agriculture

MDPI

Article

Efficient Real-Time Row Detection and Navigation Using LaneATT for Greenhouse Environments

Ricardo Navarro Gómez, Joel Milla, Paolo Alfonso Reyes Ramírez, Jesús Arturo Escobedo Cabello * and Alfonso Gómez-Espinosa

Tecnologico de Monterrey, School of Engineering and Sciences, Monterrey 64700, Mexico; a01708825@tec.mx (R.N.G.); a01177727@tec.mx (J.M.); paolo.alfonso.reyes@gmail.com (P.A.R.R.); agomeze@tec.mx (A.G.-E.)

* Correspondence: arturo.escobedo@tec.mx

Abstract

This study introduces an efficient real-time lane detection and navigation system for greenhouse environments, leveraging the LaneATT architecture. Designed for deployment on the Jetson Xavier NX edge computing platform, the system utilizes an RGB camera to enable autonomous navigation in greenhouse rows. From real-world agricultural environments, data were collected and annotated to train the model, achieving 90% accuracy, 91% F1 Score, and an inference speed of 48 ms per frame. The LaneATT-based vision system was trained and validated in greenhouse environments under heterogeneous illumination conditions and across multiple phenological stages of crop development. The navigation system was validated using a commercial skid-steering mobile robot operating within an experimental greenhouse environment under actual operating conditions. The proposed solution minimizes computational overhead, making it highly suitable for deployment on edge devices within resource-constrained environments. Furthermore, experimental results demonstrate robust performance, with precise lane detection and rapid response times on embedded systems.

Keywords: row detection; LaneATT; greenhouse automation; deep learning; real-time robotics; autonomous navigation; edge computing

Academic Editor: Ning Yang

Received: 3 December 2025
Revised: 22 December 2025
Accepted: 29 December 2025
Published: 31 December 2025

1. Introduction

Greenhouse environments play an important role in modern food production by providing controlled climates that reduce the impact of external weather conditions [1]. This ability to produce crops regardless of external weather conditions is a major benefit, helping to ensure a consistent food supply and support agricultural development [2]. Yet, the sector faces challenges, such as persistent labor shortages and escalating costs for farmers and agriculturists [3]. These pressures have made the automation of greenhouse operations desirable [4]. In response, unmanned agricultural robots have emerged as a promising solution, equipped with key capabilities such as navigation, detection, action, and mapping [5]. Of these, navigation stands out as particularly critical, since efficient and autonomous movement is required for robots to effectively operate within the complex layouts typical of greenhouse environments [6].

Building on the need for effective automation in greenhouses, a wide range of studies have investigated methods to achieve autonomous navigation in agricultural settings. Among these, one prominent solution is the use of the Global Navigation Satellite System

(GNSS) combined with Real-Time Kinematic (RTK) technology, which delivers centimeter-level positional accuracy [7,8]. This high precision enables robots to accurately localize themselves and navigate fields [8]. However, greenhouse structures attenuate and reflect satellite signals, often causing multipath, poor satellite visibility, and loss of RTK corrections, which severely degrade accuracy and availability indoors [9]. As a result, many studies have shifted toward local, onboard sensor approaches for autonomous navigation in greenhouses.

Among these local sensor-based navigation approaches, Light Detection and Ranging (LiDAR) has become a popular alternative. LiDAR sensors generate point clouds, which, when combined with algorithms like Simultaneous Localization and Mapping (SLAM), enable robots to map and localize in real-time in unknown environments [10,11]. Despite these advantages, the high cost and complexity of LiDAR-based solutions have led researchers to investigate alternative navigation techniques.

Visual sensors have gained increasing favor in recent research [12]. Their low computational cost and reliability make them particularly advantageous for autonomous navigation tasks [12]. Current computer vision methods utilizing visual sensors can be broadly categorized into two main approaches: traditional methods and deep learning techniques.

Traditional computer vision methods typically use algorithms that rely on manually designed algorithms and mathematical models to process and interpret images, with the aim of extracting navigation lines from visual data. For instance, ref. [13] developed a computationally efficient algorithm that utilizes Otsu thresholding and blob detection with a low-cost 2D camera to identify the center of vineyard rows, enabling collision-free navigation. Similarly, ref. [14] proposed a multi-stage algorithm that segments images, extracts crop row feature points, and uses linear regression to accurately calculate navigation lines, enabling navigation for high-ridge, broad-leaved crops across different growth periods.

In another example, ref. [15] introduced a prediction-point Hough transform algorithm that uses a novel grayscale factor for improved image segmentation and achieves accurate navigation path fitting for greenhouse cucumber-picking robots. Meanwhile, ref. [16] presented a low-cost local motion planner for vineyard navigation, leveraging RGB-D cameras and depth maps for real-time row-end detection as the primary control algorithm.

While the traditional computer vision methods described previously can perform in targeted agricultural scenarios, they tend to rely on carefully tuned parameters or environment-specific cues, which limit their ability to adapt across diverse settings. As a result, their generalizability remains a significant challenge.

To overcome these limitations, recent research has shifted toward deep learning approaches, which are designed to address the generalization problem, although this often comes with increased computational demands. For instance, ref. [4] applies YoloV8 to segment crop-free ridges, which are the areas where the robot will traverse, and then uses the least-squares method (LSM) to obtain the ridge centerline. Similarly, ref. [12] enhances the ENet network to segment rice inter-rows and utilizes the LSM algorithm to obtain the navigation line, achieving a pixel accuracy of 96.39%. Ref. [17] uses a U-Net model to segment rows between trees, then uses a navigation line to traverse with an average mean intersection over union of 0.973. However, these approaches still struggle with common computer vision problems, such as errors under changing illumination.

Other methods, such as refs. [5,18], utilize YOLOV8 to detect tree trunks as reference points to define row limits, then calculate the navigation line, reporting 95% and 92.11% precision, respectively. In addition, ref. [19] introduces a multi-perception network that simultaneously detects tree trunks, obstacles, and traversable areas in a single pass. However, tree trunks may be unreliable landmarks inside greenhouses due to occlusions, crop—stage variability, and crop types lacking prominent trunks; moreover, as noted in [20],

the computational overhead of deep learning can be prohibitive on low-resource systems typical of greenhouse robotics.

To address these persistent challenges, researchers have developed neural network architectures that prioritize lane identification over more complex and heavy compute tasks, such as image segmentation and feature extraction. These specialized neural networks are designed to robustly detect navigation lines under diverse and challenging conditions, including low-light and visually complex environments. While much of the research on lane identification networks has emerged from the self-driving car domain [18], these advances have also found application in adjacent fields, such as autonomous mine exploration, where conventional computer vision methods often fail [21]. Among these, LaneATT stands out as an efficient approach capable of detecting lane anchors and achieving strong alignment with real-world lanes [22]. Despite its success in these domains, the lane identification network LaneATT remains largely unexplored in the agricultural sector.

This study introduces the novel use of lane identification networks, specifically the Real-Time LaneATT network, for autonomous greenhouse navigation. The model achieved 91% F1 Score with the real-time performance of 48 ms per frame on embedded hardware. Experimental validation demonstrates that the robotic platform can maintain its trajectory within a safe lateral deviation from the row center, thereby preventing interference with crops.

This paper is organized as follows: Section 2 presents the LaneATT methodology with its ResNet-18 backbone and anchor-based detection mechanism. Section 3 describes the implementation, including the robotic platform, ROS2 architecture, custom greenhouse dataset preparation, and control algorithms. Section 4 demonstrates results achieving 91% F1 Score and 90% accuracy with real-time inference. Section 5 concludes the work's contributions to autonomous greenhouse navigation.

2. Materials and Methods

2.1. Lane Detection Model: LaneATT

LaneATT, as mentioned in the previous section, is a neural network architecture that utilizes an anchor-based approach to detect lanes in images. LaneATT is built upon a lightweight deep learning architecture, making it suitable for real-time applications such as autonomous navigation. The model's efficiency and accuracy are achieved through a combination of a backbone network, anchor-based detection, and a prediction head [22], as shown in Figure 1.

2.1.1. Backbone Network

LaneATT uses a ResNet-18 as its backbone network to extract features from input images. The ResNet-18 architecture is a convolutional neural network (CNN) that addresses the vanishing gradient problem through residual learning. It is an 18-layer deep convolutional neural network consisting of an initial 7×7 convolutional layer, followed by four stages of residual blocks (each containing two 3×3 convolutional layers with skip connections), and a final fully connected layer. Residual connections enable the network to learn identity mappings, allowing it to train deeper networks without performance degradation in subsequent layers. This ensures that the model captures intricate lane features, even in challenging conditions such as varying lighting and complex road textures [23].

The input to the model is an RGB image $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$, where H and W denote the image height and width, respectively. The backbone network processes the image to generate a feature map $\mathbf{F} \in \mathbb{R}^{H' \times W' \times C}$, where H' and W' are reduced spatial dimensions, and C is the number of channels [23], as shown in the first half of Figure 1 (from left to right). This feature map serves as a compact representation of the input image, retaining

essential lane information while reducing computational complexity. The choice of ResNet-18 achieves a balance between computational efficiency and feature richness, making it suitable for deployment on resource-constrained platforms, such as embedded systems in autonomous vehicles [22].

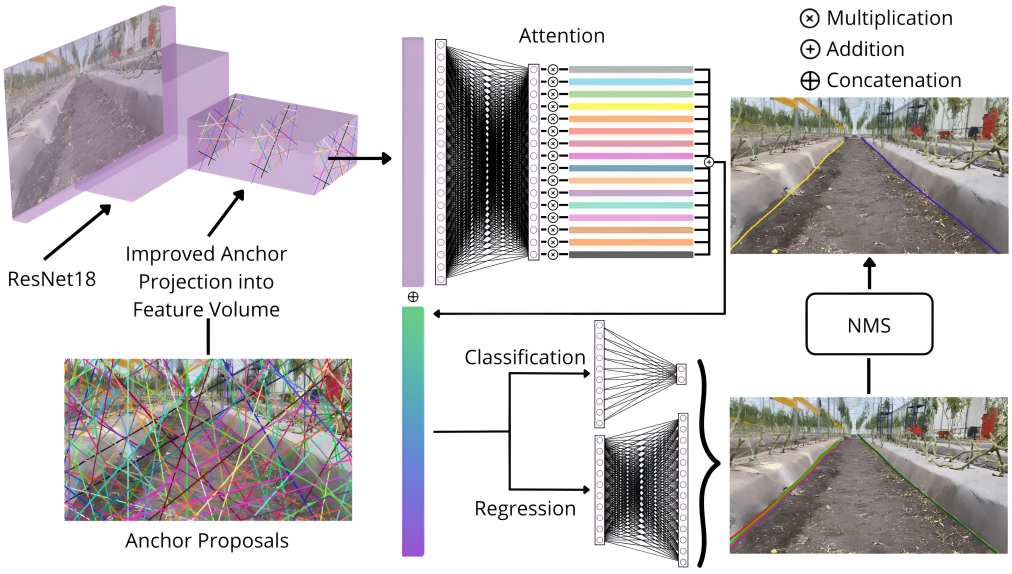

Figure 1. Diagram of LaneATT applied to greenhouses.

2.1.2. Anchor-Based Feature Pooling

LaneATT adopts an anchor-based approach, where predefined anchors represent candidate lane positions in the image. Anchors serve as references for possible lane locations, allowing the model to focus on areas likely to contain lanes. Each anchor is parameterized by its position and orientation in the image, and the model predicts adjustments (or offsets) that refine these anchors to align with detected lanes. This enables LaneATT to detect lanes of varying shapes and perspectives by leveraging the diversity of anchor placements.

2.1.3. Attention, Prediction Head and Loss Function

The prediction head of LaneATT is composed of a series of fully connected layers starting with an attention mechanism that takes the feature map from the backbone as input and outputs lane parameters for each anchor. These parameters include the lane existence probability, offsets for adjusting anchor positions, and class scores that differentiate between lane types (e.g., left lane, right lane, or center lane). By modeling lane detection as a parameter regression problem, LaneATT can predict lane curves, adapting to the varying curvature and width of real-world lanes [24].

The loss function ($\mathcal{L}$) used for training LaneATT is a combination of focal loss ($\mathcal{L}_{\text{focal}}$) and smooth_L1 loss ($\mathcal{L}_{\text{smooth\_L1}}$):

$$\mathcal{L} = \mathcal{L}_{\text{focal}} + \lambda \mathcal{L}_{\text{smooth\_L1}}, \tag{1}$$

where $\mathcal{L}_{\text{focal}}$ is designed to handle the class imbalance between lane and non-lane regions by focusing more on hard-to-classify samples. This is particularly important in lane detection, where lane pixels are often sparse compared to background pixels. The smooth_L1 loss $\mathcal{L}_{\text{smooth\_L1}}$ penalizes the difference between the predicted offsets and the ground-truth values, ensuring that the predicted lanes align accurately with the ground-truth positions. The hyperparameter λ controls the balance between the two loss components, allowing

the model to trade off between classification accuracy and regression precision during training [24].

2.1.4. Inference

During inference, LaneATT processes each input image through the backbone network to generate a feature map. This feature map is then passed through the prediction head, which generates lane existence probabilities, offsets, and class scores for each anchor. The anchor-based detection mechanism uses these predictions to refine lane candidates by adjusting the anchor points based on the offsets and selecting the candidates with the highest existence probabilities [22].

The output of the inference process is a set of lane polynomials, each representing a detected lane in the image space. These polynomials describe the lane shape as a function of pixel coordinates, allowing for the precise localization of lanes in various driving environments.

2.2. Non-Maximum Suppression (NMS) Filter

Non-Maximum Suppression (NMS) is the post-processing step used in LaneATT to refine lane predictions by eliminating redundant detections. Given the anchor-based nature of LaneATT, multiple anchors may predict similar lanes with varying confidence scores. Without NMS, this overlap could lead to cluttered predictions, complicating downstream tasks such as navigation.

The NMS process begins by sorting all lane predictions based on their confidence scores, which are derived from the lane existence probabilities predicted by the model. For the most confident prediction, its Intersection over Union (IoU) with all other lanes is computed as follows:

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|}, \tag{2}$$

where A and B represent the regions associated with two lane predictions. Any predictions with an IoU greater than a predefined threshold θ (e.g., $\theta = 0.5$) are considered redundant and discarded. This process iterates until all valid lanes are selected, ensuring that only the most confident, non-overlapping lanes are retained.

NMS enhances LaneATT's robustness by reducing false positives and ensuring that the final outputs represent distinct lane curves.

2.3. Evaluation Metrics

To assess the performance of LaneATT, the following metrics are used: F1 Score, Precision, Recall, Accuracy, and Runtime. These metrics collectively evaluate the model's predictive capabilities and computational efficiency.

2.3.1. Precision

Precision measures the fraction of correctly predicted lanes among all predicted lanes. It is defined as

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}, \tag{3}$$

where True Positives represent correctly identified lanes, and False Positives denote incorrectly predicted lanes.

2.3.2. Recall

Recall evaluates the fraction of correctly predicted lanes among all ground-truth lanes. It is computed as

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}, \tag{4}$$

where False Negatives are ground-truth lanes missed by the model. High recall indicates the model's effectiveness in capturing all relevant lanes.

2.3.3. F1 Score

The F1 Score provides a harmonic mean of Precision and Recall, offering an assessment of the model's performance. It is calculated as

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{5}$$

This metric is important for handling imbalanced datasets, where precision and recall alone may provide incomplete insights.

2.3.4. Accuracy

Accuracy measures the overall proportion of correct predictions, defined as follows:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Predictions}}, \tag{6}$$

where True Negatives represent the absence of false lane detections. This metric is a straightforward indicator of the model's reliability across the dataset.

2.3.5. Average Yaw Angle Error

The average yaw angle error quantifies the angular deviation between predicted and ground-truth navigation paths. For each image, both the predicted navigation line and ground-truth navigation line are represented as linear equations using the least-squares method:

$$y = k \cdot x + b \tag{7}$$

$$y' = k' \cdot x + b' \tag{8}$$

where k and k' are the slopes, and b and b' are the intercepts of the predicted and ground-truth lines, respectively. The corresponding yaw angles are calculated as

$$\theta = \tan^{-1}(k), \quad \theta' = \tan^{-1}(k') \tag{9}$$

The average yaw angle error across the dataset is then computed as follows:

$$\text{Mean Yaw Error} = \frac{1}{n} \sum_{i=1}^{n} |\theta_i - \theta_i'| \tag{10}$$

where n represents the total number of images, θ_i is the predicted yaw angle, and θ_i' is the ground-truth yaw angle for image i. This metric provides a measure of the model's angular prediction accuracy across the entire test set.

2.3.6. Runtime

Runtime quantifies the computational efficiency of the model, measured as the average time required to process a single image. This metric is critical for real-time applications.

2.4. Differential Drive Kinematics

Motion control employs a differential-drive approximation for the skiddrive kinematics, which relates the robot's linear and angular velocities to the individual wheel velocities. Assuming all wheels on the left side move at the same angular velocity ω_l and all wheels on the right side move at the same angular velocity ω_r, with wheelbase length L and wheel radius r being constants (see Figure 2), the linear velocity v and angular velocity ω of the center of mass G can be derived as follows [25]:

$$v = \frac{r}{2}(\omega_l + \omega_r) \tag{11}$$

$$\omega = \frac{r}{L}(\omega_r - \omega_l) \tag{12}$$

The world reference frame $\{W\}$ is denoted by $\{\hat{x}_W, \hat{y}_W\}$ and the robot reference frame $\{R\}$ by $\{\hat{x}_R, \hat{y}_R\}$.

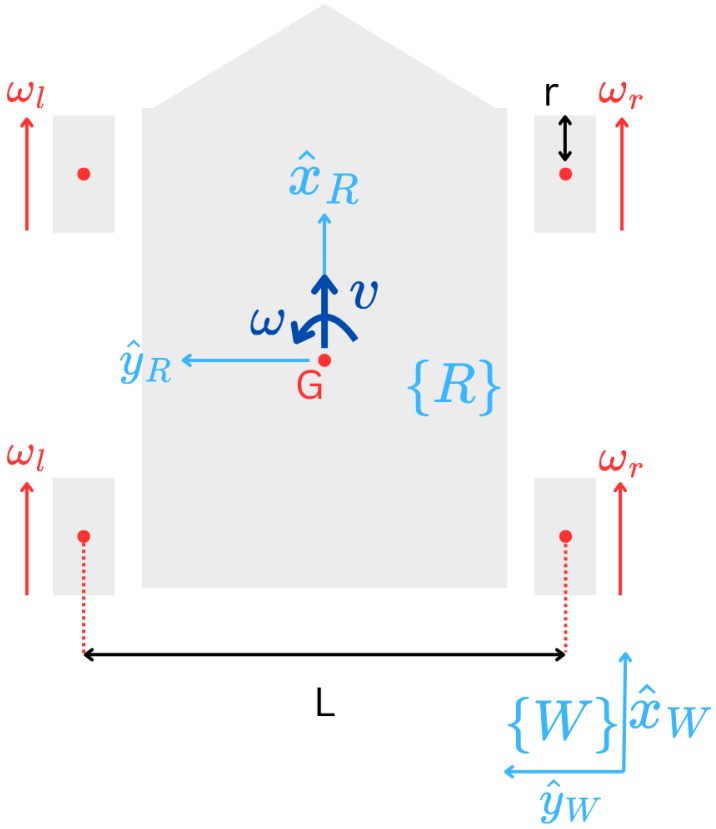

Figure 2. Differential-drive approximation of the skid-steer kinematics, showing the robot's geometric parameters and velocity components. The robot has four wheels (two left, two right) separated by distance L, each with radius r. Point G represents the center of mass with linear velocity v and angular velocity ω.

3. Implementation

3.1. Greenhouse

The research scenario depicted in Figure 3 is the tomato greenhouse at CAETEC (Tec de Monterrey Experimental Agricultural Field), which serves as the site for all of the experimental trials. Within this greenhouse, multiple narrow cultivation rows are arranged with white plastic mulch covering the soil, a common practice for moisture retention. Notably, each row measures an average of 104 cm in width, and the floor is characterized by loose, uneven soil. The experimental trials were conducted exclusively on straight row configurations with north–south orientation, as this layout represents the conventional architecture

for indoor greenhouse tomato cultivation, where space optimization and irrigation efficiency favor linear row arrangements [26]. However, the LaneATT architecture has already demonstrated robust performance in handling curved lanes within the autonomous driving domain [24], suggesting that the model's capabilities can generalize to curved agricultural paths in future outdoor orchard applications.

Figure 3. A greenhouse row from CAETEC. The image shows the scenario where the present work was tested.

3.2. Robotic Platform

The robotic platform used in this research is the Jackal UGV from Clearpath Robotics (Kitchener, ON, Canada), a skid-steer mobile robot designed for field robotics research (see "a" in Figure 4). While the Jackal employs a skid-steer drive configuration, its kinematics can be approximated as a differential drive to simplify the control strategy, which is explained in Section 2.4.

Figure 4. Robotic platform: "a" Jackal Robot from Clearpath Robotics; "b" Gimbal Feiyu Mini 2; "c" Camera Intel RealSense D435.

The Jackal features external dimensions of 508 mm in length, 430 mm in width, and 250 mm in height. All robot control and image processing were performed directly on the platform using Robot Operating System (ROS2 Humble).

For visual input, an Intel RealSense D435 depth camera (Intel Corporation, Santa Clara, CA, USA) was mounted on the system ("c" in Figure 4). D435 features active stereoscopic depth technology with a depth resolution of 1280 × 720 at 15 frames per second. The

camera was mounted on a Feiyu Mini 2 gimbal from FeiyuTech (Guilin, China), which was positioned atop the Jackal (see "b" in Figure 4). The gimbal features 3-axis stabilization and measures 210.1 × 133.7 × 296 mm in its balanced position. The gimbal was configured so the yaw remains unstabilized, allowing the camera to rotate freely for navigation, while the roll and pitch axes were actively stabilized to minimize image degradation caused by motion and to ensure high-quality image acquisition. The Jetson Xavier NX Developer Kit (NVIDIA Corporation, Santa Clara, CA, USA) of 8 GB used has an NVIDIA Volta GPU with 384 CUDA cores and 48 Tensor cores, a 6-core Carmel ARM v8.2 CPU, and 8 GB of LPDDR4x memory. The board interfaces with the RealSense camera via USB 3.1.

3.3. Software

For the deployment on the Jetson Xavier NX Developer Kit, ROS2 Humble was chosen as the middleware due to its compatibility with Ubuntu 22.04, which is the default operating system for the Jetson platform. The system's core functionality was implemented using both rclcpp 16.0.17 (C++) and rclpy 3.2.1 (Python) to leverage the strengths of each language in different modules. The essential ROS2 packages required for the software are geometry_msgs, for handling velocity and pose messages, and cv_bridge, which facilitates the conversion between ROS2 image messages and OpenCV image formats. OpenCV version 4.12 was used throughout the image processing pipeline. The software operates by continuously running the trained lane detection model, performing inference on images captured from the onboard camera. The detected lane centerline is then extracted and used in a control loop to guide the robot's navigation within the greenhouse environment.

Figure 5 illustrates the ROS node architecture where the Image Publisher node acquires frames from the Intel RealSense D435 at 1280 × 720 resolution and 15 FPS, the Lane Detector node processes these frames to predict lane boundaries, and the Velocity Controller node computes and publishes linear and angular velocities to actuate the robot.

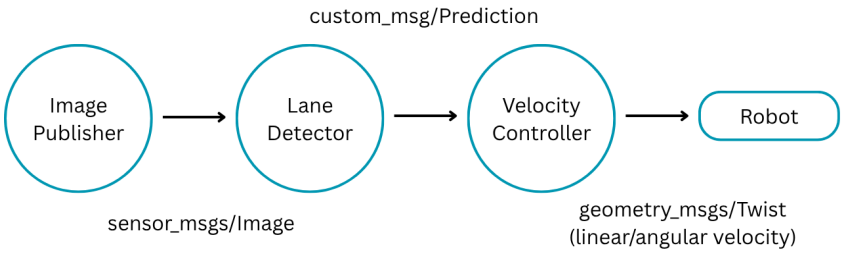

Figure 5. ROS node architecture showing the data flow from image acquisition through lane detection to velocity control.

3.4. Framework and Environment

The implementation of LaneATT was carried out using the PyTorch deep learning framework. The model was trained and evaluated on a workstation equipped with an NVIDIA RTX 3070 Ti GPU, 8 GB of VRAM, and a system running Ubuntu 22.04 with CUDA version 12.6. For reproducibility, all experiments were conducted using the same software environment, comprising Python 3.12.3, PyTorch 2.5, and other necessary libraries, such as OpenCV and NumPy 2.1.2. To aim the research question, the network runtime was also tested in the Jetson Xaver NX (Nvidia Corporation, Santa Clara, CA, USA), and in a 12th Gen Intel i5.

The dataset preprocessing, model training, and evaluation were automated using Python scripts, with a series of logs across all the processes, providing a way to manage the experimental workflow.

3.5. Dataset Preparation

The TuSimple dataset [27] was utilized, which contains images from various driving scenarios, including urban streets, highways, and rural roads. The dataset consists of over 6408 annotated images with corresponding lane markings, making it suitable for training deep learning models for lane detection. Each image in the dataset is accompanied by a label file that contains pixel-wise lane annotations, which define the lane locations within the image.

The training set consisted of 3626 images, while the validation set comprised 358 images, with an additional 2782 images reserved for testing. The training images were normalized using the mean and standard deviation calculated from the ImageNet dataset to maintain consistency with the ResNet-18 backbone, which was pre-trained on ImageNet.

Although the TuSimple dataset is based on various driving scenarios, which differ significantly from the current application (greenhouse navigation), it can still be used for pre-training the network. Pre-training on TuSimple enables the network to learn visual features such as edge detection, line recognition, and spatial relationships that are transferable across domains. These low-to-mid-level features learned from detecting road lanes provide a foundation that can be fine-tuned for detecting navigation paths in greenhouses, resulting in faster convergence and better performance than training from scratch with limited greenhouse data alone. The large size of the dataset allows for the effective initialization of the network before fine-tuning it with the dataset prepared.

In addition to the TuSimple dataset, a custom dataset [28] was prepared in a greenhouse scenario, following the TuSimple annotation format. This choice was made because the network's dataloader is compatible with datasets in this format. To create the dataset, videos were recorded over a one-year period by mounting the camera atop the Jackal while navigating through the greenhouse at three distinct cultivation stages: during initial tomato growth, at harvest maturity, and post-harvest conditions. These videos were then divided into multiple frames, which were then labeled to indicate where the lanes should be. The two different datasets used to train the model can be seen in Figure 6, which shows the different cultivation stages of the greenhouse captured in the custom dataset.

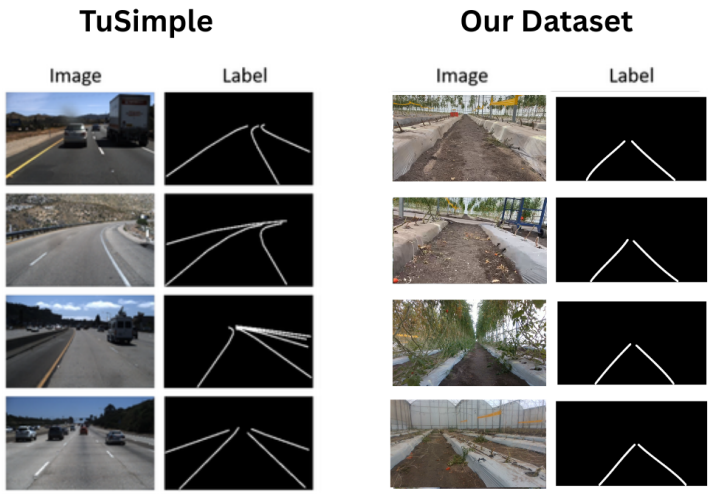

Figure 6. The two datasets used for model training: TuSimple (**left**) featuring highway scenes with lane markings, and our custom dataset (**right**) containing path images with corresponding lane annotations.

Ultimately, the dataset used to train the model comprises 3872 greenhouse images from CAETEC. The resolution of the images is 1280 × 720. The dataset consists of 3171 images for

training, 501 for validation, and 200 for testing, as shown in Figure 7. All images were captured exclusively during daylight conditions, as the system operates under the assumption that the robotic platform will navigate the greenhouse during daytime hours when natural lighting is available. It is important to note that all images were captured from a single greenhouse environment (Figure 3); therefore, to apply this approach to different greenhouses, new images should be collected, and the model should be fine-tuned accordingly.

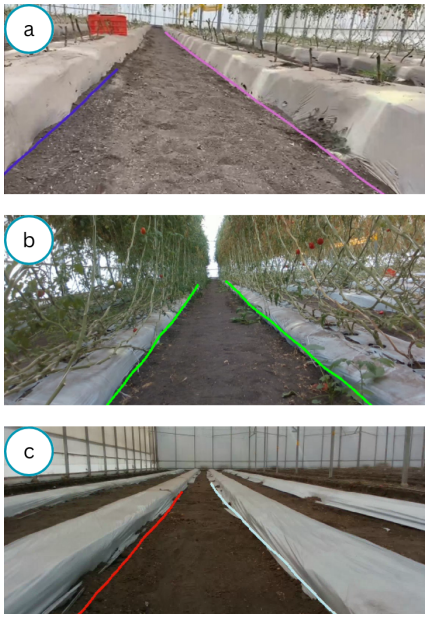

Figure 7. Example of greenhouse dataset captured at CAETEC showing lane detection (colored lines) across different cultivation stages: (**a**) lanes detected during the initial tomato growth phase, (**b**) lanes identified when tomatoes reached harvest maturity, and (**c**) post-harvest greenhouse conditions with cleared rows.

3.6. Model Training

The LaneATT model was initialized with weights from a ResNet-18 backbone pretrained on ImageNet, which provided a strong starting point for lane detection. The training process was carried out using the Adam optimizer with an initial learning rate of 1×10^{-4} and a weight decay of 5×10^{-4}. A learning rate scheduler was employed, reducing the learning rate by a factor of 0.1 every 5 epochs to ensure stable convergence. The total number of epochs was set to 100, and a batch size of 8 was used for training.

The loss function was focal loss, as described in Section 2.1.3. To further enhance the model's performance, a set of anchor points was carefully chosen to cover a wide range of lane shapes and positions in the image. During training, the model learned to adjust these anchors to fit the ground-truth lane annotations.

The training process was monitored using validation loss, F1 Score, Precision, Recall, and Accuracy as evaluation metrics. Model checkpoints were saved at each epoch, enabling systematic monitoring and recovery. The final model was selected based on the epoch with the lowest validation loss.

An important thing to note is that the projection of the anchors was modified to preserve the angles of the original image. Unlike the original LaneATT implementation, which distorted the angles when projecting anchors due to the mismatch in scale between the original image and the height and width of the activation volume, the method ensures that the geometric relationships of the anchors remain consistent. This adjustment enhances the alignment between the predicted anchors and the actual lanes, particularly in scenarios

where maintaining angular consistency is crucial, enabling the network to achieve high results in fewer epochs.

3.7. Inference and Post-Processing

During inference, the trained LaneATT model takes an RGB image as input and outputs a set of lane predictions. For each image, the model produces lane existence probabilities, anchor offsets, and lane parameters. These parameters were used to compute the lane position and curvature in image space. Since lane lines can overlap in complex driving scenarios, non-maximum suppression (NMS) was applied to the output to remove duplicate lane predictions, ensuring that only the most confident lanes are retained.

The lane predictions were then transformed into polynomials, representing the lane curves in pixel coordinates. These curves were overlaid on the original image to visualize the detected lanes.

For real-time applications, the model was optimized using TorchScript 2.5.4, allowing the inference to be deployed on embedded systems. This optimization enabled fast inference times, achieving around 30 frames per second (FPS) on various systems, as detailed in the Section 4, making the system suitable for real-time lane detection in autonomous vehicles.

3.8. Control

A proportional control strategy is employed, where the following sections describe how linear and angular velocities are obtained from the model outputs.

3.8.1. Calculation of Lane Point

The control strategy for calculating the centerline proceeds as follows: First, the trained neural network performs inference on each input image to detect lane points corresponding to the left and right lanes, which are then stored in separate arrays (see Figure 8a). Next, the midpoints between the two detected lanes are computed. From these midpoints, the first x points are selected starting from the bottom of the image upward, representing the region closest to the robot (Figure 8b). The average of these selected points yields the reference lane point for navigation control, calculated as follows:

$$(X_p, Y_p) = \left(\frac{1}{n} \sum_{i=1}^{n} x_i, \frac{1}{n} \sum_{i=1}^{n} y_i \right) \tag{13}$$

where n is the number of detected midpoints, and (x_i, y_i) represents each individual midpoint ("a" in Figure 9). Each point is defined in the image coordinate system, as shown in Figure 9. This averaging approach prioritizes immediate navigation decisions by focusing on the region closest to the robot while simultaneously filtering potential noise from discontinuous or deformed lane predictions that may arise during inference. With this approach, a higher number of detected points indicates greater model confidence in the lane structure.

3.8.2. Calculation of Linear and Angular Velocity

Inspired by [16], the control approach uses the calculated lane point to generate real-time navigation commands. First, the angle θ between the image center and the reference lane point is computed using

$$\theta = \tan^{-1}(Y_p - Y_c, X_p - X_c) \tag{14}$$

where (X_c, Y_c) represents the horizontal center at the bottom of the image, and (X_p, Y_p) is the computed lane point with $\theta \in [0, \pi]$ as seen in Figure 8c. The angular error E_θ is then obtained by subtracting $\pi/2$.

$$E_\theta = \theta - \frac{\pi}{2} \tag{15}$$

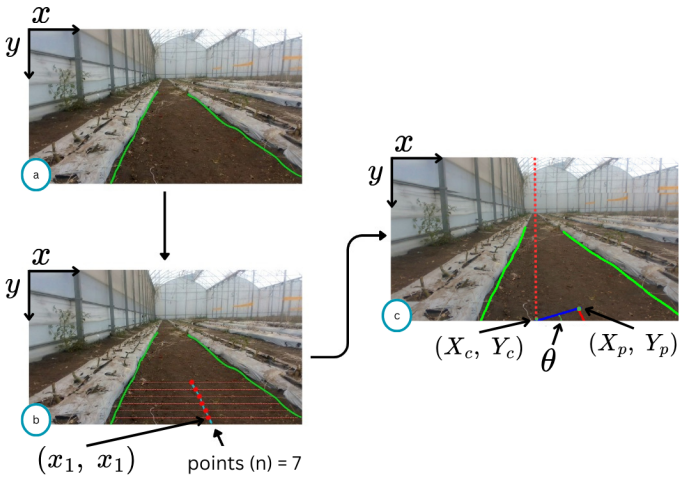

Figure 8. Lane detection and error calculation pipeline: (**a**) inference that detects left and right green lanes, (**b**) computation of the first 7 midpoints from bottom upward, and (**c**) calculation of the angle θ.

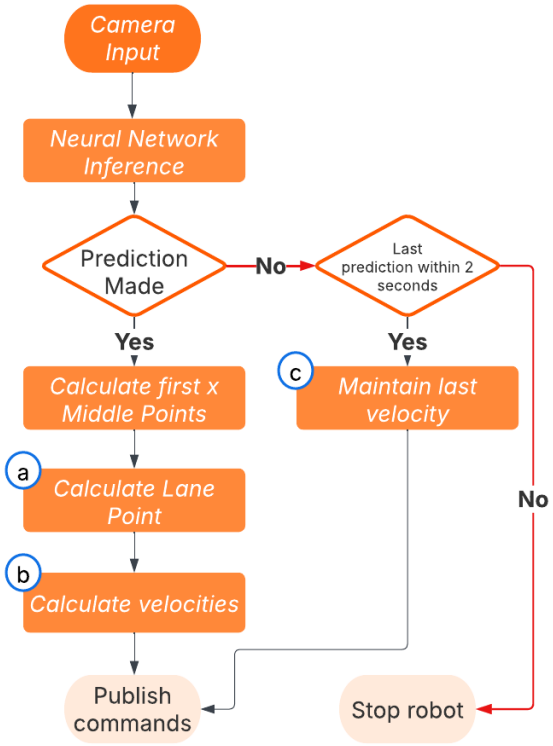

Figure 9. Lane detection and control pipeline with a fail-safe mechanism, with "a", "b", and "c" being the main steps of the algorithm.

This transformation ensures that $E_\theta \in [-\pi/2, \pi/2]$, where $E_\theta = 0$ indicates the robot is aligned with the lane centerline.

Linear velocity v is calculated based on the number of detected midpoints n relative to the maximum expected points x:

$$v = \frac{k_v \cdot n}{x} \cdot \text{max\_lin\_vel} \tag{16}$$

where max_lin_vel is the maximum linear speed, and k_v is a proportional constant selected empirically. This formulation allows the robot to move at maximum speed when the model detects all expected points (high confidence), and reduces speed when fewer points are detected (lower confidence), prioritizing safer navigation during uncertain conditions.

The angular speed is computed using the proportional constant k_ω:

$$\omega = k_\omega \cdot \text{max\_ang\_vel} \cdot E_\theta \tag{17}$$

where max_ang_vel is the maximum angular speed, and k_ω is a proportional constant selected empirically. These proportionality constants are tuned based on the navigation error and desired system response (see "b" in Figure 9). To enhance robustness, when no lane prediction is detected, the system maintains the previous control commands for up to two seconds. If no valid prediction is received for more than two seconds, such as during temporary occlusions or other failure scenarios, the robot is brought to a complete stop to ensure safe operation ("c" in Figure 9).

3.9. Experimental Setup

To evaluate the performance of the algorithm, tests were conducted in three different greenhouse rows. For each row, a 6 m straight path was designated with an ArUco marker placed at the 6 m endpoint. This setup allowed for consistent distance measurement and validation across all experimental trials. To ensure precise marker alignment, the row centerline was measured at each 0.1 s interval along the 6 m path, and the ArUco marker was positioned at the averaged centerline position. The alignment accuracy was verified by manually positioning the robotic platform at the marker location and confirming that the validation node reported lateral displacement within the acceptable range of [−3 cm, 3 cm].

Three initial positioning conditions were tested for each row to assess the system's ability to handle different starting scenarios and field-of-view configurations. The first condition placed the robot centered in the row with the camera facing directly forward, representing ideal alignment. The second condition positioned the robot 10 cm from the left side of the row, resulting in an off-center field of view. The third condition placed the robot 10 cm from the right side of the row, similarly challenging the system with a laterally shifted perspective. These varied starting positions were designed to evaluate the control system's performance in correcting both lateral positioning errors and maintaining consistent navigation under non-ideal camera viewing angles. Although the model was trained across different cultivation stages, the tests were only conducted under post-harvest greenhouse conditions with cleared rows, as shown in Figure 7. However, the model performance metrics and average yaw angle error were evaluated across all cultivation stages using the complete test dataset.

3.10. Parameter Tuning

According to [5], sprayers and harvesters achieve velocities between 0.44 m/s and 1.39 m/s in agricultural operations. Since the primary applications for autonomous navigation in greenhouses include monitoring and harvesting tasks, a maximum linear velocity of 0.7 m/s was selected to fall within this operational range.

For the expected points parameter (x), 7 was selected based on the model's prediction capabilities and navigation requirements. The LaneATT model can generate up to 72 lane points representing the complete path from the robot to the end of the visible lane. By selecting the first 7 points (approximately 10% of the total predictions), the system focuses on the immediate path region, which corresponds to approximately 1.6 m ahead of the robot. Given the maximum linear velocity of 0.7 m/s, this provides the robot with approximately 2 s of lookahead time for trajectory adjustments.

Finally, the timeout threshold of 2 s was selected based on navigation safety and model performance considerations. At 0.7 m/s, the robot travels approximately 1.4 m during this interval, which remains within the consistent row structure of the greenhouse, where previous predictions maintain validity. This duration was selected considering two constraints: it prevents excessive unguided movement (the robot maintains previous commands when the model outputs zero predictions), which could compromise navigation safety, while providing sufficient time for the model to recover from areas with challenging visual conditions where no lane predictions are generated.

The final parameters used for all the tests are shown in Table 1.

Table 1. Control parameters used in the experiments.

Parameter	Value
max_lin_vel	0.7 m/s
max_ang_vel	0.5 rad/s
x (expected points)	7
k_v	0.8
k_ω	0.5
Timeout threshold	2 s

3.11. Validation Node

To track the performance of the algorithm, a validation node was implemented using ArUco marker detection. The ArUco marker, positioned at 6 m in front of the robot, served as a reference point for measuring the lateral error of the robot with respect to the row centerline.

The validation process operates as follows: First, the system detects the ArUco marker and computes the transformation matrix representing the pose of the marker with respect to the robot's camera T_{marker}^{camera} as indicated in Equation (18):

$$T_{marker}^{camera} = \begin{bmatrix} R_{marker}^{camera} & t_{marker}^{camera} \\ 0 & 1 \end{bmatrix} \tag{18}$$

where R_{marker}^{camera} is the rotation matrix, and t_{marker}^{camera} is the translation vector. From this, the inverse transformation provides the camera pose with respect to the marker frame:

$$T_{camera}^{marker} = (T_{marker}^{camera})^{-1} \tag{19}$$

Since the ArUco marker is positioned centered in the row, the deviation from the centerline can be measured by analyzing the z-axis displacement in the marker coordinate frame. As shown in Figure 10, this z-axis component represents the lateral displacement (in centimeters) of the robot from the row center, providing a direct quantitative measure of centering error throughout navigation. The recorded data were then used to analyze how the robot followed the track, enabling visualization of the corrections made by the navigation system over time.

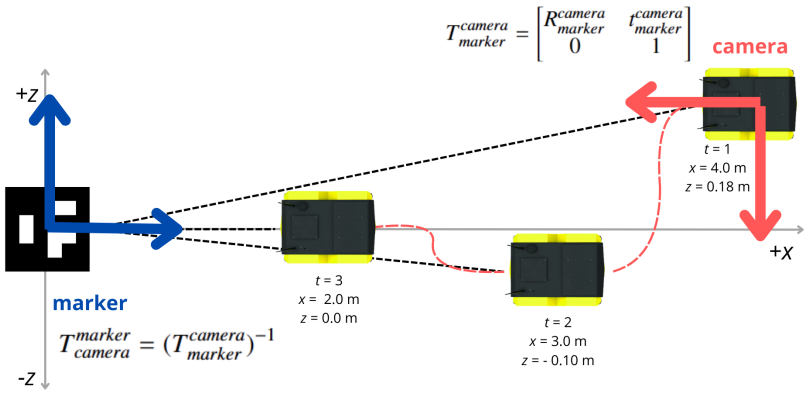

Figure 10. Validation setup with ArUco marker positioned at the row centerline and the robot's trajectory at different time steps, where x represents the distance between the ArUco marker and the robot, and z indicates deviation from the centerline.

4. Results

The quantitative performance of LaneATT was assessed on a benchmark lane detection dataset created from the custom dataset. For obtaining the metrics described in Figure 11 and Table 2, the model had an evaluation mode, in which, using the validation dataset, it would calculate all the metrics with the previously mentioned methods.

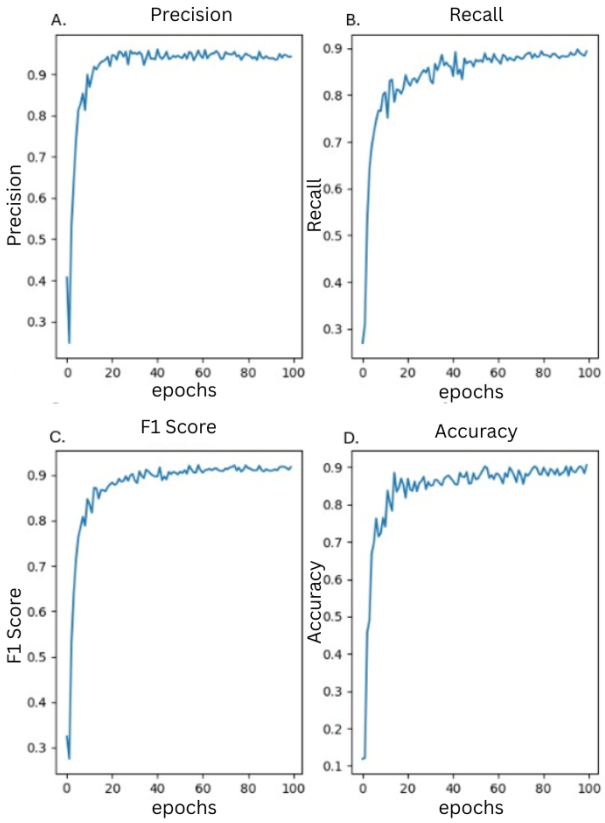

Figure 11. Precision (**A**), recall (**B**), F1 Score (**C**), and accuracy (**D**) evaluated on the validation dataset over successive epochs.

For this, the evaluation mode was run after each epoch to track the performance history of the project throughout the entire training period. Table 2 summarizes the final results in terms of precision, recall, F1 Score, accuracy, and runtime of the model trained by 100 epochs across the different cultivation stages.

Table 2. Performance of LaneATT on the benchmark dataset.

Metric	Value
Precision	0.94
Recall	0.89
F1 Score	0.91
Accuracy	0.90
Runtime (Nvidia RTX3070Ti GPU)	11 ms per frame
Runtime (12th Gen Intel i5)	32 ms per frame
NVIDIA Jetson Xavier	48 ms per frame

The high precision and recall values demonstrate the model's ability to accurately identify lanes while minimizing false positives and false negatives. The F1 Score of 0.91 indicates a strong balance between precision and recall, while the 90% accuracy highlights the reliability of predictions across diverse scenarios. Moreover, the inference speeds obtained demonstrate that the model can process frames fast enough to enable autonomous navigation in the greenhouse.

Apart from the dataset, LaneATT was also tested in real-world environments using the Jackal Robot. Figure 7 illustrates examples of lane detection in a greenhouse setting under varying conditions, such as uneven lighting and partial occlusions. The model consistently identified lane boundaries. Furthermore, as shown in Table 3, the model achieved an average yaw angle error of 1.67°, demonstrating performance comparable to current state-of-the-art navigation methods.

Table 3. Comparison of average yaw-angle error for the proposed navigation approach versus existing methods.

	Average Yaw Angle Error (°)
Real-time Row Detection	1.67
[17]	0.397
[15]	1.91

Using the validation node described in Section 3.11, ten experimental runs were conducted along different greenhouse rows to evaluate the navigation accuracy and network performance under real operating conditions. For each trajectory, the lateral error was defined as the displacement of the robot with respect to the centerline of the row. Representative trajectories are illustrated in Figure 12.

For each run, the mean lateral error and its standard deviation were computed to assess systematic bias and lateral oscillations, respectively, as seen in Table 4. Additionally, the root mean square (RMS) lateral error was calculated for each trajectory to provide an overall accuracy metric that accounts for both bias and variability.

Across all trajectories, the mean RMS lateral error was 0.133 m with a standard deviation of 0.048 m. Considering an average greenhouse row width of 1.04 m, this corresponds to a normalized RMS error of $12.8 \pm 4.6\%$ of the row width. The maximum normalized RMS error observed among all runs was 17.4%, remaining below the predefined safety threshold of 20% of the row width (0.208 m).

The mean lateral errors exhibited alternating signs and small magnitudes, indicating the absence of systematic drift toward either side of the row. Although lateral variability was observed due to real-world disturbances and sensing noise, the bounded RMS error across all experiments demonstrates that the robot consistently remained within the crop corridor, validating the practical suitability of the proposed navigation system for autonomous greenhouse operation.

Table 4. Lateral navigation-error metrics for each experiment, with RMS errors provided in meters and as percentages relative to the row width (1.04 m).

Run	Mean Error (m)	STD (m)	RMS (m)	RMS (% Row Width)
1	0.040	0.125	0.131	12.7
2	0.041	0.128	0.134	13.0
3	0.090	0.129	0.158	15.2
4	0.005	0.115	0.115	11.1
5	0.021	0.086	0.089	8.6
6	−0.013	0.017	0.021	2.1
7	−0.008	0.169	0.169	16.3
8	0.054	0.150	0.160	15.4
9	−0.007	0.180	0.180	17.4
10	−0.013	0.165	0.166	16.0
Mean ± Std	—	—	0.133 ± 0.048	12.8 ± 4.6
Max	—	—	0.180	17.4

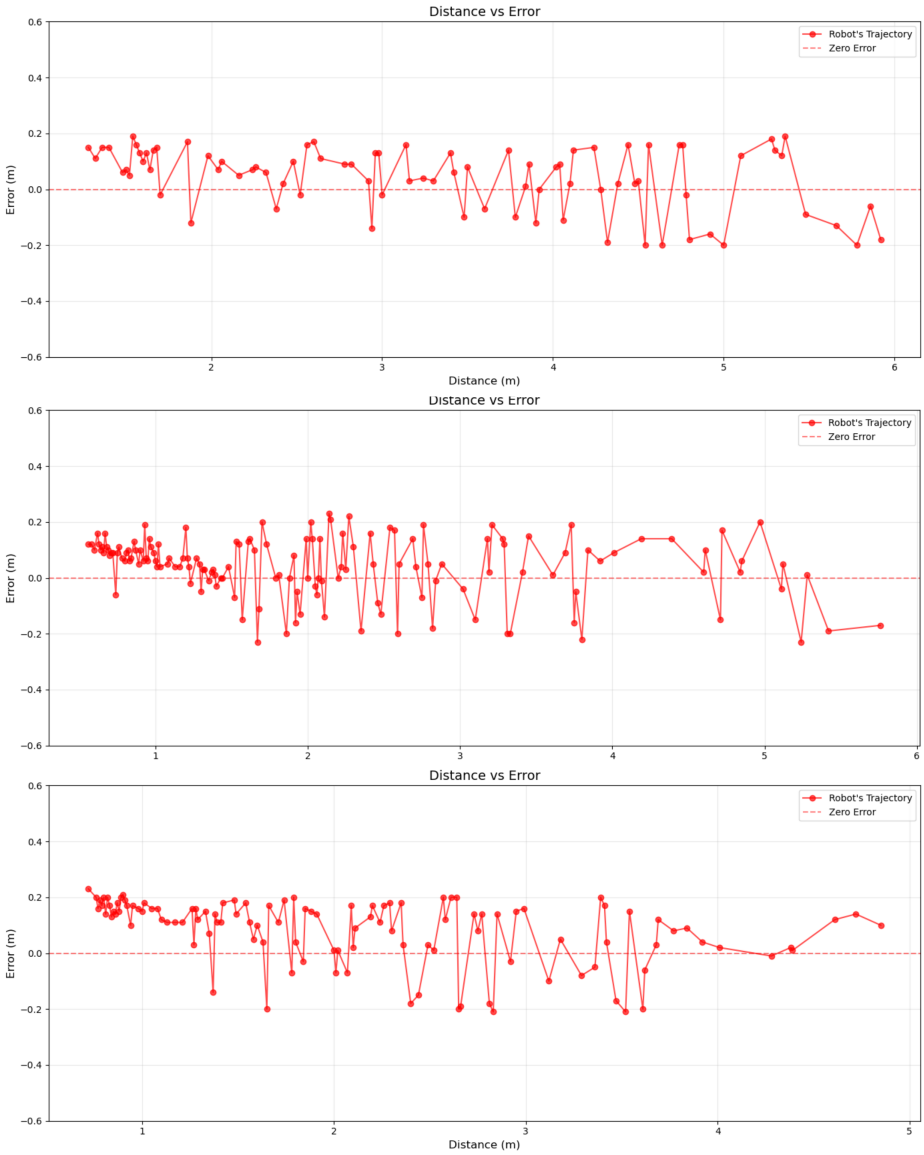

Figure 12. Recorded trajectory from a trial carried out in a greenhouse row. The figure plots distance (m) versus lateral error (m).

At the same time, the runtime analysis revealed that LaneATT operates in real-time on both high-performance GPUs and embedded devices. The model achieved an average processing time of 11 ms per frame on an NVIDIA RTX GPU, 32 ms per frame on a 12th Gen Intel i5, and 48 ms per frame on NVIDIA Jetson Xavier. These results confirm the feasibility of deploying LaneATT for real-time applications on edge devices.

Apart from the previously mentioned results, in Figure 11, it can be seen how the majority of the metrics, in only a few epochs, achieved a high value, while the recall (Figure 11) required more epochs to surpass 85%.

The results demonstrate that LaneATT delivers accurate lane detection in both controlled and real-world scenarios. The model's high precision and recall validate its robustness, while the runtime results affirm its suitability for embedded deployment. These results validate the model's ability to handle diverse lane types and challenging conditions, such as varying lighting and occlusions, without significant performance degradation. As shown in Table 5, the proposed model achieves a state-of-the-art precision of 0.94.

Table 5. Precision comparison of navigation against the presented model (Real-time LaneATT).

	Precision
Real-time LaneATT	0.94
[4]	0.90
[12]	0.96
[17]	0.95

Additionally, as shown in Table 6, the model's compact size of 13.5 M parameters represents an improvement over current state-of-the-art segmentation and object detection approaches. This is the result of LaneATT's lightweight anchor-based architecture. Finally, LaneATT handled challenges such as varying lighting, occlusions, and complex lane geometries, further showcasing its reliability in practical settings.

Table 6. Comparison of model sizes between the proposed method (Real-time LaneATT) versus state-of-the-art approaches.

	Model	**Parameters**
Real-time LaneATT	LaneATT	13.5 M
[19]	Custom YOLOP	14 M
[5]	YOLOv8m vine classes	25.8 M

5. Conclusions

In this work, an implementation and evaluation of the LaneATT model for lane detection was presented, with a focus on its application in greenhouse environments.

Quantitative evaluation on a benchmark dataset highlighted the model's robustness, achieving an F1 Score of 0.91, accuracy of 90%, and real-time inference speeds of 18 ms per frame on a high-performance GPU and 100 ms per frame on a low-performance GPU. The results demonstrate that LaneATT balances high accuracy, precision, and recall while maintaining computational efficiency.

LaneATT's implementation with only 13.5 M parameters enabled real-time network execution on edge computing devices, providing enough inference speed for autonomous navigation and maintaining the capability for direct deployment on edge devices in agricultural environments.

Another thing to note is the ability of the network to achieve high metrics in just a few epochs; being this achievement mainly by the change of the anchor proposal system,

searching to project the anchors on the activation module, maintaining the angles in critical scenarios. This can be seen in Figure 11, where in less than 20 epochs, the model achieved an F1 Score, accuracy, and precision above 85.

Qualitative results from real-world tests in greenhouse environments further demonstrated LaneATT's capability to identify lane boundaries accurately, enabling efficient navigation. This validates the model's reliability in practical autonomous navigation tasks.

For future work, it is important to note that since the current research focused on validating LaneATT as an efficient alternative to segmentation and object detection models, the development of a specialized control system for dynamic obstacle handling in complex environments remains as a future improvement. Furthermore, while the tests were conducted on the linear row configurations typical of conventional indoor greenhouses, future iterations should validate the model's performance on curved corridors, junctions, and irregular widths, either through simulated environments or by gaining access to greenhouses with more complex layouts than those currently available. Finally, to ensure the system's robustness across diverse agricultural settings, further data collection and training in multiple greenhouses will be conducted to generalize the model's capabilities.

Author Contributions: Conceptualization, R.N.G., P.A.R.R., J.M. and J.A.E.C.; methodology, R.N.G., P.A.R.R., J.M. and J.A.E.C.; software, R.N.G., P.A.R.R. and J.M.; validation, R.N.G. and J.M.; formal analysis, R.N.G., P.A.R.R., J.M. and J.A.E.C.; investigation, R.N.G., P.A.R.R. and J.M.; resources, J.A.E.C.; data curation, R.N.G. and J.M.; writing—original draft preparation, R.N.G., P.A.R.R. and J.M.; writing—review and editing, R.N.G., J.M., J.A.E.C. and A.G.-E.; visualization, R.N.G. and J.M.; supervision, J.A.E.C. and A.G.-E.; project administration, J.A.E.C. and A.G.-E.; funding acquisition, J.A.E.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data presented in this study are available upon request from the corresponding author.

Acknowledgments: The authors would like to acknowledge the financial support from SECIHTI for the Ph.D. studies of the first author and CAETEC for allowing their greenhouse to be used for testing.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. El-Gayar, S.; Negm, A.; Abdrabbo, M. *Greenhouse Operation and Management in Egypt*; Springer International Publishing: Cham, Switzerland, 2019. [CrossRef]
2. Yu, J.; Sun, C.; Zhao, J.; Ma, L.; Zheng, W.; Xie, Q.; Wei, X. Prediction and control of greenhouse temperature: Methods, applications, and future directions. *Comput. Electron. Agric.* **2025**, *237*, 110603. [CrossRef]
3. Ray, D.K.; Mueller, N.D.; West, P.C.; Foley, J.A. Yield Trends Are Insufficient to Double Global Crop Production by 2050. *PLoS ONE* **2013**, *8*, e66428. [CrossRef] [PubMed]
4. Lv, R.; Hu, J.; Zhang, T.; Chen, X.; Liu, W. Crop-Free-Ridge Navigation Line Recognition Based on the Lightweight Structure Improvement of YOLOv8. *Agriculture* **2025**, *15*, 942. [CrossRef]
5. Saha, S.; Noguchi, N. Smart vineyard row navigation: A machine vision approach leveraging YOLOv8. *Comput. Electron. Agric.* **2025**, *229*, 109839. [CrossRef]
6. Onishi, Y.; Yoshida, T.; Kurita, H.; Fukao, T.; Arihara, H.; Iwai, A. An automated fruit harvesting robot by using deep learning. *ROBOMECH J.* **2019**, *6*, 13. [CrossRef]
7. Singh, C.; Randhawa, G.S.; Farooque, A.A.; Gill, Y.S.; Fraser, A.; K.M., L.K.; Barrett, R.; Al-Mughrabi, K. AgriScout: AI-powered robot for precise detection of PVY-infected potato plants. *Comput. Electron. Agric.* **2025**, *238*, 110781. [CrossRef]
8. Valentea, D.S.M.; Mominb, A.; Griftb, T.; Hansen, A. Accuracy and precision evaluation of two low-cost RTK global navigation satellite systems. *Comput. Electron. Agric.* **2020**, *168*, 105142. [CrossRef]
9. Kabir, M.S.N.; Song, M.Z.; Sung, N.S.; Chung, S.O.; Kim, Y.J.; Noguchi, N.; Hong, S.J. Performance comparison of single and multi-GNSS receivers under agricultural fields in Korea. *Eng. Agric. Environ. Food* **2016**, *9*, 27–35. [CrossRef]

10. Jiang, S.; Wang, S.; Yi, Z.; Zhang, M.; Lv, X. Autonomous Navigation System of Greenhouse Mobile Robot Based on 3D Lidar and 2D Lidar SLAM. *Front. Plant Sci.* **2022**, *13*, 815218. [CrossRef]
11. Mai, C.; Chen, H.; Zeng, L.; Li, Z.; Liu, G.; Qiao, Z.; Qu, Y.; Li, L.; Li, L. A Smart Cane Based on 2D LiDAR and RGB-D Camera Sensor-Realizing Navigation and Obstacle Recognition. *Sensors* **2024**, *24*, 870. [CrossRef]
12. Kong, X.; Guo, Y.; Liang, Z.; Zhang, R.; Hong, Z.; Xue, W. A method for recognizing inter-row navigation lines of rice heading stage based on improved ENet network. *Measurement* **2025**, *241*, 115677. [CrossRef]
13. Mendez, E.; Piña Camacho, J.; Escobedo Cabello, J.A.; Gómez-Espinosa, A. Autonomous Navigation and Crop Row Detection in Vineyards Using Machine Vision with 2D Camera. *Automation* **2023**, *4*, 309–326. [CrossRef]
14. Zhou, X.; Zhang, X.; Zhao, R.; Chen, Y.; Liu, X. Navigation Line Extraction Method for Broad-Leaved Plants in the Multi-Period Environments of the High-Ridge Cultivation Mode. *Agriculture* **2023**, *13*, 1496. [CrossRef]
15. Chen, J.; Qiang, H.; Wu, J.; Xu, G.; Wang, Z. Navigation path extraction for greenhouse cucumber-picking robots using the prediction-point Hough transform. *Comput. Electron. Agric.* **2021**, *180*, 105911. [CrossRef]
16. Aghi, D.; Mazzia, V.; Chiaberge, M. Local Motion Planner for Autonomous Navigation in Vineyards with a RGB-D Camera-Based Algorithm and Deep Learning Synergy. *Machines* **2020**, *8*, 27. [CrossRef]
17. Zhang, L.; Li, M.; Zhu, X.; Chen, Y.; Huang, J.; Wang, Z.; Hu, T.; Wang, Z.; Fang, K. Navigation path recognition between rows of fruit trees based on semantic segmentation. *Comput. Electron. Agric.* **2024**, *216*, 108511. [CrossRef]
18. Zou, Q.; Jiang, H.; Dai, Q.; Yue, Y.; Chen, L.; Wang, Q. Robust Lane Detection from Continuous Driving Scenes Using Deep Neural Networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 41–54. [CrossRef]
19. Xu, S.; Rai, R. Vision-based autonomous navigation stack for tractors operating in peach orchards. *Comput. Electron. Agric.* **2024**, *217*, 108558. [CrossRef]
20. Nguyen, N.T.A.; Pham, C.C.; Lin, W.C. Development of a line following autonomous spraying vehicle with Machine vision-based leaf density estimation for cherry tomato greenhouses. *Comput. Electron. Agric.* **2023**, *215*, 108429. [CrossRef]
21. Zhang, R.; Peng, J.; Gou, W.; Ma, Y.; Chen, J.; Hu, H.; Li, W.; Yin, G.; Li, Z. A robust and real-time lane detection method in low-light scenarios to advanced driver assistance systems. *Expert Syst. Appl.* **2024**, *256*, 124923. [CrossRef]
22. Tabelini, L.; Berriel, R.; Paixão, T.M.; Badue, C.; Souza, A.F.D.; Oliveira-Santos, T. Keep your Eyes on the Lane: Real-time Attention-guided Lane Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; IEEE: Piscataway, NJ, USA, 2020. [CrossRef]
23. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *arXiv* **2015**. [CrossRef]
24. Lin, T.Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal Loss for Dense Object Detection. *arXiv* **2018**. [CrossRef]
25. Corke, P. *Robotics, Vision and Control: Fundamental Algorithms in Python*, 3rd ed.; Springer: Berlin/Heidelberg, Germany, 2023. [CrossRef]
26. Li, Y.; Henke, M.; Zhang, D.; Wang, C.; Wei, M. Optimized Tomato Production in Chinese Solar Greenhouses: The Impact of an East–West Orientation and Wide Row Spacing. *Agronomy* **2024**, *14*, 314. [CrossRef]
27. Sridhara, M. TuSimple. 2021. Available online: https://www.kaggle.com/datasets/manideep1108/tusimple (accessed on 25 November 2024).
28. Inspires, P. Greenhouse-Lanes. 2025. Available online: https://www.kaggle.com/datasets/paoloinspires/greenhouse-lanes (accessed on 25 November 2025).