IJRCS

International Journal of Robotics and Control Systems
Vol. 5, No. 6, 2025, pp. 3129-3146
ISSN 2775-2658
http://pubs2.ascee.org/index.php/ijrcs

ASCEE

Stereo Vision-Based Vehicle Distance Estimation with a Two-Stage Deep Learning Approach

Benhamida Adel [a,1,*], Guessas Laarem [a,2], Benmahammed Khier [a,3], Refoufi Salim [a,4], Boutalbi Oussama [a,b,5]

[a] LSI Laboratory, Ferhat Abbas University - Setif 1, Algeria
[b] University Center Morsli Abdellah of Tipaza, Algeria
[1] a.benhamida.del@gmail.com; [2] guessaslar@yahoo.fr; [3] khierben@yahoo.com; [4] refoufi2003@yahoo.fr;
[5] botalbioussama@gmail.com
[*] Corresponding Author

ARTICLE INFO

Article History
Received August 17, 2025
Revised October 03, 2025
Accepted December 22, 2025

Keywords
Vision;
Drones;
Stereo;
CNN;
YOLO

ABSTRACT

This study presents a two-stage deep learning pipeline for vehicle distance estimation in urban environments, using synthetic stereo dataset generated via Unreal Engine 5.4. The Unreal Engine 5.4 is simulation tool is highly realistic and closely mimics real-world conditions, while being significantly cost-effective and more time-efficient to produce compared to collecting labeled data manually in real-world conditions. A drone-mounted stereo camera system with an 80 cm baseline captured 7,000 stereo image pairs across diverse vehicle positions and dynamic lighting conditions, annotated with precise Euclidean drone-to-car distances, automatically computed via Unreal Engine's Physics Engine, ranging from 10 to 200 meters. The first stage uses YOLOv11 for vehicle detection, and it was trained on this synthetic dataset to identify vehicles in urban scenes. The second stage uses a Convolutional Neural Network (CNN) that processes stereo image crops and normalized bounding box dimensions to predict drone-car distance. The proposed approach uses geometric features (such as bounding box scaling) with learned visual features to enhance distance estimation accuracy. The pipeline achieves a 1.02 meters training RMSE and 4.2 meters validation RMSE, with a mean relative error of 4.02%. The system demonstrates a validation error of 2.2% relative to the maximum distance of 200 meters, demonstrating the effectiveness of the proposed approach within a synthetic environment. Real-world deployment remains a clear objective for future work. However, a key limitation lies in the domain adaptation gap, as our pipeline is trained solely on synthetic data may face challenges when directly deployed in real-world scenarios.

1. Introduction

Urban environments present unique challenges for autonomous driving [1], particularly when it comes to vehicle detection, localization, and depth estimation. Vision-based methods powered by Deep Convolutional Neural Networks (DCNNs) [2] (YOLOv11 [3], [4]) have shown tremendous promise for these tasks. However, training such networks demands vast amounts of annotated data, a process that is both expensive and extremely time-consuming when relying solely on real-world

imagery. Collecting diverse datasets that capture a wide range of weather conditions, varying sun positions (morning, sunset, sunrise), and dynamic urban scenarios often requires extensive manual labor, while domain adaptation and generalization remain open challenges when models are trained only on limited real-world data.

Our work includes leveraging the powerful simulation capabilities of **Unreal Engine 5.4** [5] to generate a highly realistic synthetic stereo dataset. Within a designed urban downtown scene, a flying drone captures stereo image pairs at various altitudes and sun positions. The Blueprint system [6] facilitates real-time control over environmental variables such as lighting intensity and sun positioning, ensuring that the synthetic images closely mimic real-world conditions while providing detailed pixel-level and geometric annotations. This study should be viewed as a demonstration within a synthetic environment, with real-world transfer and deployment framed as future objectives.

Datasets such as SYNTHIA [7] focus on single-view images and are generated using 3d engines like Unity [8], which do not achieve the same level of photorealism as **Unreal Engine**, which offers a level of visual fidelity that surpasses previous synthetic datasets. Also, there are real-world datasets like Cityscapes [9] and KITTI [10] that require extensive manual annotation, our method automates the annotation process.

We used the stereo images dataset to train a YOLOv11 [3], [4] model for vehicle detection, which forms the first stage of our two-stage pipeline. This model is specifically trained to identify and localize cars within the diverse urban scenes captured by our simulated drone. The detection outputs are then passed to the second stage, where a convolutional neural network processes the stereo image crops and normalized bounding box dimensions to perform distance estimation. This integrated approach seamlessly combines advanced object detection with stereo vision-based depth estimation, while mitigating domain adaptation issues through synthetic data generation.

In the second stage of our pipeline, as shown in Fig. 1, we deploy a deep neural network architecture comprising 6 convolutional layers and 3 fully connected layers to perform distance estimation. This model receives as input the cropped car boxes from the YOLOv11 [3], [4] detection stage, along with their normalized width and height. The convolutional layers are responsible for extracting hierarchical visual features from the cropped images, while the fully connected layers integrate these features with the geometric information provided by the normalized bounding box dimensions. Ultimately, the network outputs a single distance value.

1.1. Related Works

Deep convolutional neural networks (DCNNs) have advanced visual recognition tasks [11]–[13], yet their effectiveness depends on large-scale annotated data. To address this, researchers increasingly rely on synthetic datasets that allow automatic annotation of diverse scenarios, including object detection and depth estimation [14]–[16]. For instance, the SYNTHIA dataset provides urban scenes with varied conditions [7], while Cityscapes offers high-resolution real-world imagery with pixel-level labels [9]. Although domain adaptation remains a challenge, studies suggest that sensor discrepancies rather than the synthetic nature of data are the main source of performance gaps, and joint training on synthetic and real data can mitigate this issue [17], [18].

Depth estimation methods can be broadly categorized into passive and active approaches. Passive methods include traditional stereo matching and triangulation [19], [20], which have evolved into deep learning models that predict dense depth maps end-to-end [21], [22]. Monocular depth estimation has likewise progressed from handcrafted cues and features [23], [24] to modern CNN- and transformer-based models [25], [26]. Active methods such as LiDAR and Time-of-Flight sensors achieve high precision but face constraints related to cost, range, and robustness in challenging environments [27]. These limitations strengthen the case for camera-based stereo vision in safety-critical applications.

Several recent studies highlight the potential of stereo vision combined with deep learning for real-world tasks. Kulawik et al. [28] proposed a CNN regression model for obstacle distance estimation from stereo pairs, outperforming traditional sensing methods but showing distance limitations due to input resolution. Lin et al. [29] demonstrated the use of stereo-equipped drones for forestry applications, finding YOLO-based detectors superior to Mask R-CNN for branch detection. In the context of intelligent transport systems, Bourja et al. [30] employed stereo cameras for vehicle distance estimation, leveraging YOLO for robust detection under traffic conditions. Collectively, these works demonstrate the promise of stereo vision and deep learning across domains such as robotics, forestry, and traffic management.

Despite this progress, most prior works rely on earlier generations of detection architectures (e.g., R-CNN, YOLOv3, YOLOv5) or focus on specific depth estimation algorithms without jointly optimizing detection and stereo-based distance estimation. Furthermore, the impact of newer architectures such as YOLOv11 offering improved accuracy, speed, and small-object detection has not been systematically explored in stereo vision applications. Our work addresses this gap by introducing a two-stage pipeline that leverages YOLOv11 for stereo-based vehicle distance estimation, validated in synthetic urban scenes generated with Unreal Engine.

1.2. Our Contribution

First, we developed a highly realistic urban scene with dynamic lighting conditions and varying sun positions using **Unreal Engine 5.4**, which we used to create diverse synthetic dataset of stereo image pairs from multiple drone-to-car distances ranging from 10 to 200 meters using a drone-mounted stereo camera with an 80 cm baseline.

Second, we designed a convolutional neural network composed of six convolutional layers and three fully connected layers. This model receives stereo image crops of detected vehicles along with their normalized bounding box dimensions that output a single value that is the distance shown in Fig. 1.

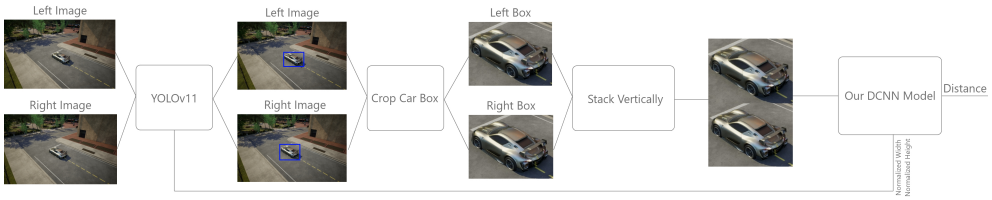


Fig. 1. Proposed two-stage pipeline (left to right)

1.3. Paper Organization

This paper is organized as follows, Section 2 discusses how the urban scene, the car pawn and the drone pawn were created. Section 3 presents how we capture the stereo images and the car's bounding boxes. Section 4 describes the training and evaluation of the YOLOv11 model trained on the generated stereo images. Section 5 presents the design and implementation of a deep convolutional neural network (CNN) for estimating the drone-car distance along with training and validation metrics. Section 6 presents the test results of our model on a diverse test dataset comprising different sun positions and camera viewpoints (Side, Back and Front Car). Section 9 concludes the paper and suggests future work.

2. Experimental Setup

A general view of the experimental setup is highlighted in Fig. 2. It is composed of three main parts: Urban Scene, Car Pawn Setup and Drone Pawn Setup.

(a) Urban Drone Scene

(b) Urban Downtown Scene

(c) Car Pawn

Fig. 2. Urban scenes

2.1. Urban Scene

The urban scene was built in **Unreal Engine 5.4**, using its cutting-edge rendering capabilities and dynamic lighting system to achieve a high level of photorealism. We downloaded a variety of pre-made assets from the **Unreal Engine Fab Store**. These assets include detailed building models, street furniture, vegetation, and other environmental props that together form a downtown urban scene.

We designed the scene to simulate diverse urban conditions and to capture images under different lighting and weather scenarios. We integrated a virtual drone into our setup to traverse the urban environment and capture stereo image pairs from multiple altitudes and viewpoints. This drone, as shown in Fig. 2a, is configured with a stereo-camera with a baseline of 80cm.

The entire process was executed on high-performance hardware, including a 32-core system (2 x Intel® Xeon® Processor E5-2683 v4), 128GB DDR4 RAM,any CUDA-Enabled GPU with enough VRAM, and a 1TB M.2 SAMSUNG EVO 980PRO SSD.

In addition, the downtown urban scene, shown in Fig. 2b, showcases the realistic downtown environment complete with detailed architecture, streets, and dynamic elements.

2.2. Car Pawn Setup

The car pawn seen in Fig. 2c was developed using both C++ and **Unreal Engine**'s Blueprint system. The car Pawn represents the vehicle that will be tracked during the simulation, serving as the moving platform for capturing stereo images via the virtual drone. In our implementation, the core functionalities such as initialization, movement control, and tracking are encapsulated in a C++ class, while Blueprint scripts are used to easily integrate and visually control these behaviors within the Unreal Editor, as shown in Fig. 2c.

2.3. Drone Pawn Setup

In our setup, the drone is equipped with two identical cameras configured with a 90° Field of View (FOV) and an aspect ratio of 16:9. These cameras are mounted with an 80 UU baseline distance (with 1 UU equaling 1 cm in **Unreal Engine**) and are aligned to face in the same direction (0° angle between them) to capture synchronized stereo images. The drone is highly maneuverable and capable of moving in all horizontal directions and rotating around both the car's Z-axis and Y-axis to capture the vehicle from diverse perspectives. While these rotations allow us to achieve varied view angles, the Fig. 3 specifically illustrates the strategic placement and configuration of the cameras in our stereo setup.

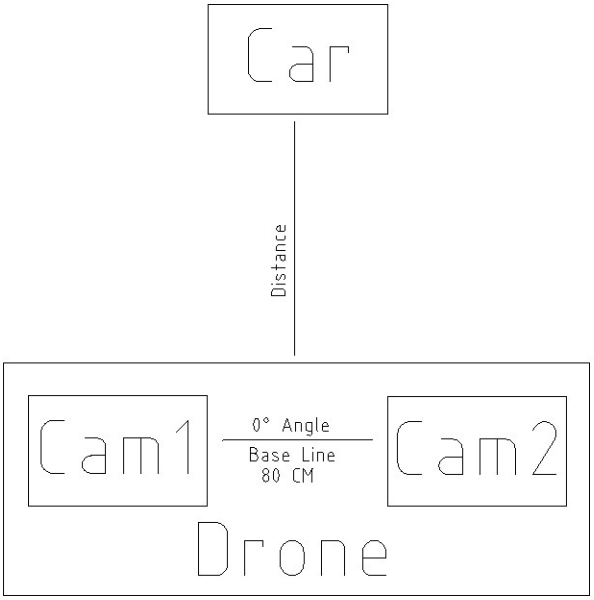


Fig. 3. Drone camera diagram

3. Data Generation Process

The Data Generation Process consists of multiple steps to optimize stereo image capture and associated metadata collection. A detailed breakdown of these steps is provided in the subsections.

3.1. Stereo Image Acquisition

Our stereo image acquisition process is designed to ensure precise and synchronized data capture. The procedure begins by positioning the drone at a chosen location relative to the car pawn, ensuring the vehicle is framed within the cameras' field of view. The drone is equipped with two identical cameras, Cam1 and Cam2, each configured with a 90° Field of View and a 16:9 aspect ratio, mounted with an 80 UU baseline distance and aligned at 0°. Once the drone is in place, the system continuously alternates between Cam1 and Cam2 to capture synchronized stereo image pairs. Furthermore, the drone is able to move forward, backward, right, and left, as well as rotate around the car's Z-axis and Y-axis, enabling it to capture the vehicle at every 20° ± 5° on the X-axis, and once a full turn is completed, the drone increments by 20° ± 5° on the Z-axis. For a visual summary of this workflow, as illustrated in Fig. 4, which provides a diagrammatic representation of the entire stereo imaging process.

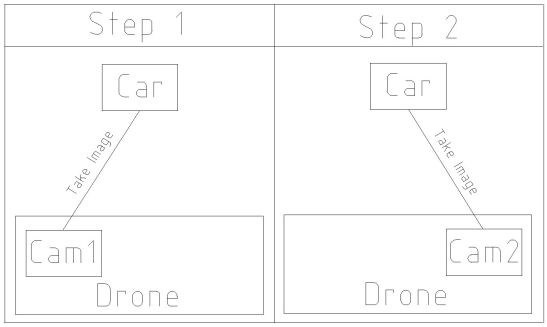


Fig. 4. Drone stereo image acquiring diagram

3.2. Distance Extraction

In this subsection, we utilize **Unreal Engine**'s `GetWorldLocation` function to retrieve the 3D coordinates of both the car and the drone. This function outputs a 3D vector representing the exact location of an object in the virtual environment. With these vectors, we then calculate the distance between the car and the drone using the standard Euclidean distance formula, as highlighted in Fig. 5.

Fig. 5. Get World Location usage

3.3. Car's Bounding Box Extraction From the Image

Unreal Engine's `ProjectWorldLocationToScreen` function is used to convert 3D world coordinates into 2D screen coordinates for accurately extracting the car's bounding box from the drone's camera view. The process begins by projecting the center of the car's bounding sphere onto the screen, establishing a central reference point. To account for perspective variations, the apparent size of the sphere is adjusted based on the alignment between the car's forward vector and the drone's camera viewing direction. By projecting an offset point along the drone's camera right vector, the method calculates the screen-space radius as the distance from the projected center to this offset. This radius is then used to define the bounding box's extreme points, and normalizing these coordinates relative to the viewport dimensions ensures a consistent representation across different resolutions, please refer to Appendix A.

3.4. Persisting Stereo Imagery, Annotations, and Distance Data

In the dataset generation process, stereo images are saved in PNG format. Each image file is named using a standardized format: "right_xxxxx.png" for images captured by the right camera and "left_xxxxx.png" for those from the left camera, where "xxxxx" represents the timestamp in seconds. Additionally, for every captured image, a text file is also generated containing detailed metadata including the car's bounding box coordinates with the same image name, the center of the car in the image, and the ground-truth distance between the drone and the car. The complete dataset comprises 7,000 stereo images (capturing both right and left views), the dataset can be scaled up if needed, and the scenes can be easily modified to generate different view scenes.

3.5. Sample Generated Images

This subsection shows representative samples from our synthetic dataset, showcasing the stereo images generated by the drone's cameras, as shown in Fig. 6.

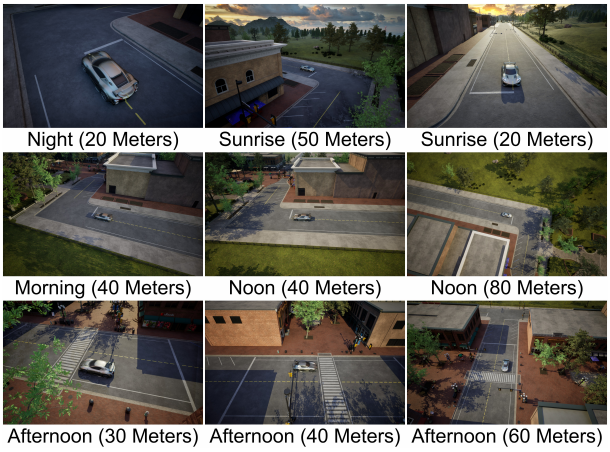


Fig. 6. Sample Images

3.6. Workflow Diagram

The dataset generation workflow digram is illustrated in Fig. 7.

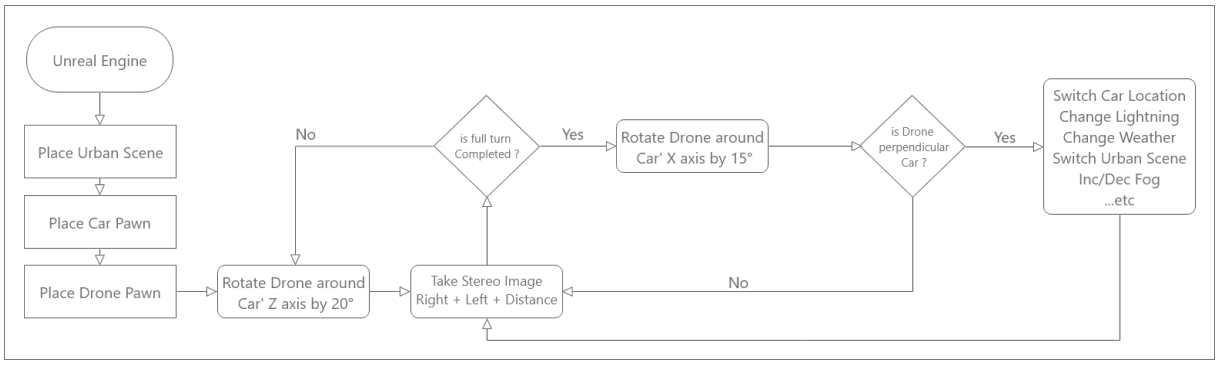


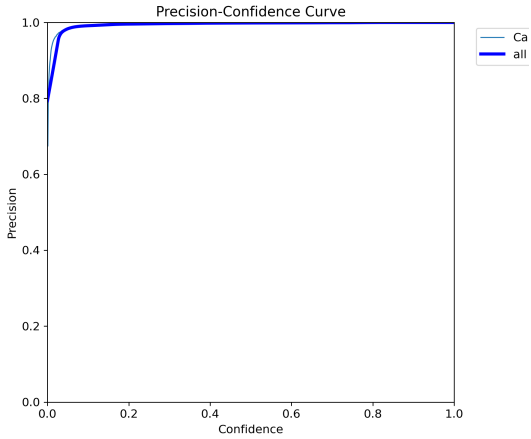
Fig. 7. Dataset Generation Workflow Diagram

4. YOLOv11 Model Training and Evaluation

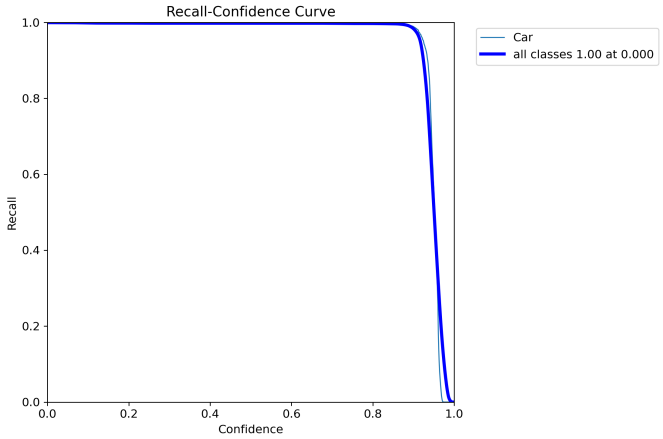
The YOLOv11 model was trained on our synthetic stereo dataset that consists of 7,000 stereo images annotated using the subsection 3.3 procedure. The dataset was split into 80% for training and 20% for validation. Additionally, the model was trained on a single class, the car class. For this work, we employed the `yolo11m.pt` (medium) variant of YOLOv11, which provided a good balance between model size and detection accuracy (Precision).

4.1. Training Results

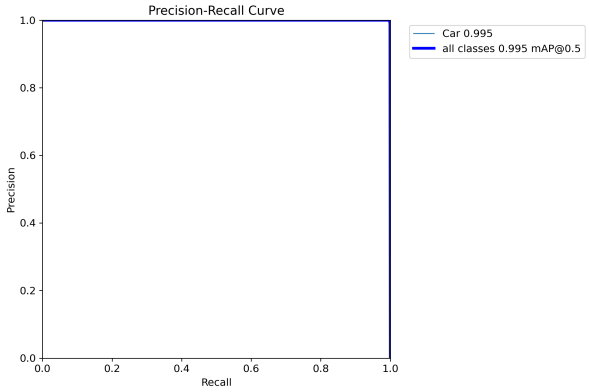
The training results indicate the model's capability to minimize false positives while accurately detecting the car class. High precision across the validation set demonstrates that the YOLOv11 model is reliable in distinguishing cars from background clutter in diverse urban scenarios. Detailed precision metrics and graphical representations, as shown in Fig. 8, achieving a precision of 99.8%.



(a) Precision curve



(b) Recall curve



(c) Precision recall curve

Fig. 8. Precision/recall training curves

4.2. Other Evaluation Metrics

In addition to precision and recall, we evaluated the model using several other standard object detection metrics. The **mAP-50** metric, which measures the mean average precision at an IoU threshold of 0.50, and the **mAP-95**, which averages precision over IoU thresholds from 0.50 to 0.95. Metrics such as box loss and train loss were monitored throughout the training process. Detailed graphs of these metrics, as shown in Fig. 9. A representative validation image, displayed in Fig. 10, further illustrates the model's capability by showing instances of the car class accurately detected and boxed, the time YOLOv11 takes to process an image (2048x1080) is **138ms** on average.

5. Deep CNN Architecture for Distance Estimation

In this section, we define a DCNN for distance estimation. This DCNN is implemented using the PyTorch framework. The network is designed to predict the distance of a detected car by processing a combined input that consists of an RGB image and geometric information. Specifically, the image input is formed by stacking the car boxes extracted from the right and left stereo cameras on top of each other, resulting in a tensor of shape (3, 828, 414). Alongside this image data, the second component is a one-dimensional vector containing two elements representing the normalized width and normalized height of the detected car bounding box. Together, these inputs form a composite input shape of (2,(3,828,414)).

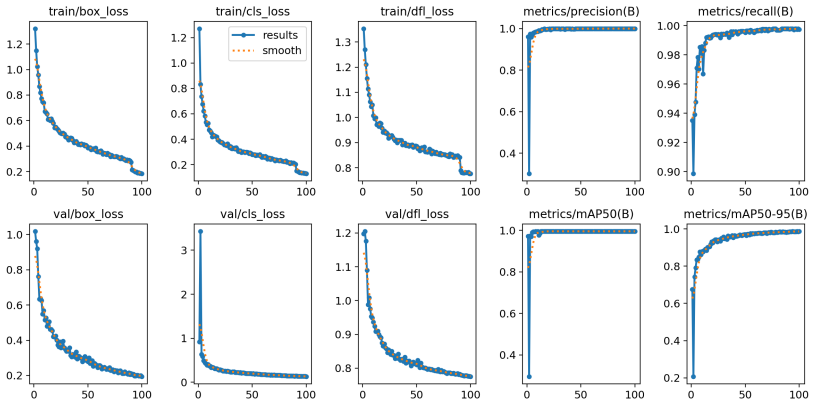


Fig. 9. YOLOv11 training metrics

Fig. 10. Validation samples

5.1. CNN Architecture

A general overview of proposed DCNN is given in Fig. 11. The architecture begins by splitting the input image directly into two parallel CNN pathways with shared weights, each composed of five successive processing blocks. In **Block 1**, three back-to-back convolutional layers (3×3 kernel, stride 1) map the 3-channel input to 32 feature maps (without ReLU between them), followed by parallel pooling: max pooling (2×2 kernel, stride 2) and ReLU on the first pathway, and average pooling (2×2 kernel, stride 2) and ReLU on the second pathway. **Block 2** applies three successive convolutions (3×3 kernel, stride 1) with 32 input and 25 output feature maps (no intermediate ReLU), then parallel pooling max in the first branch and average in the second (2×2 kernel, stride 2) each followed by ReLU. **Blocks 3** and **4** each consist of two successive convolutional layers (3×3 kernel, stride 1) mapping 25→18 and 18→10 feature maps respectively (no ReLUs between layers), each followed by the same branch-specific pooling (max on the first pathway, average on the second; 2×2 kernel, stride 2) and ReLU. Finally, **Block 5** comprises two successive convolutions (3×3 kernel, stride 1) from 10 to 5 feature maps (without an intermediate ReLU), followed by parallel pooling (max on the first pathway, average on the second; 2×2 kernel, stride 2) and ReLU.

After the convolutional stages, the outputs from both branches are flattened and concatenated, forming a unified feature vector that encapsulates the learned spatial hierarchies. This vector is then

fed into a series of fully connected (FC) layers. The first FC layer reduces the feature dimensionality to 128 outputs, followed by a ReLU activation. The subsequent FC layer further condenses the features to 64 outputs, again followed by a ReLU activation. At this juncture, the network concatenates the FC output with the original normalized width and height vector. This fusion of visual and geometric information is critical for accurately estimating the distance. Finally, the output layer, consisting of a single neuron, produces the predicted distance value, effectively combining both visual cues and spatial dimensions to deliver a precise estimation.

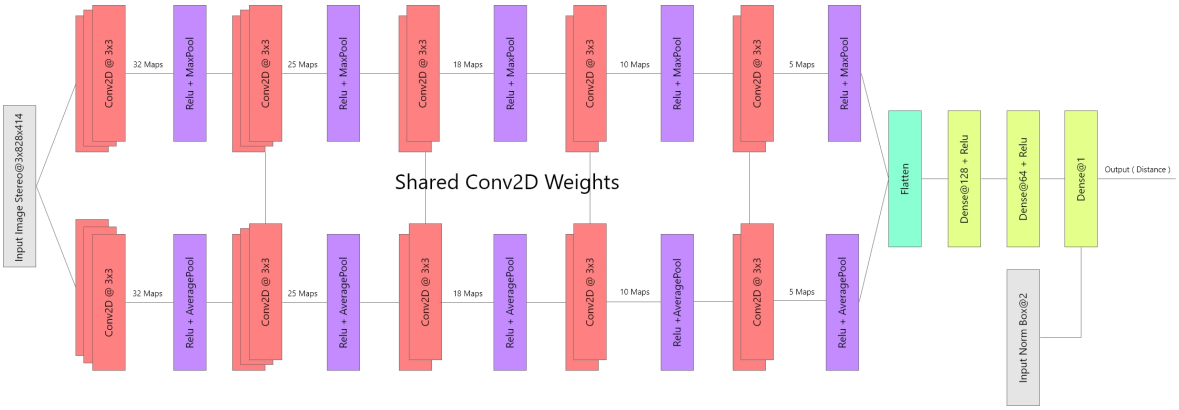


Fig. 11. Deep neural network for distance estimation

5.2. Training Setup and Data Preparation

For the distance estimation model, the dataset was meticulously prepared from synthetic stereo images generated earlier. Each scene was captured from both the left and right cameras and saved as `left_xxxxx.png` and `right_xxxxx.png` respectively. A custom Python script was developed to automate the preprocessing: it cropped the car bounding boxes from both images, concatenated them vertically to form a single composite image, and generated a corresponding text file (with the same base name but a `.txt` extension). This text file contains the actual car distance and the normalized width and height of the bounding box (CSV).
The prepared dataset was first normalized using the channel-wise statistics

$$\mu_c = \frac{1}{N \cdot H \cdot W} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} I_{n,c,h,w} \tag{1}$$

$$\sigma_c = \sqrt{\frac{1}{N \cdot H \cdot W} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} I_{n,c,h,w}^2 - \mu_c^2} \tag{2}$$

where:

- μ_c is the Mean value of all pixels in a channel c (Red, Green or Blue).
- σ_c is the Standard Deviation of a specific channel c.
- $I_{n,c,h,w}$ is the pixel intensity at position (h, w) in channel c of image n.
- N is the total number of images.
- H Height of the image in pixels.
- W Width of the image in pixels.

We got mean values of 0.3641, 0.3769 and 0.3985 for RGB Channels respectively and std values of 0.1710, 0.1630 and 0.1481 for RGB Channels respectively, then we it split into 80% for training and 20% for validation. The training process was implemented using the PyTorch framework. We used the ADAM optimizer with a learning rate of 0.001 and used the Mean Squared Error (MSE)

with the Mean Squared Relative Error (MSRE) loss functions to guide the network's training. A batch size of 16 was chosen. Additionally, the TensorBoard library was used to display the training logs and monitor performance metrics in real-time, This carefully designed setup laid the groundwork for effectively training the deep CNN architecture for accurate distance estimation.

5.3. Loss Function

We used two loss functions to optimize the performance of our model: the MSE and the MSRE. The MSE loss function penalizes larger errors more significantly, ensuring that the model minimizes substantial deviations between the predicted and ground truth distances.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i^{\text{pred}} - y_i^{\text{true}}\right)^2 \tag{3}$$

In parallel, the MSRE loss penalizes errors relative to the magnitude of the ground truth. For example, when the true distance is 10 meters, a 2-meter error results in a 20% relative error, whereas the same 2-meter error for a 200-meter ground truth is only a 1% relative error.

$$\text{MSRE} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i^{\text{pred}} - y_i^{\text{true}}}{y_i^{\text{true}}} \times 100\right)^2 \tag{4}$$

Where y_i^{pred} and y_i^{true} denote the predicted and ground truth values respectively and N is the number of samples.
The final Loss function is:

$$L = Max(\text{MSE}, \text{MSRE}) \tag{5}$$

This approach ensures that our model is more sensitive to errors when estimating shorter distances, which is particularly important for accurate vehicle distance estimation in our stereo vision system.

5.4. Training Sample

A representative training sample is illustrated in Fig. 12 with the bounding boxes for the detected vehicles from the right and left images stacked vertically.

Fig. 12. Training Sample (33.3 Meters)

5.5. Loss Curves and Convergence

The network was trained using the MSE and the MSRE loss functions, and the loss curves for both training and validation indicate effective convergence. During training, the MSE loss consistently decreased and eventually converged to approximately 4, demonstrating that the model has learned the mapping from the input to the target distance accurately. In contrast, the validation MSE loss converged to around 22. The convergence behavior is visually represented in the training and validation loss curves seen in Fig. 13a and Fig. 13b respectively , which illustrate a steady decline in loss values over epochs and confirm the overall stability of the training process.

In addition to the MSE loss, we also evaluated the model's performance using several key error metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Relative Error (MRE). The MAE provides a direct measure of the average absolute error between the predicted and actual distances, while the RMSE, by penalizing larger errors more heavily, offers insight into the model's sensitivity to outlier predictions. The MRE quantifies the average relative error by normalizing the absolute error with respect to the actual value, providing a relative measure of performance. For example, an error of 2 meters constitutes a 20% error when the actual distance is 10 meters, whereas the same 2-meter error is only a 1% error when the actual distance is 200 meters, giving a normalized perspective on performance. During training, the MAE and RMSE curves illustrated in Fig. 13e and Fig. 13c respectively demonstrated a steady decline, converging to low error values that indicate precise distance predictions. On the validation side, while the MAE, RMSE, and MRE curves shown in Fig. 13f, Fig. 13d, and Fig. 13g converging to **3.8, 4.4, and 4.4%** respectively, indicated a slight increase in error values compared to training, they still reflect the model's robust performance in estimating distances on unseen data.

6. Results

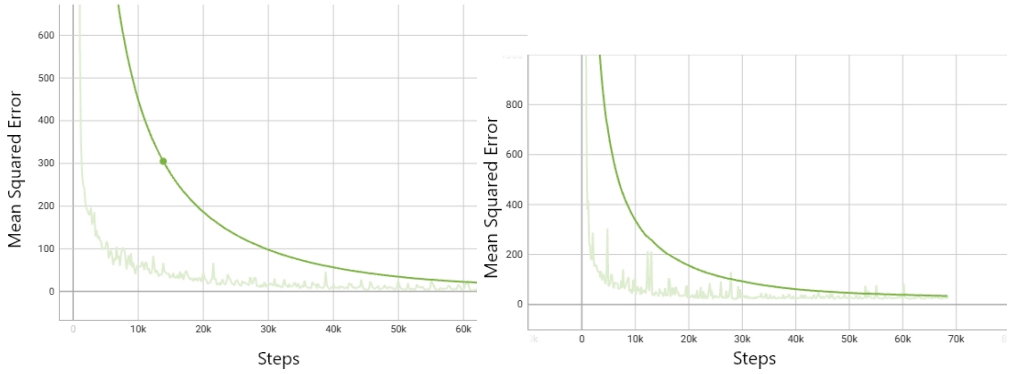
To evaluate the performance of our distance estimation model, we constructed a test dataset. The test dataset includes a total of 108 images, consisting of 12 samples for each target distance: 15 meters, 35 meters, 55 meters, 85 meters, 105 meters, 135 meters, 155 meters, 185 meters, and 200 meters.

Each set of 15 samples per distance covers various lighting and positional scenarios. Specifically, the dataset incorporates five different sun positions (sunrise, morning, noon, and afternoon), and for each sun position, three distinct stereo viewpoints were captured relative to the vehicle: front-facing, side view, and rear view.

The evaluation results, listed in Table 1 using the RMSE metric between predicted and ground-truth distances, shows that our model achieves an acceptable accuracy in the range of 15–55 m with a RMSE below 4 m and reaches a minimum of 1.55 m at 85 m, identifying this as the model's "sweet spot." Beyond 105 m, error grows predictably as disparity cues weaken rising from 3.29 m at 105 m to 7.29 m at 135 m and peaking at 11.29 m at 200 m. Illumination has only a modest effect within the reliable envelope (15–105 m), introducing under 2 m of variation across sunrise, morning, noon, and afternoon scenarios; at longer ranges (135–200 m), however, sun-position swings can amplify uncertainty by up to 10 m, with noon (minimal shadows, highest contrast) yielding slightly better long-range estimates. The inference time that each test image took is around **150ms** on average. This two-stage process takes less than **300ms** to complete.

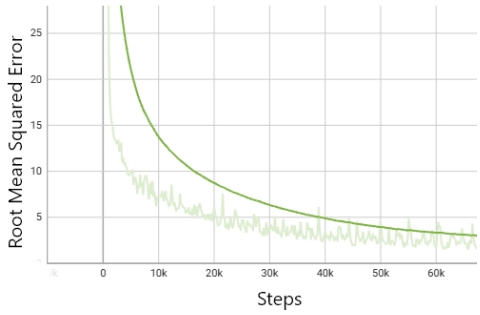
7. Importance of Stereo Input

This experiment was performed using the same model architecture, input shape, and dataset size to isolate the effect of input variation. As shown in Fig. 14, the model trained on the right–right configuration failed to converge and diverged across all training metrics (RMSE, MAE, MRE).

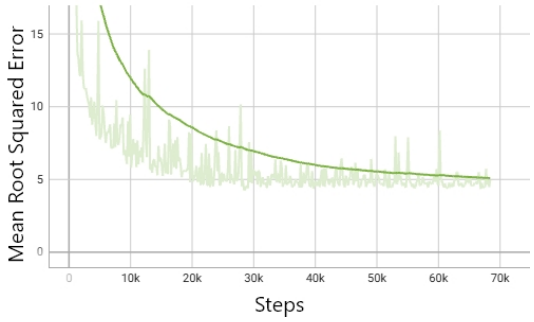


(a) Training Loss Curve (MSE)

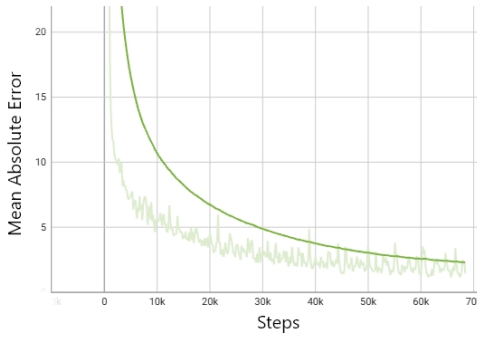
(b) Validation Loss Curve (MSE)



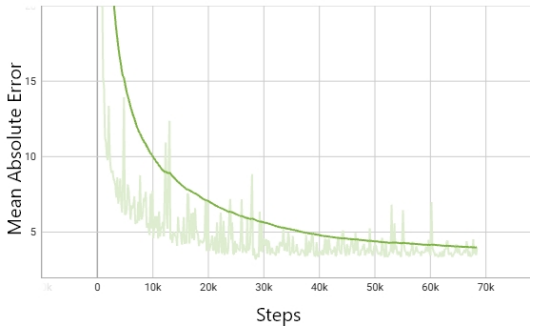
(c) Training Loss Curve (RMSE)



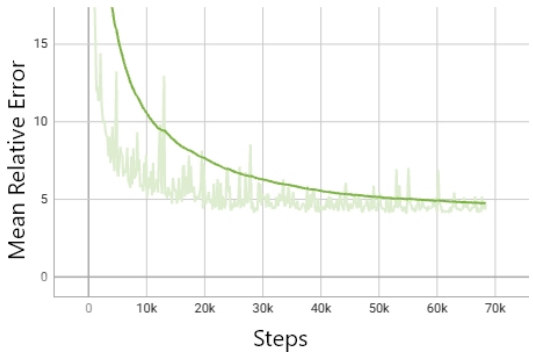
(d) Validation Loss Curve (RMSE)



(e) Training Loss Curve (MAE)



(f) Validation Loss Curve (MAE)



(g) Validation Curve (MRE)

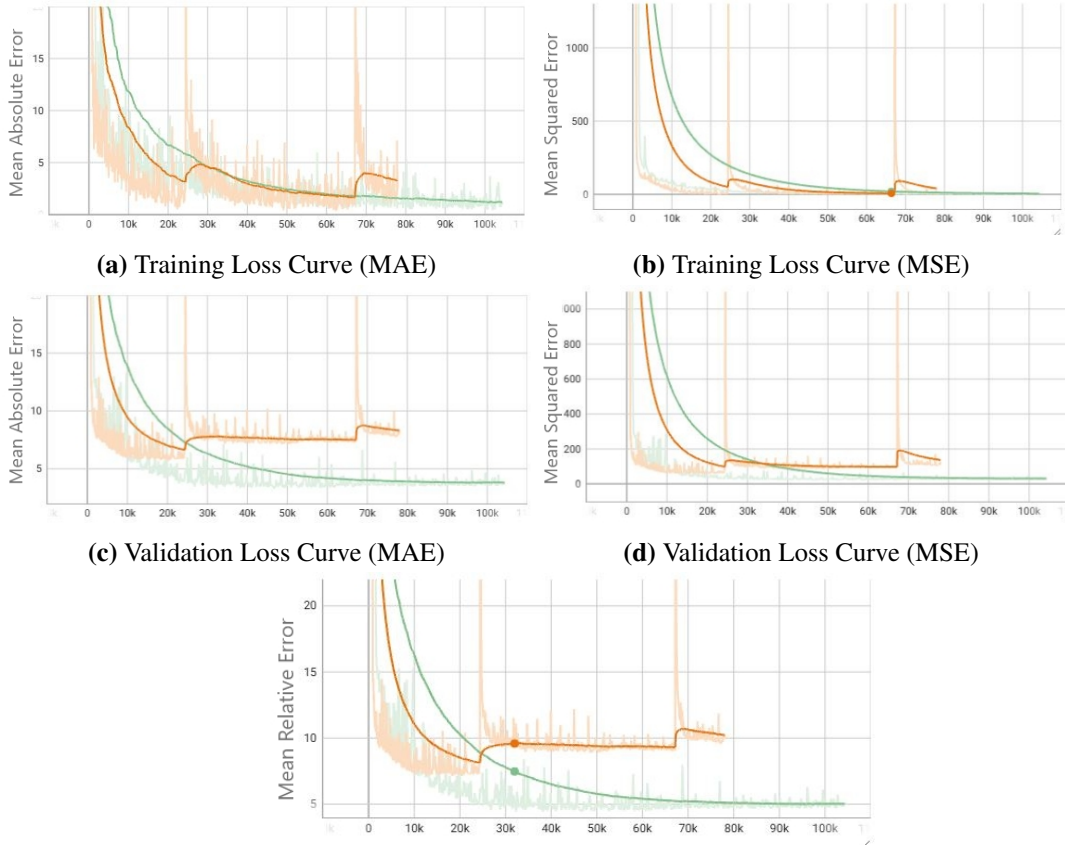
Fig. 13. Training curves

These observations indicate that the model struggled under this non-physical input setup, suggesting the need for more rigorous ablation studies (e.g., monocular or disparity-based inputs) to

better assess the role of stereo disparity in feature learning.

Table 1. RMSE of predicted distances at various ground truth distances

Sun Position	Ground-Truth Distance (Meters)								
	15	**35**	**55**	**85**	**105**	**135**	**155**	**185**	**200**
Sunrise	16.53	32.76	49.95	83.42	101.57	122.64	143.97	174.03	183.80
Morning	17.21	28.15	51.78	82.44	102.37	127.79	148.76	174.48	187.96
Noon	16.62	31.97	53.50	84.18	100.00	128.13	147.95	172.13	192.03
Afternoon	17.48	32.97	53.28	83.74	102.89	132.28	151.93	176.57	191.04
RMSE	1.96	3.54	2.87	**1.55**	3.29	7.29	6.85	10.70	11.29
MRE	13.06%	10.11%	5.02%	**1.82%**	3.13%	5.40%	4.41%	5.78%	5.60%



(a) Training Loss Curve (MAE)

(b) Training Loss Curve (MSE)

(c) Validation Loss Curve (MAE)

(d) Validation Loss Curve (MSE)

(e) Validation Curve (MRE)

Fig. 14. Right-Right vs Right-Left crops training curves

8. Dataset Availability and Access

The complete synthetic dataset is hosted online and can be accessed upon request from the author. For those interested in utilizing the dataset for research or development purposes, please contact the corresponding author to receive detailed access instructions and any additional supporting documentation.

9. Conclusion

In conclusion, our work demonstrates the efficacy of integrating synthetic data generation with a two-stage deep learning pipeline for robust vehicle distance estimation in urban environments. Lever-

aging the advanced simulation capabilities of **Unreal Engine 5.4**, we produced a highly realistic stereo dataset that not only reduces the need for costly, manual annotations but also captures diverse environmental conditions essential for real-world applications. The first stage, based on YOLOv11, effectively detects vehicles within these scenes, while the second stage employs a dedicated CNN to predict distances by integrating visual features with geometric priors. Our approach achieves a training RMSE of 1.02 m and a validation RMSE of 4.2 m, corresponding to a mean relative error of 4.02% and a validation error of 2.2% relative to the 200 m maximum distance, demonstrating its effectiveness in a synthetic environment. While these results are promising, applying the pipeline to real-world urban traffic monitoring and drone navigation remains future work. Overall, this study underscores the advantages of synthetic datasets and deep learning in overcoming traditional limitations in stereo vision and paves the way for further advancements in both vehicle localization and drone-assisted operations in dynamic urban settings.

Nonetheless, this study has important limitations that should be acknowledged. First, the bounding box approximation employed in the dataset generation pipeline represents a simplification of real vehicle geometry and may introduce inaccuracies in object localization. Second, the experimental evaluation was conducted in a single synthetic urban environment, which limits the diversity of road layouts, vehicle types, and environmental conditions represented in the data. These constraints may affect the generalizability of the proposed pipeline.

As future work, we aim to apply our current pipeline to real-world datasets containing stereo imagery of urban scenes with ground truth distance labels for various vehicles. Particular attention will be given to addressing the domain adaptation challenge to mitigate the gap between synthetic and real data. Our objective is to evaluate whether the combined detection and distance estimation pipeline can maintain robust performance under real-world conditions, including variations in noise, occlusion, and sensor quality.

Author Contribution: All authors contributed equally to the main contributor to this paper. All authors read and approved the final paper.

Acknowledgment: Big Thanks to Oussama Boutalbi and the LSI Laboratory for the help they provided, Oussama Belkhaouni for sponsoring us, Big Thanks to My Mother who has passed way, Big Love.

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

Appendix

A. Mathematical Derivations for Car Bounding Box Extraction

This appendix provides the detailed mathematical formulation of the method used to extract the car's bounding box from the drone's camera view.

A.1. Projection of the Sphere Center

Let the sphere's center in world space be given by

$$\mathbf{C} = \begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix} \tag{6}$$

Its projection onto the 2D screen is denoted as

$$\mathbf{S}_C = \text{Proj}(\mathbf{C}) = (x_C, y_C) \tag{7}$$

where $\text{Proj}(\cdot)$ represents the projection operation as implemented by **Unreal Engine**.

A.2. Determination of the Effective Radius

The effective radius is adjusted based on the relative orientation between the car's forward vector and the drone's camera view. Let $\mathbf{F}$ and $\mathbf{V}$ be the car's forward vector and the drone's camera forward (view) vector,

respectively. Their alignment is measured by the dot product:

$$d = \mathbf{F} \cdot \mathbf{V}. \tag{8}$$

We then compute the factor as:

$$\text{Factor} = 1 - |d| \tag{9}$$

This factor is chosen because when $|d| = 0$ (i.e., the drone's camera is oriented perpendicular to the car's forward vector), the bounding sphere appears at its maximum size in the 2D view. Conversely, when $|d| = 1$, the sphere appears smaller, reducing the effective radius to the minimum value. Given a minimum radius MinRadius and the full sphere radius SphereRadius, the effective radius R_{eff} is computed via linear interpolation:

$$R_{\text{eff}} = \text{MinRadius} + \text{Factor} \cdot (\text{SphereRadius} - \text{MinRadius}) \tag{10}$$

A.3. Calculation of the Screen-Space Radius

To determine the screen-space radius, we first compute an offset point along the drone's camera right vector. Let $\mathbf{R}$ denote the drone's camera right vector. Then, the world-space coordinates of the sphere's edge are:

$$\mathbf{C}_{\text{edge}} = \mathbf{C} + R_{\text{eff}} \cdot \mathbf{R} \tag{11}$$

Projecting this point onto the screen yields:

$$\mathbf{S}_{\text{edge}} = \text{Proj}(\mathbf{C}_{\text{edge}}) = (x_E,\, y_E) \tag{12}$$

The screen-space radius r is then defined as the Euclidean distance between the projected center and edge:

$$r = \sqrt{(x_E - x_C)^2 + (y_E - y_C)^2} \tag{13}$$

A.4. Construction and Normalization of the Bounding Box

Assuming that the projected sphere is enclosed in a square, the extreme points of the bounding box in screen space are:

$$\begin{aligned} \text{Top-Left}: &\quad (x_C - r,\, y_C - r) \\ \text{Top-Right}: &\quad (x_C + r,\, y_C - r) \\ \text{Bottom-Right}: &\quad (x_C + r,\, y_C + r) \\ \text{Bottom-Left}: &\quad (x_C - r,\, y_C + r) \end{aligned}$$

Let the viewport have dimensions W (width) and H (height). The width and height of the bounding box are given by:

$$\text{BoxWidth} = \text{BoxHeight} = 2r.$$

Normalizing these dimensions relative to the viewport, we have:

$$\begin{aligned} \text{Normalized Width} &= \frac{2r}{W} \\ \text{Normalized Height} &= \frac{2r}{H} \end{aligned} \tag{14}$$

This formulation rigorously encapsulates the process used to extract the car's bounding box from the drone's camera view, ensuring that the final bounding box is resolution-independent and accurately reflects the car's position and apparent size.

References

[1] E. Yurtsever, J. Lambert, A. Carballo and K. Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," in *IEEE Access*, vol. 8, pp. 58443-58469, 2020, https://doi.org/10.1109/ACCESS.2020.2983149.

[2] W. Rawat and Z. Wang, "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review," in *Neural Computation*, vol. 29, no. 9, pp. 2352-2449, 2017, https://doi.org/10.1162/neco_a_00990.

[3] M. L. Ali and Z. Zhang, "The yolo framework: A comprehensive review of evolution, applications, and benchmarks in object detection," *Computers*, vol. 13, no. 12, p. 336, 2024, https://doi.org/10.3390/computers13120336.

[4] R. Khanam and M. Hussain, "Yolov11: An overview of the key architectural enhancements," *arXiv*, 2024, https://doi.org/10.48550/arXiv.2410.17725.

[5] U. Engine, "Unreal engine," *Real-time 3D creation tool*, 2018, https://d1.awsstatic.com/awsmp/solutions/mk-sol-files/media-entertainment/AWSMP-MandE-Game-Tech-Datasheet-Epic-Games.pdf.

[6] M. Romero and B. Sewell, *Blueprints Visual Scripting for Unreal Engine 5: Unleash the true power of Blueprints to create impressive games and applications in UE5*, Packt Publishing Ltd, 2022, https://books.google.co.id/books?id=yhZuEAAAQBAJ&hl=id&source=gbs_navlinks_s.

[7] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3234–3243, 2016, https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Ros_The_SYNTHIA_Dataset_CVPR_2016_paper.html.

[8] Unity Technologies, "Unity Game Engine," 2025, https://unity.com/.

[9] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016, https://openaccess.thecvf.com/content_cvpr_2016/html/Cordts_The_Cityscapes_Dataset_CVPR_2016_paper.html.

[10] Y. Cabon, N. Murray, and M. Humenberger, "Virtual kitti 2," *arXiv*, 2020, https://doi.org/10.48550/arXiv.2001.10773.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012, https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014, https://openaccess.thecvf.com/content_cvpr_2014/html/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.html.

[13] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *Advances in neural information processing systems*, vol. 27, 2014, https://proceedings.neurips.cc/paper_files/paper/2014/hash/ca007296a63f7d1721a2399d56363022-Abstract.html.

[14] J. Marín, D. Vázquez, D. Gerónimo and A. M. López, "Learning appearance in virtual scenarios for pedestrian detection," *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 137-144, 2010, https://doi.org/10.1109/CVPR.2010.5540218.

[15] J. Shotton *et al*., "Real-time human pose recognition in parts from single depth images," *CVPR 2011*, pp. 1297-1304, 2011, https://doi.org/10.1109/CVPR.2011.5995316.

[16] M. Aubry, D. Maturana, A. A. Efros, B. C. Russell, and J. Sivic, "Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3762–3769, 2014, https://doi.org/10.1109/CVPR.2014.487.

[17] D. Vázquez, A. M. López, J. Marín, D. Ponsa and D. Gerónimo, "Virtual and Real World Adaptation for Pedestrian Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 797-809, 2014, https://doi.org/10.1109/TPAMI.2013.163.

[18] A. Torralba and A. A. Efros, "Unbiased look at dataset bias," *CVPR 2011*, pp. 1521-1528, 2011, https://doi.org/10.1109/CVPR.2011.5995347.

[19] A. N. Gorban, E. M. Mirkes, and I. Y. Tyukin, "How deep should be the depth of convolutional neural networks: a backyard dog case study," *Cognitive Computation*, vol. 12, no. 2, pp. 388–397, 2020, https://doi.org/10.1007/s12559-019-09667-7.

[20] J. Žbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *Journal of Machine Learning Research*, vol. 17, no. 65, pp. 1–32, 2016, https://www.jmlr.org/papers/volume17/15-535/15-535.pdf.

[21] J. M. Facil, B. Ummenhofer, H. Zhou, L. Montesano, T. Brox, and J. Civera, "Cam-convs: Camera-aware multi-scale convolutions for single-view depth," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11826–11835, 2019, https://doi.org/10.1109/CVPR.2019.01210.

[22] Z. Lai, E. Lu, and W. Xie, "Mast: A memory-augmented self-supervised tracker," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6479–6488, 2020, https://doi.org/10.1109/CVPR42600.2020.00651.

[23] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, pp. 404–417, 2006, https://doi.org/10.1007/11744023_32.

[24] D. G. Lowe, "Object recognition from local scale-invariant features," *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol.2, pp. 1150-1157, 1999, https://doi.org/10.1109/ICCV.1999.790410.

[25] A. Bhoi, "Monocular depth estimation: A survey," *arXiv*, 2019, https://doi.org/10.48550/arXiv.1901.09402.

[26] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao, "Depth anything: Unleashing the power of large-scale unlabeled data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10371–10381, 2024, https://doi.org/10.1109/CVPR52733.2024.00987.

[27] X. Zhao, P. Sun, Z. Xu, H. Min and H. Yu, "Fusion of 3D LIDAR and Camera Data for Object Detection in Autonomous Vehicle Applications," in *IEEE Sensors Journal*, vol. 20, no. 9, pp. 4901-4913, 2020, https://doi.org/10.1109/JSEN.2020.2966034.

[28] J. Kulawik, "Estimating the distance to an object from grayscale stereo images using deep learning," *Journal of Applied Mathematics and Computational Mechanics*, vol. 21, no. 4, pp. 60–72, 2022, http://dx.doi.org/10.17512/jamcm.2022.4.06.

[29] Y. Lin, B. Xue, M. Zhang, S. Schofield, and R. Green, "Drone stereo vision for radiata pine branch detection and distance measurement: Utilizing deep learning and yolo integration," *arXiv*, 2024, https://doi.org/10.48550/arXiv.2409.17526.

[30] O. Bourja, H. Derrouz, H. Abdelali, A. Maach, R. Thami, and F. Bourzeix, "Real time vehicle detection, tracking, and inter-vehicle distance estimation based on stereovision and deep learning using yolov3," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 8, pp. 915–923, 2021, https://www.academia.edu/download/71897946/Paper_101-Real_Time_Vehicle_Detection_Tracking.pdf.