Report on Backup Camera Simulation with Dynamic Trajectory

Zaid Waghoo

November 26, 2023

Introduction

This report outlines the implementation of a backup camera simulation with dynamic trajectory visualization. The simulation, based on a 10-second clip recorded from a phone's perspective, aims to overlay the trajectory of the rear wheels on the video. The project involves assumptions about steering geometry, vehicle dimensions, and the application of ADAS (Advanced Driver Assistance Systems) principles.

Objective

The primary goal is to demonstrate proficiency in ADAS software development, understanding vehicle dynamics, and basic object classification. The simulation should accurately represent the trajectory of rear wheels during a backup maneuver.

1 Implementation

1.1 Vehicle Dimensions

The vehicle used for this project is a Toyota RAV4 with the following specifications:

- Wheelbase: 2.69 meters,
- Track Width: 1.69 meters,
- Steering Ratio: 14.3:1

1.2 Ackermann Steering Model

The Ackermann steering system, commonly used in car-like vehicles, minimizes tire slippage by adjusting the steering angles of the inner and outer wheels. Figure 1 illustrates this configuration, where w is the track width, l is the wheelbase, ϕ_i and ϕ_o are the steering angles, and r is the distance to the Instantaneous Center of Curvature (ICC).

The relationship between the Ackermann angle ϕ and the steering wheel input δ_{in} is defined by:

$$\phi = \frac{\delta_{in}}{\gamma} \tag{1}$$

Here, ϕ is the Ackermann angle, δ_{in} is the steering wheel input in radians, and γ is the steering ratio.

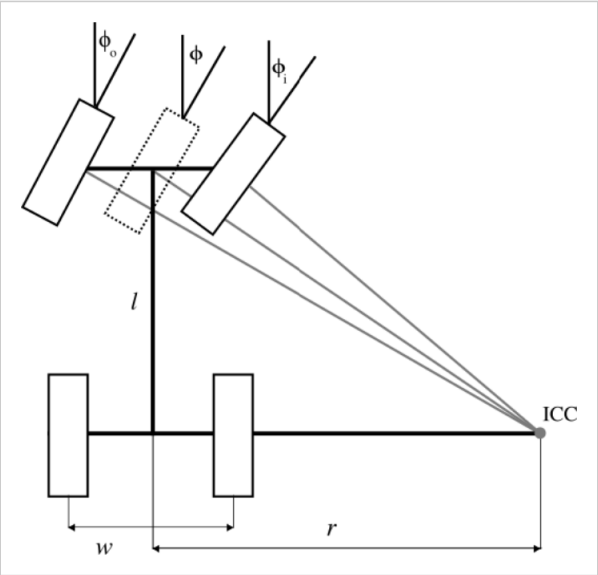

Figure 1: Simplified Ackermann Steering Model

To simulate the vehicle's trajectory, the heading direction was calculated based on the steering input over time. The forward kinematics, predicting the future state given the current configuration, is expressed as a quadruple (x, y, θ, ϕ) with δ as the heading and (x, y) as the position. Given vehicle speed s, the angular velocity ω is determined by:

$$\omega = \frac{s}{WB} \cdot \phi \tag{2}$$

Here, ω is the angular velocity, WB is the wheelbase, and ϕ is the Ackermann angle. Using Ackermann steering principles, the heading angle θ was computed at each time step, capturing the vehicle's directional orientation.

The vehicle's position was determined using kinematic equations:

$$\dot{x} = s \cos\theta \tag{3}$$

$$\dot{y} = s \sin\theta \tag{4}$$

By iterating through these equations over a specified time period, a comprehensive trajectory of the rear wheels was generated.

1.3 3D to 2D Projection

The primary concept involved projecting the trajectory's 3D points, defined in the world frame, onto the 2D image plane. This projection was determined by considering the camera's pose relative to a world origin. In this context, the world origin was chosen to coincide with the center of the rear axle, a decision influenced by treating the turning radius as the distance from the Instantaneous Center of Curvature (ICC) to the center of the rear axle.

The camera's placement was specified at an elevation of approximately 1 meter above the ground, and its lateral positioning was aligned with the rear overhang offset along the y-axis. Additionally, the camera was oriented at an angle of 70 degrees from the x-axis. It's crucial to acknowledge that these values are approximations derived from schematic representations and measurements.

The functions use a pinhole camera model for projecting 3D points onto the image plane, represented as:

$$sp = A[R|t]P_w$$

where:

P_w : 3D point in world coordinates
p : 2D pixel in the image plane
A : Camera intrinsic matrix
R : Rotation matrix
t : Translation vector
s : Arbitrary scaling (not part of the camera model)

For this project, 2D trajectory lines were converted to 3D by augmenting them with a constant depth, matching the camera height.

The camera intrinsic matrix A projects 3D points to 2D pixel coordinates:

$$p = AP_c, \quad A = \begin{bmatrix} 747.3591495 & 0.0 & 492.38024765 \\ 0.0 & 754.45639953 & 282.35015183 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Calibration was performed using Zhang's Checkerboard method and OpenCV's calibration functions.

The joint rotation-translation matrix $[R|t]$ involves a projective and homogeneous transformation:

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

Assuming the camera pose at the origin with a 70-degree clockwise rotation from the x-axis, the homogeneous transformation is:

$$P_c = [R|t]P_w, \quad [R|t] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(70) & \sin(70) & 0 \\ 0 & -\sin(70) & \cos(70) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The overall projection is given by:

$$s\begin{bmatrix} uv \\ 1 \end{bmatrix} = \begin{bmatrix} 747.3591495 & 0.0 & 492.38024765 \\ 0.0 & 754.45639953 & 282.35015183 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(70) & \sin(70) & 0 \\ 0 & -\sin(70) & \cos(70) & 0 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Using OpenCV's `projectPoints` function, accurate estimations of points in the image plane were obtained with respect to the trajectory.

1.4 Object Detection

The objective of this phase was to implement object detection for objects visible in a rear-view mirror utilizing a new or pre-trained model. The goal was to fine-tune the model using a provided vehicle dataset containing five different classes of vehicles to enable detection of various vehicle types.

The selected pre-trained model for this task was YOLO (You Only Look Once) version 8.

1.4.1 YOLO Overview:

YOLO represents a family of real-time object detection algorithms. Its core concept involves dividing an image into a grid and predicting bounding boxes and class probabilities for each grid cell. YOLO's ability to detect multiple objects in a single pass makes it highly efficient for real-time applications.

1.4.2 Fine-Tuning a YOLO Model:

Fine-tuning a YOLO model entails taking a pre-trained model and further training it on a specific dataset or task. This process aims to adapt the model to a particular domain, enhance its performance on specific classes, or handle data variations from the original training data.

1.4.3 General steps used to fine tune the vehicles dataset:

- **Pre-trained Model Selection:** Chose the YOLO V8 nano pre-trained model, which is optimized for faster execution and requires less computational resources.
- **Dataset Preparation:** Utilized a vehicles dataset from Roboflow, encompassing diverse vehicle images with corresponding bounding box annotations and class labels.
- **Fine-Tuning Process:**
  - **Model Initialization:** Loaded the pre-trained YOLO model weights.
  - **Transfer Learning:** Froze the initial layers capturing general features and modified the final layers to align with the new task.
  - **Dataset Loading:** Loaded the fine-tuning dataset, segmented into training, test and validation sets.
  - **Training:** Trained the model on the fine-tuning dataset. Adjusted hyperparameters, including learning rate and batch size.

– **Validation:** Evaluated the model's performance on a separate validation set to monitor its generalization capability.

This process allowed the YOLO model to be tailored for the specific task of detecting vehicles visible in a rear-view mirror, optimizing its accuracy and efficiency for real-time applications.

1.5 Video Visualization

In the final step, the trajectories are brought to life through a visual representation overlaid onto a sample reversing camera video.

This visualization dynamically adjusts the trajectories in response to the steering angle. The steering angle can remain constant, providing a straightforward view, or it can be dynamically altered using an interactive slider in a separate video interface. This real-time adjustment of the steering angle results in corresponding updates to the trajectories, enhancing the interactive and dynamic nature of the visualization.