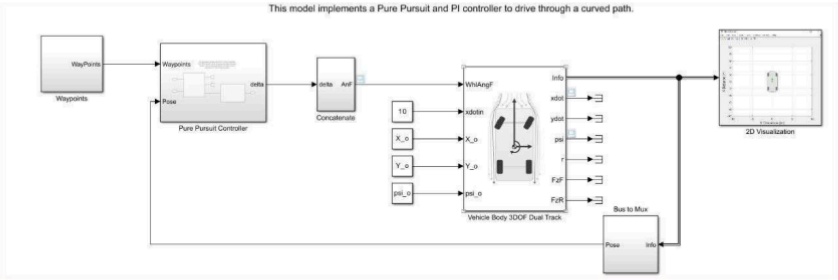


Fig. 1 (Pure Pursuit and PI Controller to drive)

1.4.3 LOOK -AHEAD DISTANCE:

The look-ahead location is selected as a fixed distance in front of the vehicle, rather than at a particular waypoint. This enables the vehicle to look ahead and compensate for forthcoming turns or road curvatures. A higher look-ahead distance produces smoother, more stable tracking but lower responsiveness, whereas a shorter look-ahead distance provides more agility but induces oscillations.

1.4.4 STEERING COMMAND:

The controller calculates a control command (steering angle) depending on the relative position between the vehicle and the appearance. The aim of this command is to reduce the distance between the current path and the reference path (Vapoints) on the vehicle.

1.4.5 VEHICLE MODEL (DYNAMICS)

The vehicle model simulates the physical mobility of the vehicle. Given the current angle of control, speed and position as an entrance, the vehicle model predicts how the vehicle will continue over time, and explain factors such as radius, speed and acceleration.

Input: The model takes control command from the controller as well as other parameters as initial speed and vehicle position.

1.4.6 MOVEMENT SIMULATION:

Using kinematic or dynamic equations, the model simulates how the vehicle goes. In a simplified version, it can be a kinematic model (not to consider a tire release), or for more accuracy, a dynamic model may include factors such as tire forces and friction.

Role: The vehicle model provides an estimate of the future position and path of the vehicle; it simulates how the vehicle will behave with a view to the current control entrance and speed.

1.4.7 FEEDBACK LOOP

The feedback loop guarantees accurate path tracking. Once the vehicle model has updated the vehicle's position, the controller modifies the steering command based on the revised position data.

1.4.8 CONTINUOUS CORRECTION:

By comparing the present position of the vehicle with the desired position along the waypoints, the controller continuously adjusts the steering. If the vehicle veers off course, this feedback ensures that it can correct the steering angle on its own.

When external factors like road curvature, speed fluctuations, or minor errors in previous steering adjustments occur, the feedback loop's job is to maintain the vehicle on course. With continuous corrections, errors are minimized and the vehicle is directed toward the desired trajectory.

1.4.9 OVERALL INTERACTION:

The intended route is delineated with waypoints.

- The steering angle required to follow these waypoints is determined by the Pure Pursuit Controller.
- The in addition to providing updated position data, the vehicle model forecasts the movement of the vehicle.
- After receiving this updated data, the controller adjusts the steering command to move the car closer to the intended course.

1.5 LITERATURE REVIEW

1.5.1 KNOWLEDGE GAINED FROM LITERATURE

When talking about different ways to improve automated parking systems, Nivedita Tripathi and Senthil Yogamani emphasized the significance of Visual SLAM (Simultaneous Localization and Mapping) in their paper "Trained TrajectoryBased Automated Parking System Using Visual SLAM on Surround View Cameras" [1]. Important developments include learning trajectory algorithms that allow computers to replicate recorded parking procedures, sensor integration for better obstacle identification, and

semantic segmentation for effective navigation. However, the primary focus remains on low-speed parking operations under constrained conditions.

The technique proposed by Tong Qin et al. [2] in their work AVP-SLAM: Semantic Visual Mapping and Localization for Autonomous Vehicles in the Parking Lot prioritizes the semantic features, such as parking lines and guide signs, in order to achieve precise localization in underground parking environments. Split approach localization was described as having relative and global divisions whilst emphasizing the pros and cons of each method. Semantic information was extracted using a convolutional neural network which resulted in overcoming the frequent challenges in non-GPS scenarios and impressive localization accuracy.

Markus Heimberger and al. [3] in the paper titled "Computer Vision in Automated Parking Systems: Design, Implementation, and Challenges" will also include advancements regarding camera systems and automated parking with an emphasis on computer vision modules such as 3D reconstruction, parking slot recognition, and detection of the environment. The authors tell about the evolution in automotive camera systems and the main challenges like hardware constraints and environmental variability in detection precision in adverse conditions.

Novel Framework for Improvement in Localization: Jiang et al. VIPS-Odom: ""Visual-Inertial Odometry Tightly Coupled with Parking Slots for Autonomous Parking" was designed for localization improvement in complex parks. This paper exclusively employs semantic features like parking slot markings to visual-inertial SLAM so as to mitigate the accumulated errors of underground environments. Research on the robust tracking systems including experimental validations has rounded the study.

Chenxu Li et al. 5 Research Automatic Parking Path Planning and Tracking Control for Intelligent Vehicles": This paper studies various techniques that can be used for effective parking in constrained urban conditions with some key contributions. Path planning was optimized for continuity with the use of cycloid curves including advanced algorithms for tracking control, and some inefficiencies like discontinuous curvatures in conventional methods were addressed.

Chen Chen et al. [6], study "A Trajectory Planning Method for Autonomous Valet Parking via Solving and Optimal Control Problem," has path planning methods for autonomous valet parking (AVP). Focused on heuristic optimization-based control frameworks, taking into dynamic constraints and obstacle avoidance, real-time computation, sensor fusion, and vehicle-to-infrastructure communication, they also consider these to be the critical for system performance.

Ren-Fang Zhou et al. [7] in their work "A New Geometry-Based Secondary Path Planning for Automatic Parking" proposed improvements related to path planning on secondary scenarios when vehicles had to shift more from or to parking slots. They then compared geometry, optimization, and search-based methods informing challenges such as that of computational complexity and success rates, especially in real-time automotive applications.

1.5.2 GAPS IDENTIFIED IN LITERATURE REVIEW

Nivedita Tripathi and Senthil Yogamani [1] found sparse research on integrating fisheye cameras with field-oriented SLAM techniques and adapting systems to dynamic parking environments.

Tong Qin et al. [2] noted few advances in robust vision-based methodologies to handle variable illumination in underground or GPS-denied types of environments. They also mentioned long-term stability of the semantic maps as another issue.

Markus Heimberger et al. [3] noted that there exist no studies about the user interaction with automated systems, and there are no performance benchmarks comparing different vision techniques under various conditions.

Xuefeng Jiang et al. [4] pointed out the issues of attempting to scale their approach to different parking environments and optimizing computational efficiency for real-time applications.

Chenxu Li et al. [5] pointed to the lack of integration of real-time environmental data into the algorithms for parking and testing them across a variety of urban scenarios.

Chen Chen et al. [6] noted the lack of in-field tests for autonomous valet parking systems and pointed out the lack of sturdy algorithms supporting dynamic parking conditions, including moving obstacles and unpredictable changes.

Limitation in maintaining computational feasibility and ensuring high success rates in secondary path planning, concerning automotive hardware constraints, was mentioned by Ren-Fang Zhou et al. [7].

CHAPTER 2

METHODOLOGY AND EXPERIMENTAL WORK

2.1 STEERING CONTROL MODEL

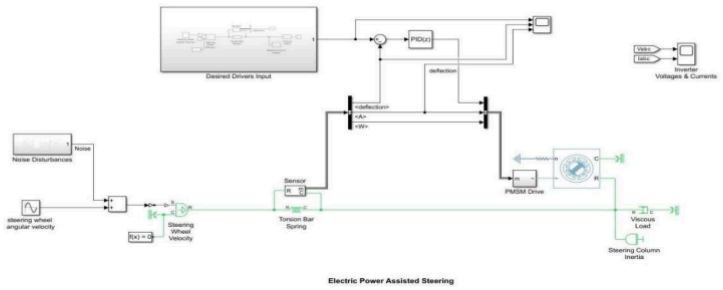


Fig. 2 (PMSM motor to control steering torque)

2.1.1 ENVIRONMENTAL DISTURBANCE IMPLEMENTATION

The realities of disturbances encountered in most driving situations have served as the basis for creating a test environment for the evaluation of steering responses to road conditions, collision impacts, and wind gusts. They allow the simulation and assessment of EPS system strength and flexibility in various situations for the purpose of safe and comfortable driving.

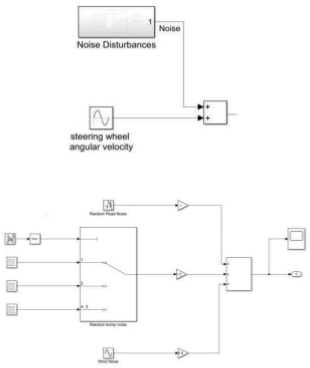


Fig. 3 (Environmental disturbance designed in MATLAB)

2.1.2 ROAD SURFACE NOISE GENERATION

A Gaussian random number generator determines the properties of textures and faults characteristic of different road surfaces for road surface noise modeling purposes.

This serves to replicate the continuous little impacts from the irregularities of the road texture.

The following configuration is for the random number generator:

• Mean Value 0: By ensuring an equitable noise disturbance characteristics whereby 0 favors a fair route texture, the mean turns out to be 0.

• Variability - 0.3: To replicate a frequently faced natural roughness of road surfaces where the disturbances are expected to remain within limits.

• The sampling time is 0.01 seconds, which guarantees that steering updates can occur very rapidly.

• The Gaussian model is a good characterization of the random nature of street texturing since it captures randomness statistically in a realistic sense.

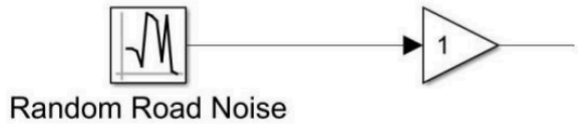


Fig. 4 (Shows Random noise input using MATLAB)

2.1.3 IMPACT DISTURBANCE SYSTEM

Impact disturbances simulate sudden events like potholes or bumps, testing the EPS machine's capability to handle abrupt force adjustments.

Pulse Generator Configuration:

- Number of generators: 3 - Adds variety in timing to duplicate numerous road events.
- Amplitude: 1.5 - Represents the force impact experienced in unexpected street obstacles.
- Period: 10 - Duration of noise generation in total.

• Pulse width: 5% of Period - Represents the duration of every impact, allowing for brief but good sized force modifications.

- Random selection enabled - Introduces randomness, mimicking the unpredictable nature of real-world road impacts.

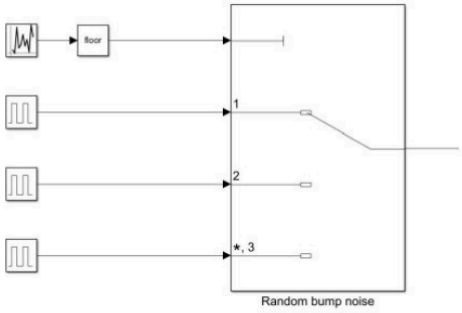


Fig. 5 (Different Pulse Generators)

2.1.4 WIND FORCE EFFECTS

Wind forces simulate aerodynamic effects at the vehicle's steerage, presenting periodic side-to side forces.

Sine Wave Configuration:

- Amplitude: 1.0 - Models the depth of lateral force from traditional wind situations.
- Frequency: 0.5 rad/sec - Sets a mild frequency for wind disturbances, representing typical gust patterns.
- Bias: 0 - Maintains a balanced push and pull effect, simulating forces without favouring one direction.
- Continuous operation - Ensures that wind outcomes act constantly, testing the EPS's response to consistent disturbances.

Fig. 6 (The noise is amplified by a gain of 0.8.)

2.1.5 DRIVER INPUT SYSTEM

Indeed, this is one of the applications whereby motive force input system interacts naturally with EPS reaction and driver intention. By simulating human steering inputs and the conditions surrounding such inputs, a realistic driving input signal is delivered to the EPS control system.

Signal Production:

Steering behaviour is modelled here with sine waves for steering movements.

- Amplitude: 1: This simulates how much steering action is inputted by the driver.
- The frequency of input activity by the driver is being simulated at a rate of 1 rad/sec.

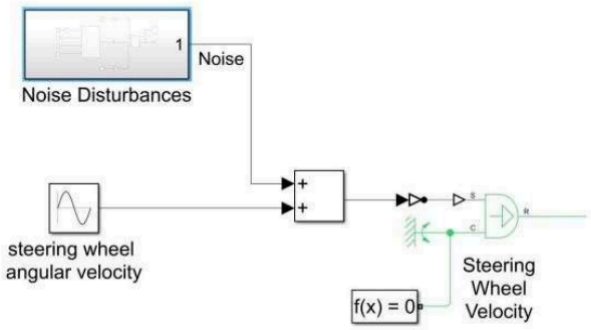


Fig. 7 (Noise Disturbances and Steering inputs passing through Steering Wheel Velocity Model)

2.2 SIGNAL PROCESSING:

The S-PS converter prepares the steering signals for transformation into a physical format, which can be processed by the EPS system, for their actuator interface.

• Noise signal integration: This adds unpredictability to the driver input, thus increasing the fidelity of the testing environment.

2.2.1 MECHANICAL SYSTEM:

The purpose of the steering mechanics simulation is to study important physical parameters affecting steering dynamics and driving experience. The other goal is to justify the steering system as a physical representation.

2.2.2 TORSION BAR IMPLEMENTATION:

Elicits feedback with the necessary twisting torque at the wheel of the steering joint resistantly mimicking his actual gestures in steering while assisting him in measuring his steering angle.

2.2.3 MECHANICAL PROPERTIES:

Spring rate of 10 Nm/rad; this perfectly balances the effort with the feedback signalling stiffness. Through the range of steering, the linear reaction curve provides uninterrupted, constant reliable feedback.

Bidirectional, allowing feedback torque in both directions.

Direct mechanical coupling: Preserving the integrity of control, this directly couples steering feedback to driver input.

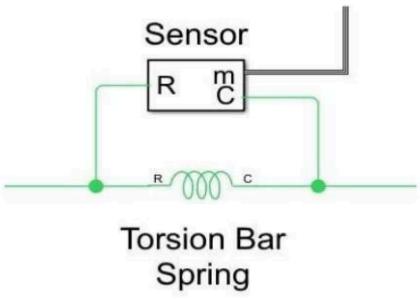


Fig. 8 (Torsion Bar Spring)

2.2.4 STEERING COLUMN CHARACTERISTICS

The steering column's inertial properties simulate rotational mass, essential for realistic steering dynamics.

Inertial Configuration:

• Moment of inertia: 1e-5 kgm² - Represents the rotational mass, influencing how the column responds to steerage.

• Single port interface - Simplifies model structure at the same time as capturing crucial dynamics.

Damping System

The damping device introduces speed-dependent resistance, stabilizing the steering and smoothing out sudden inputs.

Damping Configuration:

• Coefficient: 5 Nm/(rad/s) - Balances resistance and responsiveness, critical for natural steering.

• Linear function - Ensures predictable behaviour, aiding driver control.

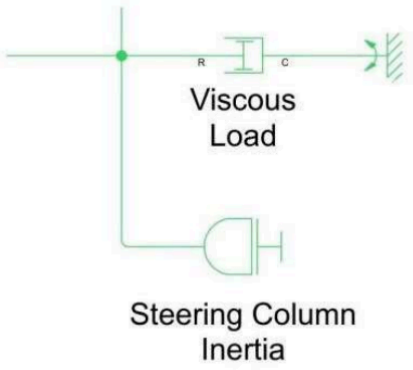


Fig. 9 (Steering Column)

2.3 ELECTRIC ASSIST SYSTEM - PMSM MOTOR

The Permanent Magnet Synchronous Motor is one electrical apparatus that makes use of permanent magnets in the rotor for field generation and thus yields very high efficiency along with accurate torque control. These features make such machines most suited for applications such as Electric Power Steering (EPS), where smooth torque delivery is required with minimum energy loss. High efficiency, compactness, and quick response ensuring an accurate control flexibility of a PMSM make it a recommended choice for EPS systems, where steering assistance is supposed to be faultlessly controlled and delivered for maximum performance and safety.

2.3.1 MOTOR CONFIGURATION:

• Three-section Wye-wound shape - Ensures balanced operation and consistent torque.

• 4 pole pairs - Balances torque and speed requirements.

• Constant Ld, Lq, and PM flux version - Simplifies control without sacrificing accuracy.

2.3.2 ELECTRICAL CHARACTERISTICS:

- Permanent magnet flux linkage: 0.02778 Wb - Defines magnetic field strength for torque generation.
- D-axis and q-axis inductance (Ld, Lq): 0.0006 H - Determines current response for stable torque.

- Zero-sequence inductance (L0): 0.00016 H - Ensures smooth current distribution.
- Phase resistance: 0.4 Ohm - Controls current flow and impacts power performance.

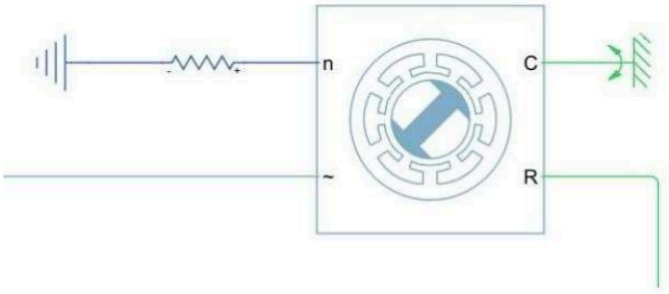


Fig. 10 (Permanent Magnet Synchronous Motor Controller for Motor)

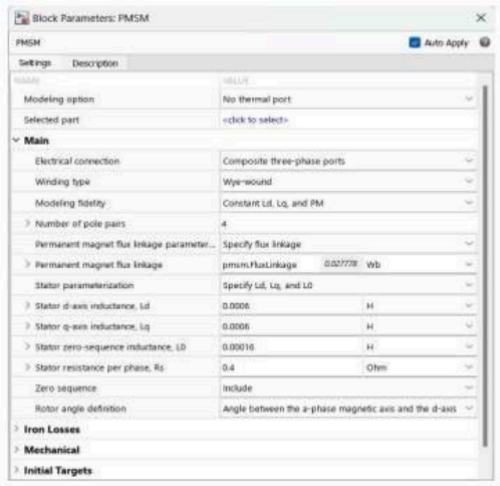


Fig. 11 (Parameters for PMSM Controller)

2.4 SENSOR IMPLEMENTATION

Sensors provide vital feedback, measuring torsion bar movement to accurately gauge steering input and response.

Sensor Configuration:

Dual rotational Motion sensors – Two Ideal Rotational Sensors are used with connections to R & C ports of measurement to gauge the changes in deflection, angle and angular velocity across the torsion bar spring which replicates the steering column. The effect of damper and steering wheel inertia is also included in the final measurement.

Output Processing:

Deflection Measurement - Used to detect the amount of angle change in a torsion bar, which is essential for measuring feedback.

• Angular velocity measurement - Measurement of the steering velocity. The same parameters is used for the dynamic adjustments of the system.

• Position monitoring - This is the constant tracking of steering positions for consistency in control.

• This is defined as the bus signal compilation that compiles signals for an optimized control processing system.

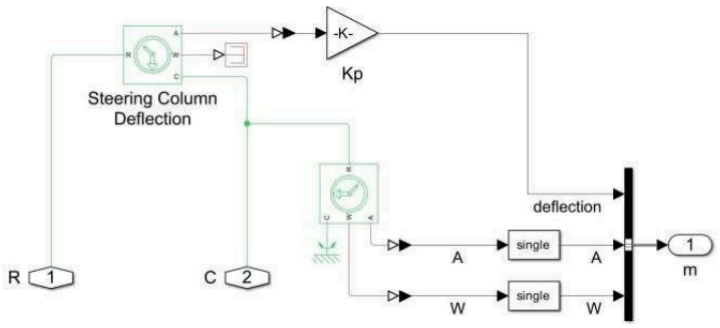


Fig. 12 (Parameters taken from Steering Column Deflection)

2.5 CONTROL SYSTEM IMPLEMENTATION

Proportional-Integral-Derivative hysteresis, a PID controller, is a feedback system to keep any desired performance by manipulating outputs corresponding to measured errors between a setpoint and the current measurement. In an Electric Power Steering system, the proportional control is used for the immediate response to steering input; the integral is correcting the accumulated errors, e.g., small position drifts; and derivative response anticipates and dampens rapid changes, thus stabilizing the system. This results in accurate, smooth steering assistance in response to the driver's command. Hence, the PID controller is the backbone for any application in EPS to improve driving comfort and safety.

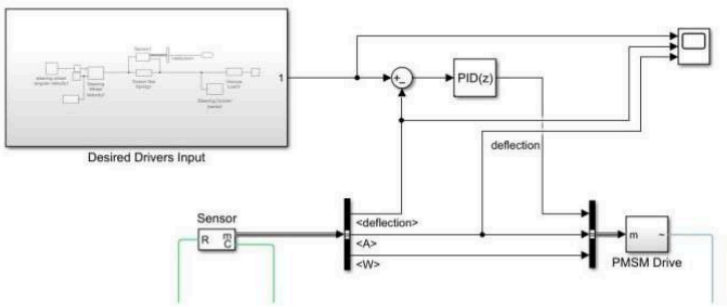


Fig. 13 (PID Controller Implementation to get desired outputs from Driver's Input)

2.6 PID CONTROLLER:

Proportional advantage: 3.326 - A proportional gain determines the response of the present error in the system. The present value of 3.326 suggests that the system reacts to deviations with speed while controlling them.

Integral gain: 12.841 - The integrated gain eliminates the steady-state error as it accounts for errors in time, their summation over time. A value of 12.841 causes small errors over time to be corrected as much as possible in accuracy.

Derivative gain: -0.0076 - Derivative gain takes care of the error rate by slowing down the change in error, preventing overshooting, and controlling oscillations. The negative value (-0.0076) assists damping, thereby preventing the need for too aggressive adjustments.

Filter coefficient: 433.60 - It inhibits high-frequency noise and ensures that the controller does not respond to very small insignificant fluctuations.

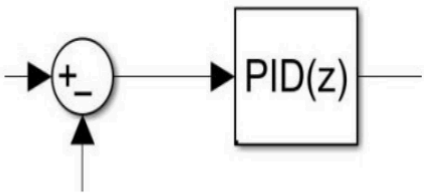


Fig. 14 (PID Controller)

LINEARIZATION SETTINGS:

- Sample time: -1 (inherited) - Matches controller timing with simulation's signal, adapts to the type of input signal given instead of rejecting different types of input.
- Algorithm: Block by using block analytic - calculates the system's linearization on a per block basis, which provides precise and reliable adjustments for each block in the simulation. It ensures that changes in the system are accounted for accurately during the linearization process.
- Rate conversion: Zero-Order Hold - The Zero-Order Hold (ZOH) is used to convert continuous signals into discrete ones by holding each sample value constant over the sampling period. This helps prevent signal fluctuations during sampling and ensures smooth signal transitions, making it ideal for controlling real-time systems.

PID Tuning: The PID controller was tuned using the PID Auto Tuner app in MATLAB, which utilizes a transfer function-based approach for PID tuning.

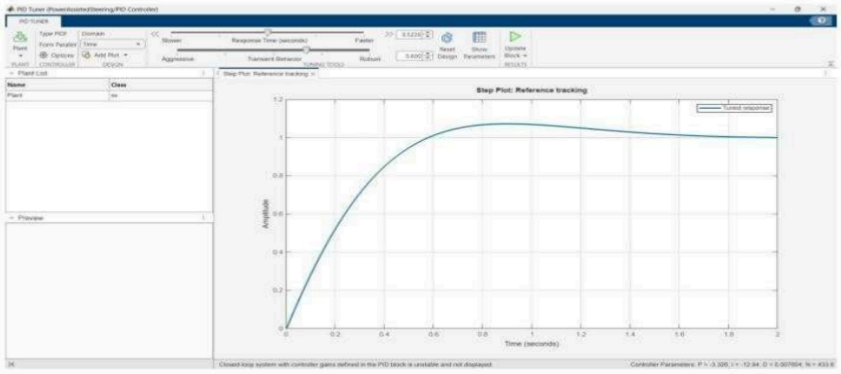


Fig. 15 (Stop Plot Reference Tracking)

2.7 PMSM DRIVE:

Inputs:

- **Motor on**: Binary input (1/0) that enables or disables the motor operation. Essential for system safety and power management
- **Command type:** Set to EnumCommandType.Torque for torque control mode. Determines how the system processes the command values
- **Command value (deflection):** Represents the steering input deflection from the driver. Converted to appropriate torque commands for the motor
- **alpha and omega**: Control parameters derived from angular positions and velocities. Help optimize motor performance and response characteristics
- **Phase currents (Iabc):** Feedback signals from the three phases of the motor. Used for closed loop current control and torque regulation.

Outputs:

- **pwm compare:** Generated PWM signals for inverter control. Determines the switching pattern of power electronics.
- **Vabc (Phase Voltages):** Three-phase voltages applied to the motor windings. Controls the electromagnetic torque production.

- **Iabc (Phase Currents):** Measured motor phase currents. Used for monitoring and feedback control purposes.
- **error**: Control system error signal. Helps maintain accurate motor response and system stability.

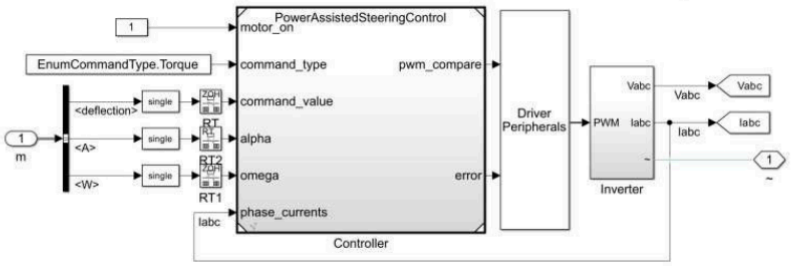


Fig. 16 (Desired Driver's Input)

2.8 TRAJECTORY AND PATH PREDICTION USING PURSUIT CONTROL MODEL

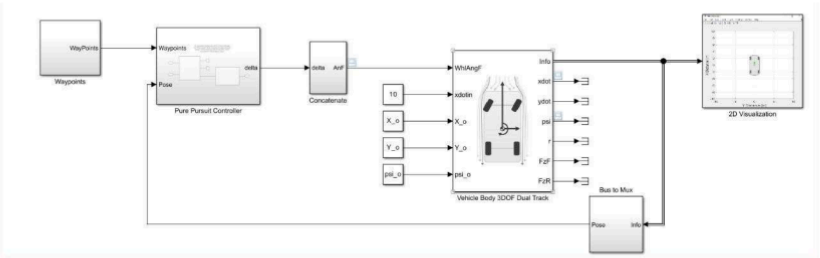


Fig. 17(Pure Pursuit and PI Controller to drive)

The model developed in Simulink presents an autonomous vehicle path-tracking application via the pure pursuit controller attached with a 3-DOF dual-track vehicle dynamics model. The system was designed to perform the following functions: track predefined waypoints, compute the steering commands, simulate the vehicle dynamics, and plot the trajectory.

This Waypoints block essentially defines a path the vehicle is to follow, a set of coordinates defining the desired route.

Pure Pursuit Controller

• Responsible for the path tracking, the controller calculates the required steering angle (delta) based on the current pose of the vehicle as well as the target waypoints.

• The controller is constantly adjusting the steering angle to align the vehicle towards the next waypoint in order to minimize the lateral deviation.

Concatenation Block

• Concatenate block combines the calculated steering angle (delta) with other vehicle parameters (like AnF acceleration input) to form the complete input vector for the dynamics model.

Vehicle Dynamics Model

• The Vehicle Body 3DOF Dual Track block facilitates the simulation of the vehicle's physical behaviour using a three-degree-of-freedom approach involving longitudinal, lateral, and yaw motions.

The inputs are as follows:

- WhlAngF: Front wheel steering angle.
- xdotin: Initial longitudinal velocity.
- X_o, Y_o: Initial global position coordinates.
- psi_o: Initial yaw angle (orientation).

The model output includes information on important dynamic parameters such as longitudinal velocity (xdot), lateral velocity (ydot), yaw angle (psi), yaw rate (r), and front/rear vertical forces (FzF, FzR).

Visualization and Feedback

The dynamic outputs will be fed to a 2D Visualization block that presents a graphic representation of the vehicle's trajectory against the waypoints for assessing path tracking performance in real-time.

At the same time, the vehicle's pose is fed back into that controller via the Bus to Mux block, thereby closing the loop in control for continuous trajectory correction.

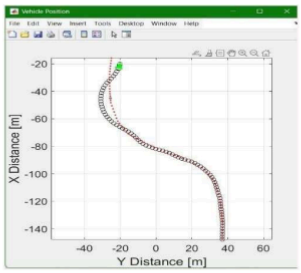


Fig. 18.a (Id==1>> avg path prediction >>high oscillation)

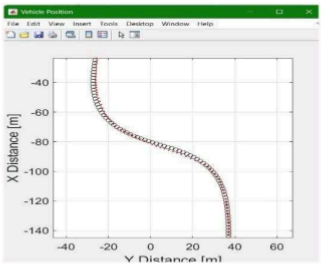


Fig. 18.b (Id==20>> Good path prediction>> low oscillation)

The simulation results are presented in two visualization plots, highlighting the impact of varying the lookahead distance (ld) parameter on path tracking performance.

1. **Path Prediction with High Oscillation (Id = 1)**

- The first plot shows the vehicle's trajectory (black circles) compared to the reference path (red dotted line).
- **Observation**: The trajectory demonstrates noticeable oscillations, indicating instability in path tracking. This is likely due to a smaller lookahead distance, resulting in the controller overcorrecting frequently.

2. **Optimized Path Prediction with Low Oscillation (Id = 20)**

- The second plot presents a smoother trajectory closely following the reference path.
- **Observation**: With an optimized lookahead distance, the oscillations are significantly reduced, resulting in a more stable and accurate path prediction.

CHAPTER 3

RESULTS AND DISCUSSION

```
%% add Image to the path
addpath(genpath('Images'));

%% load the scene data file generated from Driving Scenario Designer
load('curveLowVel.mat');

%% define reference points
refPose = data.ActorSpecifications.Waypoints;
xRef = refPose(:,1);
yRef = -refPose(:,2);

%% define reference time for plotting
Ts = 50; % simulation time
s = size(xRef);
tRef = (linspace(0,Ts,s(1)))'; % this time variable is used in the "2D

%% define parameters used in the models
L = 3; % length
ld =20; % lookahead distance
X_o = refPose(1,1); % initial vehicle position
Y_o = -refPose(1,2); % initial vehicle position
psi_o = 0; % initial yaw angle
```

Fig. 19 (Code for Path Prediction)

3.1 REAL-TIME OBJECT DISTANCE DETECTION IN DYNAMIC ENVIRONMENTS USING OPENCV:

The present computer vision-based distance estimation application, recently developed in a Python framework for use on OpenCV, is intended for real-time object detection and distance estimation with a webcam. It works in a manner in which the video stream is captured and run in real-time, processed for the detection of certain objects, and measuring their distance from the camera by means of the pinhole camera model.

In this method, the object size and camera focal length are used to estimate a distance; this method uses the size of an object to be known and also the camera's focal length to have an approximate idea about the distance.

The key steps are:

Video Frame Capture: The webcam video keeps rolling to allow real-time processing of frames.

Background Subtraction: To concentrate on moving objects while ignoring static elements, strategies like background subtraction (for instance, MOG2 or KNN) are used to eliminate the stationary objects and enhance the detection capability of relevant moving objects.

Preprocessing and Noise Filtering: Morphological operations such as dilation and erosion are performed for removing small noise particles and improving the visibility of object edges.

Some smoothing techniques like Gaussian blurring or median filtering can be applied to reduce image noise to avoid some false detections**.**

Contour Extraction and Object Identification: A contour is extracted of the object, and around the shape, a bounding box is drawn. Then, it identifies an object that is the nearest one from the detection range.

Distance Estimation Using Pinhole Camera Model:

This program estimates the distance of an object by the following formula:

$$D=(W\times F)/P$$

Where:

D is the estimated distance,

W is the actual width of the object,

F is the focal length of the camera,

P is the perceived width of the object in pixels.

The focal length (F) is calculated using a calibration method with a reference object located at a known distance.

Distance Information Overlaying on Video Stream:

The computed distance gets dynamically updated and superimposed over the video feed such that the user sees it in real-time.

Applications:

The main applications of the system, hence being able to operate in dynamic areas, will be:

Autonomous Navigation: Assisting robots and self-driving vehicles in avoiding obstacles and understanding their surroundings.

Object Tracking: The identification and tracking of moving objects in applications such as surveillance and security.

Proximity Alert Systems: These trigger alerts when an object goes beyond its acceptable proximity threshold, which finds applications in collision avoidance systems used for drones and industrial automation.

Incorporating computer vision and online processing in these implementations further elevates object detection and distance measurement accuracy, which ultimately broadens this technique's application scope for most of the variety of vision-based applications.

```python
import cv2
import numpy as np

# === Parameters ===
KNOWN_WIDTH_OBJECT = 1.0  # Width of the smallest object (e.g., pin) in m
FOCAL_LENGTH = 500        # Adjust this value after calibration
MAX_DISTANCE = 30.0       # Maximum distance to detect objects in m

# === Distance Calculation Function ===
def calculate_distance(known_width, focal_length, per_width):  1 usage
    """
    Calculate the distance from the object to the camera.
    """
    return (known_width * focal_length) / per_width

# === Camera Initialization ===
cap = cv2.VideoCapture(0)  # Use the default camera

if not cap.isOpened():
    print("Camera not detected.")
    exit()

# === Background Subtractor ===
back_sub = cv2.createBackgroundSubtractorMOG2(history=100, varThreshold=50, detectShadows=False)

print("Press 'q' to quit the application.")
```

```python
    # === Object Detection and Distance Calculation ===
    closest_distance = MAX_DISTANCE + 1
    closest_contour = None

    for contour in contours:
        # Filter small contours
        if cv2.contourArea(contour) > 100:
            x, y, w, h = cv2.boundingRect(contour)

            # Calculate distance
            distance = calculate_distance(KNOWN_WIDTH_OBJECT, FOCAL_LENGTH, w)

            # Find the closest object within the distance range
            if distance < closest_distance and distance < MAX_DISTANCE:
                closest_distance = distance
                closest_contour = contour

    # === Display the closest object and its distance ===
    if closest_contour is not None:
        x, y, w, h = cv2.boundingRect(closest_contour)
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.putText(frame, text: f"Distance: {closest_distance:.2f} m", org: (x, y-10),
                    cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.6, color: (0, 255, 0), thickness: 2)
        print(f"Object detected at: {closest_distance:.2f} m")
```

Fig. 20 (Python Code for Real-Time Object Distance Detection in Dynamic Environments Using OpenCV)

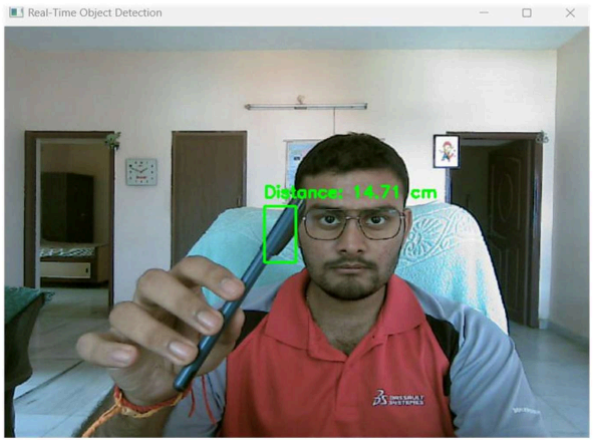


Fig. 21a (OpenCV camera vision)

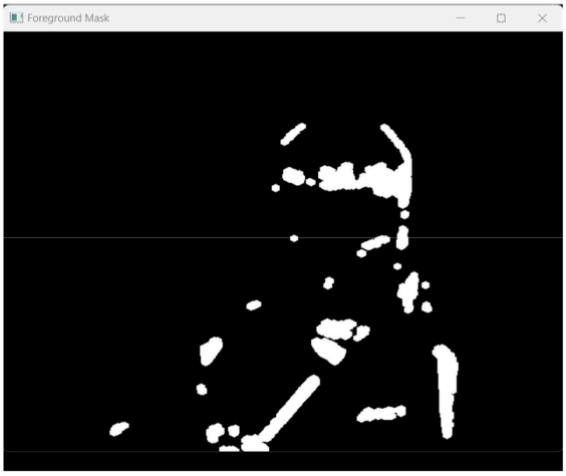


Fig. 21b (OpenCV Fg Mask)

3.2 VISION BASED BRAKING AND STEERING SYSTEM DETECTION:

Provides an intelligent braking and steering control system for autonomous vehicles through computer vision and machine learning. The system aims to improve the safety features of a vehicle through real-time detection, estimation of distance, and adaptive mechanisms to avoid collision.

Main Features of the System:
Camera Module & Obstacle Detection

The Camera Module captures real-time video streams using an OpenCV compiler. Objects and obstacle detection is made with contour analysis and bounding box extraction. The distance to obstacles is estimated with the help of a pinhole camera model by considering focal length and object width**.**

AI-Based Decision Making for Braking and Steering:
The Autonomous Car AI initiates action if two conditions are met: the obstacle is within a critical distance threshold, and this has been maintained over a significant duration. Multilayer Perceptron Regressor (Progressor) model is used by the system to make decisions. The AI model accurately decides the real-time optimum brake force and deflections of steering.

Flowing Dynamic Vehicle Parameter Adjustment:

Thus it employs some critical dynamic changes for controlled and safe deceleration, such dynamic parameters are:

Braking pressure - Changes itself to the proximity of obstacles.

Reducing the fuel flow helps in making the vehicle stop smoothly.

Steering angle - Change in steers for safe navigation.

Brake Temperature- Monitored to prevent overheating.

Engine temperature- Ensured within optimum operating limits.

Performance Monitoring and Safety Reporting:

The system is concerned with tracking the application of braking and steering. Upon successful stopping of the vehicle, an in-depth safety reporting is carried out. The report consists of performance graphs depicting the vehicle's responses, braking force, and trajectory corrections carried out.

Advanced AI-Based Navigation & Control System:

The project controls not just braking and steering, but also performs real-time monitoring of the vehicle and dynamic adjustments to navigation by means of AI detection of objects and lane identification.

Key Features:
YOLO-based Object Recognition:

The YOLO (You Only Look Once) model detects items around a car. Using the size of bounding boxes and focal lengths, it can estimate the distance to obstacles. Lane Categorization Safe Manoeuvring. Hinders are restricted into their respective lanes left, centre, or right lanes. If an object such as the current one is blocked with obstacles from the current lane, the system suggests making a safer lane change.

Dual PID Controller for Vehicle Dynamics:

Speed Control PID: Formula by which speed can be controlled through distance.

Steering Control PID: Responsible for adjusting the angle of steering towards safe navigation around obstacles.

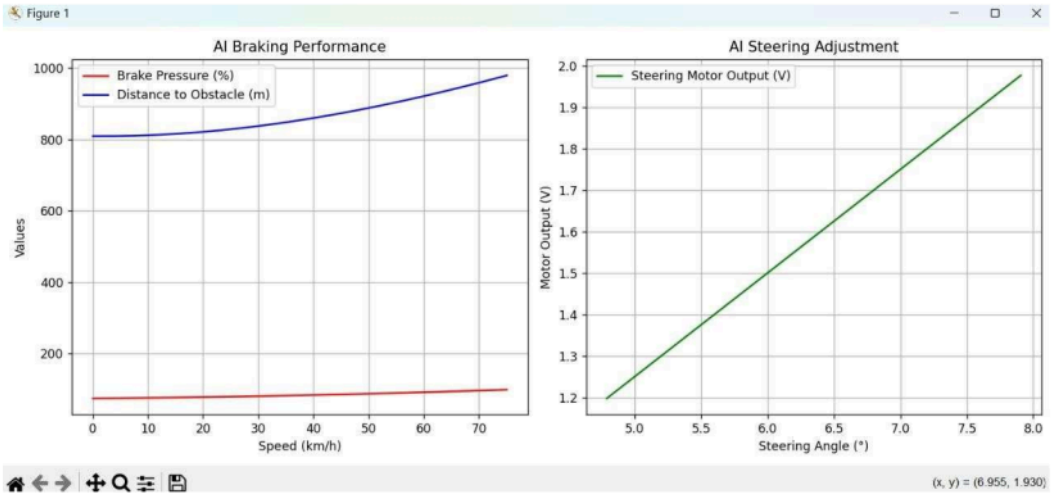
Real-Time Vehicle Parameter Monitoring:
The system computes various important parameters in real-time.
Pressure in the brake - safe deceleration.
Fuel reduction - Energy utilization ratio improved.
Monitoring engine temperature - Prediction of overheating risk.
Brake temperature - Avoids significant wear and tear.
Visual Feedback for Driver & System Monitoring:

The real time is expressed with the help of annotated video frames.

CODE:

```
class CameraModule:
    def __init__(self, focal_length=700, real_object_width=1.8):
        """Camera module to estimate distance from an obstacle."""
        self.focal_length = focal_length
        self.real_object_width = real_object_width
        self.cap = cv2.VideoCapture(0)

    def get_distance_from_camera(self):
        """Detects motion and estimates real-world distance."""
        ret, frame = self.cap.read()

    def monitor_distance(self, duration=10):
        """Monitors the distance for a given duration to determine if braking is needed."""
        start_time = time.time()
        below_threshold_time = 0

        while time.time() - start_time < duration:
            distance = self.get_distance_from_camera()
            if distance:
                print(f"📏 Camera Detected Distance: {distance}m")
                if distance < 500:
                    below_threshold_time += 1

            time.sleep(1)

        return below_threshold_time >= duration  # Apply brakes if true

    def release_camera(self):
        """Releases the camera resource."""
        self.cap.release()
        cv2.destroyAllWindows()
```

OUTPUT:



```
Detected Distance: 6.49m
Detected Distance: 10.0m
Detected Distance: 12.48m
Detected Distance: 52.5m
Detected Distance: 315.0m
Detected Distance: 60.0m
Detected Distance: 7.24m
Detected Distance: 28.0m
Detected Distance: 8.75m
Detected Distance: 252.0m
☑ Safe distance maintained: NO BRAKING REQUIRED.

🚗 Speed: 49.4 km/h | 🔵 Distance: 55.6 m
Brake Pressure: 36.6% | Fuel Flow Reduction: 29.3%
Brake Temperature: 170.0°C | Engine Temp: 97.0°C
➡ Steering Angle: 2.0° | ⚙ Steering Motor Output: 0.5V

🚗 Speed: 47.6 km/h | 🔵 Distance: 42.4 m
Brake Pressure: 35.3% | Fuel Flow Reduction: 28.2%
Brake Temperature: 180.0°C | Engine Temp: 97.5°C
➡ Steering Angle: 1.3° | ⚙ Steering Motor Output: 0.3V

🚗 Speed: 45.9 km/h | 🔵 Distance: 29.6 m
Brake Pressure: 34.0% | Fuel Flow Reduction: 27.2%
Brake Temperature: 190.0°C | Engine Temp: 98.0°C
➡ Steering Angle: 0.7° | ⚙ Steering Motor Output: 0.2V

🚗 Speed: 44.3 km/h | 🔵 Distance: 17.3 m
Brake Pressure: 32.7% | Fuel Flow Reduction: 26.2%
Brake Temperature: 200.0°C | Engine Temp: 98.5°C
➡ Steering Angle: 0.1° | ⚙ Steering Motor Output: 0.0V

🚗 Speed: 42.7 km/h | 🔵 Distance: 5.4 m
Brake Pressure: 31.5% | Fuel Flow Reduction: 25.2%
Brake Temperature: 210.0°C | Engine Temp: 99.0°C
➡ Steering Angle: 0.0° | ⚙ Steering Motor Output: 0.0V

🚗 Speed: 41.2 km/h | 🔵 Distance: 0.0 m
Brake Pressure: 30.4% | Fuel Flow Reduction: 24.3%
Brake Temperature: 220.0°C | Engine Temp: 99.5°C
➡ Steering Angle: 0.0° | ⚙ Steering Motor Output: 0.0V

☑ Car Stopped Safely.

📋 Final AI Safety Report:
Brake Fade: 9.5%
Final Brake Temperature: 220.0°C
Final Engine Temperature: 99.5°C
Brake Fluid Level: 100.0%
Oil Level: 100.0%
```

PARAMETERS USED:

```
# === Parameters ===
STOP_DISTANCE = 30.0  # Minimum distance before stopping (cm)
KNOWN_WIDTH_OBJECT = 1.0  # Width of the smallest object in cm
FOCAL_LENGTH = 500        # Adjust this value after calibration
MAX_DISTANCE = 100.0      # Maximum distance to detect objects in cm
SAFE_DISTANCE = 50.0      # Safe following distance in cm
MAX_SPEED = 80.0  # Maximum speed (km/h)
MIN_SPEED = 10.0  # Minimum speed to keep moving
```

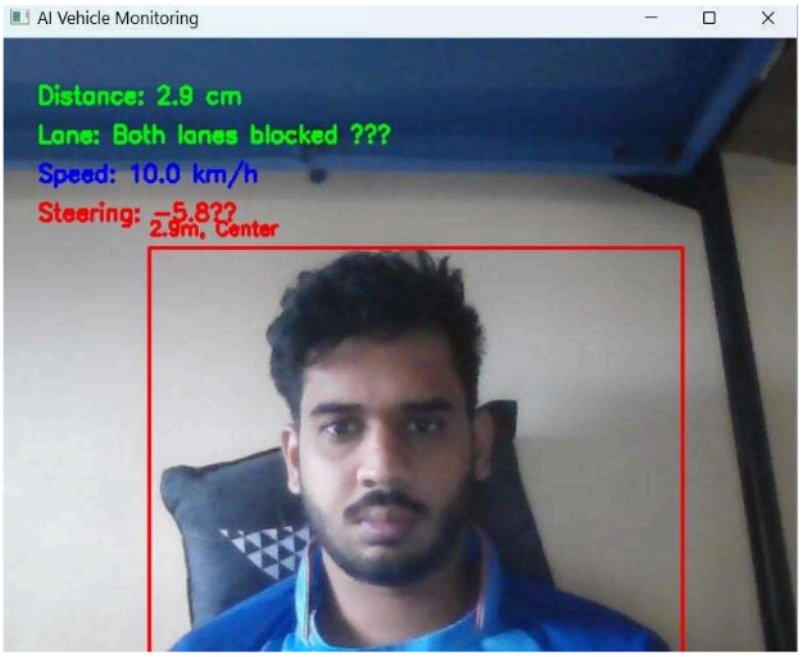
Background Subtractor:

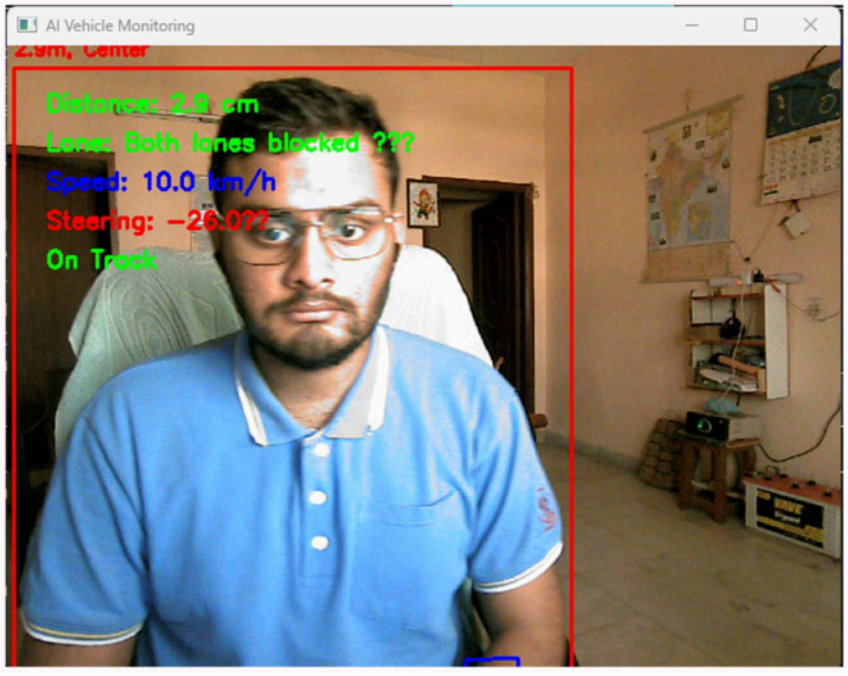
```
# === Background Subtractor ===
back_sub = cv2.createBackgroundSubtractorMOG2(history=100, varThreshold=50, detectShadows=False)

def estimate_distance(bbox_width, focal_length=700, real_width=1.8):
    """Estimate distance based on bounding box width."""
    return (real_width * focal_length) / (bbox_width + 1e-6)

def determine_lane(x_center, frame_width):
    """Determine which lane the object is in."""
    lane_width = frame_width // 3
    if x_center < lane_width:
        return "Left"
    elif x_center < 2 * lane_width:
        return "Center"
    else:
        return "Right"

def get_safety_lane(current_lane):
    """Determine the safest lane to move to."""
    if current_lane == "Left":
        return "Move to Center or Right"
    elif current_lane == "Center":
        return "Move to Left or Right"
    else:
        return "Move to Center or Left"
```

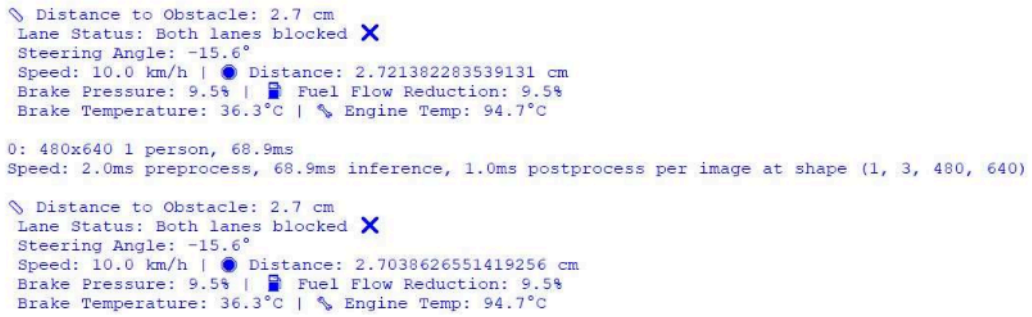
Lane Detection:

```
lane_status = "Left lane clear ☑" if closest_distance > SAFE_DISTANCE else "Maintaining Safe Distance ⓘ" if closest_distance > STOP_DISTANCE else "Both lanes blocked
steering_angle = steering_pid(-1 if "Left" in lane_status else 1)
speed_adjustment = speed_pid(closest_distance)
current_speed = max(MIN_SPEED, min(MAX_SPEED, MAX_SPEED - speed_adjustment))
brake_pressure = max(0, min(100, (SAFE_DISTANCE - closest_distance) / 5))
fuel_reduction = min(100, brake_pressure)
engine_temp = min(110, 90 + (brake_pressure / 2))
brake_temp = min(100, 30 + (brake_pressure / 1.5))

print(f"\n\ Distance to Obstacle: {closest_distance:.1f} cm")
print(f" Lane Status: {lane_status}")
print(f" Steering Angle: {steering_angle:.1f}°")
print(f" Speed: {current_speed:.1f} km/h | ● Distance: {closest_distance} cm")
print(f" Brake Pressure: {brake_pressure:.1f}% | ▮ Fuel Flow Reduction: {fuel_reduction:.1f}%")
print(f" Brake Temperature: {brake_temp:.1f}°C | ✎ Engine Temp: {engine_temp:.1f}°C")
```

Output:


AI Vehicle Monitoring
Distance: 2.9 cm
Lane: Both lanes blocked ???
Speed: 10.0 km/h
Steering: −5.8??
2.9m, Center


AI Vehicle Monitoring
Distance: 2.9 cm
Lane: Both lanes blocked ???
Speed: 10.0 km/h
Steering: −26.0??
On Track



Mid Lane Detection:

The project implements object detection and computer vision techniques in the development of a Mid-Lane Warning System that monitors and guides pedestrians to lock into an aligned lane. OpenCV's HOG-based people detectors and distance estimation techniques provide real-time feedback on the position of the object in the view of the camera.

Real-Time Object Detection:

Uses OpenCV's HOG pedestrian detector based on a pre-trained SVM classifier. Obtains live streaming from webcam for real-time detection of people in the scene.

Lane Centering and Error Calculation:

Find the middle point of the frame (i.e. mid-lane).Calculates the center offset (error) of the object from the lane middle.Suggests correction of movement to summon the object in the lane.

Distance Estimation by the Pinhole Camera Model from the Detected Person:

Calculates an approximate distance in real coordinates with respect to the detected person:

Distance = (Real Width of Person × Focal Length) / Bounding Box Width

It outputs the smoothed distance in millimeters using a moving average filter.

Average Moving Smoothing:

This uses a deque buffer-based storage that holds recent values of errors and distances. Smoothing brings in stable procedures of calculations.

Dynamic Lane Adjustment Suggestions:

If centered, it'll say "Centered" (error < 20px).

Drift Left, it will say "Move Right."

Drift Right, it says "Move Left."

That is how it holds towards the pedestrian side of the lane.

GUI (Graphical User Interface) Overlays:

Draws boxes around the detected people**.**

Shows live texts for-

-Distance of detected object in mm.

-Status of alignment (Centered, Move Left, Move Right).

-Error (px) from center of the lane.

-Vertical blue reference line which is the lane center.

Console/Linux Based Logging for Debugging:

Real time monitoring of smoothed distance, alignment error and suggested actions.

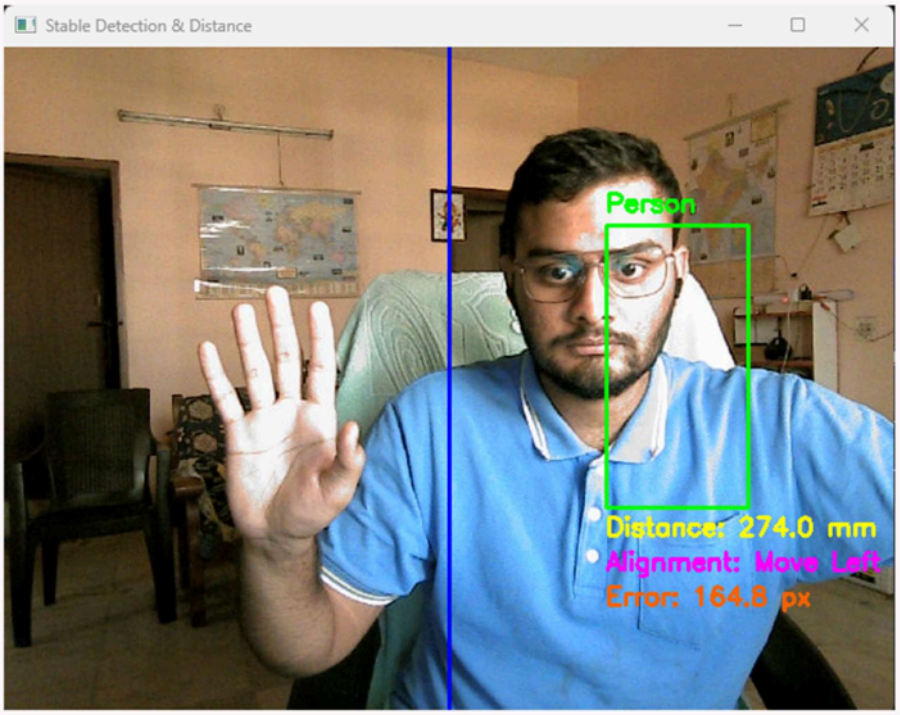
Safe Exit Mechanism:

Push 'q' to terminate execution, release webcam, close all open OpenCV windows.

PYTHON CODE:

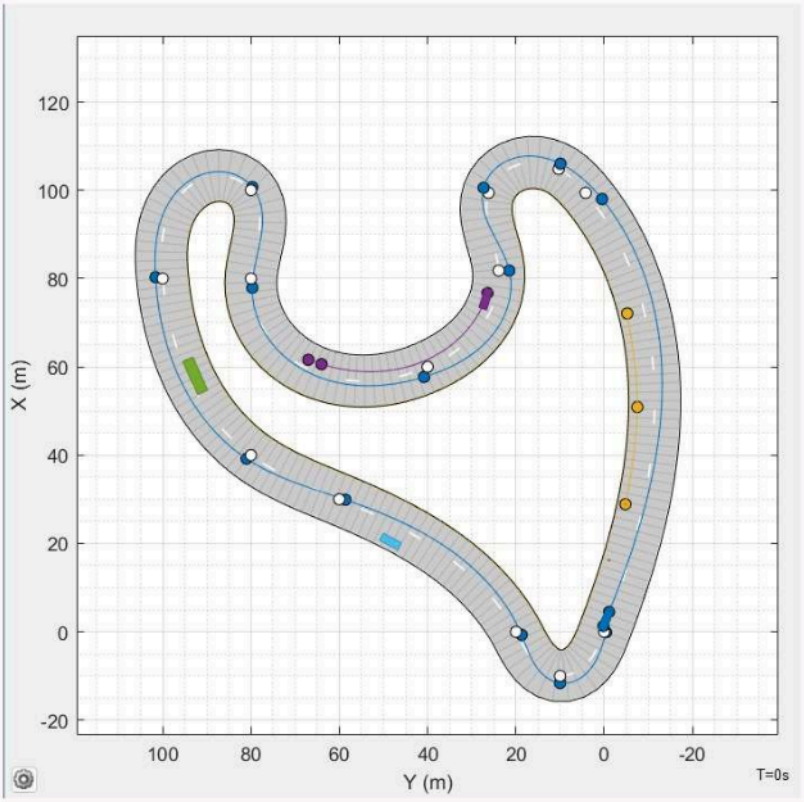
```python
import cv2
import numpy as np
from collections import deque

# Initialize HOG detector
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

FOCAL_LENGTH = 700
REAL_WIDTH_PERSON = 40  # in cm

cap = cv2.VideoCapture(0)

# Smoothing buffers (deque for moving average)
distance_buffer = deque(maxlen=5)  # Adjust window size for smoother effect
error_buffer = deque(maxlen=5)

while True:
    ret, frame = cap.read()
    if not ret:
        print("Camera feed error.")
        break

    (h, w) = frame.shape[:2]
    center_x = w // 2

    boxes, weights = hog.detectMultiScale(frame, winStride=(8, 8))

    for (x, y, bw, bh), weight in zip(boxes, weights):
        if weight > 0.5:
            object_center_x = x + bw // 2
            error = object_center_x - center_x

            # Distance estimation
            distance = (REAL_WIDTH_PERSON * FOCAL_LENGTH) / bw

            # Store values for smoothing
            distance_buffer.append(distance)
            error_buffer.append(error)

            # Smooth values using moving average
            smooth_distance = round(np.mean(distance_buffer), 2)
            smooth_error = round(np.mean(error_buffer), 2)

            # Adjustment suggestion logic
            if abs(smooth_error) < 20:
                adjustment = "Centered"
            elif smooth_error < 0:
                adjustment = "Move Right"
            else:
                adjustment = "Move Left"

            # PRINT to console
            print(f"\n[Detection]")
            print(f"  Smoothed Distance: {smooth_distance} mm")
            print(f"  Smoothed Center Error: {smooth_error} px")
            print(f"  Suggestion: {adjustment}")

            # GUI Overlay
            cv2.rectangle(frame, (x, y), (x + bw, y + bh), (0, 255, 0), 2)
            cv2.putText(frame, f"Person", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
            cv2.putText(frame, f"Distance: {smooth_distance} mm", (x, y + bh + 20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 255), 2)
            cv2.putText(frame, f"Alignment: {adjustment}", (x, y + bh + 45), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 255), 2)
            cv2.putText(frame, f"Error: {smooth_error} px", (x, y + bh + 70), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 100, 255), 2)

    cv2.line(frame, (center_x, 0), (center_x, h), (255, 0, 0), 2)
    cv2.imshow("Stable Detection & Distance", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        print("Exiting...")
        break

cap.release()
cv2.destroyAllWindows()
```

OUTPUT:


Stable Detection & Distance
Person
Distance: 274.0 mm
Alignment: Move Left
Error: 164.8 px

RESULTS:

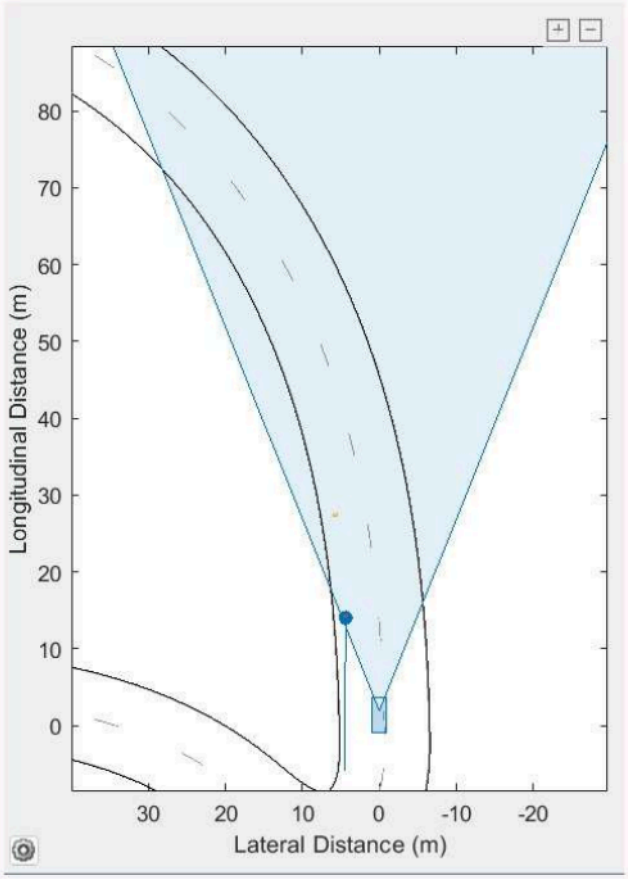
1. ROAD DESIGN



Track Overview

- This shows an entire track layout with different markers along the path.
- The shaded area represents the road boundaries.
- The small colored circles could indicate vehicles or objects detected by sensors.
- Rectangular colored objects likely represent obstacles, parked vehicles, or pedestrians.
- The visualization is probably used to track multiple vehicles and objects in the simulation.

2. BIRD EYE VIEW



Top-Down Sensor View

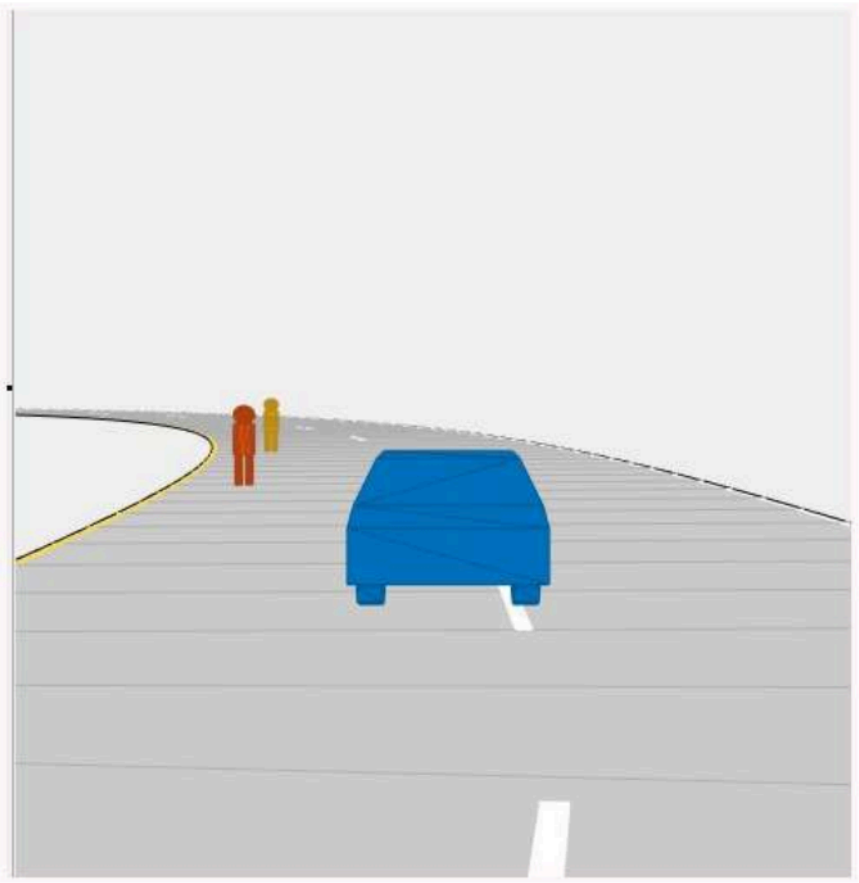
This seems to be a top-down perspective from a vehicle.

The blue shaded triangular area represents the vehicle's sensor (such as a camera or LiDAR) field of view.

The vehicle is tracking an object ahead on a curving road.

The dashed and solid lines may represent lane boundaries.

3. ECO CENTRIC VIEW



Driver's Perspective View

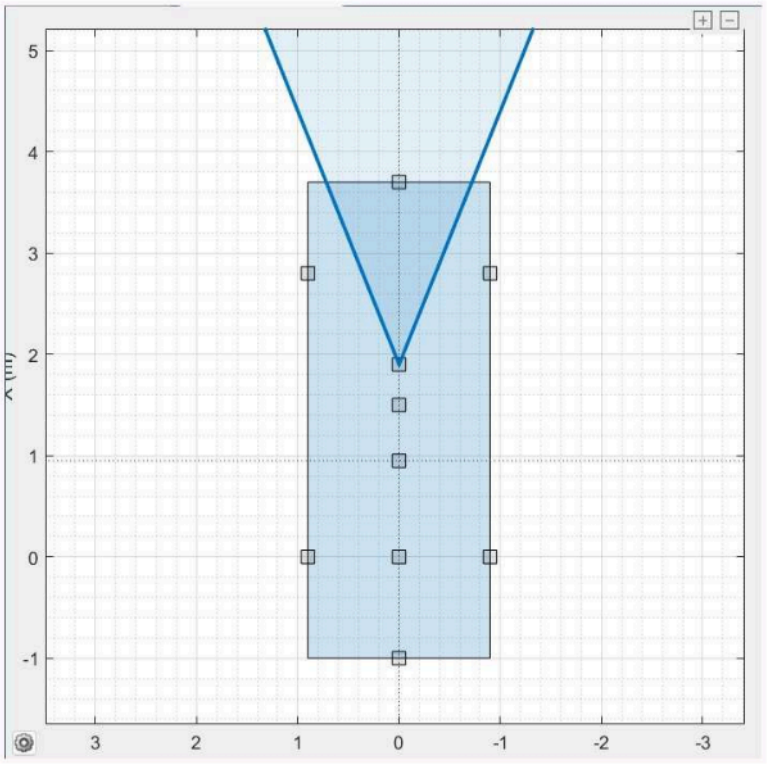
It gives a first-person viewpoint as in first-hand view of the autonomous vehicle itself.

The blue car here denotes the simulated ego vehicle.

Two pedestrians on the left, likely detected by the system.

This is the visualization of what "sees" to the vehicle using its sensors.

4. CAMERA POSITION



Vehicle Sensor Coverage

- A top-down view of the vehicle and its perception range.
- The blue shaded region indicates the field of view of the vehicle's sensors.
- The rectangular markers around the car suggest points of interest for collision detection or object avoidance.

CHAPTER 4

CONCLUSION

This paper sets out for a complete analysis of the simulation and behaviour of autonomous vehicles into a structured environment while detecting as well as responding to obstacles. The advanced perception capabilities of the vehicle involve individualized sensors such as LiDAR, cameras, and radar, which successfully identify objects, obey lane discipline, simultaneously optimize the path in real-time. The fusion of these sensors into path-planning algorithms and adaptive control strategies ensures smooth, safe navigation that attests to modern technology in autonomous driving.

The outcome of the simulation justifies the importance of real-time decision-making relating to the operation of an autonomous vehicle. Detection of pedestrians, other vehicles, and environmental obstacles is made by the vehicle, proving the need for advanced driver assistance systems (ADAS). Vision-based tracking, in response to which it makes a dynamic path adjustment, allows for greater maneuverability in difficult road conditions.

However, some challenges, such as handling unexpected obstacles, increasing reaction time in high-speed scenarios, and trajectory planning, still remain for future work. Future work might involve improving deep-learning-based perception models, optimizing motion control strategies, and providing robust performance across a variety of unstructured and diverse environments.

With autonomous vehicle technology advancing in leaps and bounds, this study can only add to the copious development of intelligent transport systems designed to be safer and effective day by day. Continuous progress in artificial intelligence, sensor technologies, and vehicle dynamics will soon see autonomous vehicles change the face of the automotive industry forever through much better safety with much less human error and mobility solutions for the future.

24BTECH20042_1

ORIGINALITY REPORT

3%	2%	2%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	docplayer.nl Internet Source	1%
2	Dan Kong, Zhiquan Feng, Jinglan Tian. "Three-dimensional Position Information Tracking Algorithm for Virtual Teaching Experiments", Proceedings of 2020 the 6th International Conference on Computing and Data Engineering, 2020 Publication	<1%
3	qje.su Internet Source	<1%
4	oa.upm.es Internet Source	<1%
5	Alkrunz, Mohammed F. M.. "Design of Discrete Time Controllers for Dc-Dc Boost Converter", Sakarya Universitesi (Turkey), 2022 Publication	<1%
6	www2.mdpi.com Internet Source	<1%

7	arxiv.org Internet Source	<1%
8	www.mdpi.com Internet Source	<1%
9	Submitted to Bilkent University Student Paper	<1%
10	cdn2.hubspot.net Internet Source	<1%
11	Naitik M. Nakrani, Maulin M. Joshi. "A human-like decision intelligence for obstacle avoidance in autonomous vehicle parking", Applied Intelligence, 2021 Publication	<1%
12	encyclopedia.pub Internet Source	<1%

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		