BABEŞ BOLYAI UNIVERSITY, CLUJ NAPOCA, ROMÂNIA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Hybrid Python–Rust Architecture for Real-Time Intelligent Perception in Autonomous Driving

– ITSG report –

Team members
David Szilagyi, SIA, 251, david.szilagyi2@stud.ubbcluj.ro
Razvan Filea, SIA, 251, razvan.filea@stud.ubbcluj.ro
Armin Torok, SIA, 251, armin.torok@stud.ubbcluj.ro

2025-2026

Abstract

Intelligent perception is a core requirement for autonomous driving, where multiple vision tasks like lane detection, object tracking, and traffic sign recognition must operate reliably in real time. This project presents an integrated **machine learning–based perception system** built upon a hybrid **Python–Rust architecture**, combining the flexibility of Python's ML ecosystem with the efficiency and determinism of Rust. The proposed system serves as both a research and deployment framework for developing and evaluating real-time vision algorithms on autonomous vehicles.

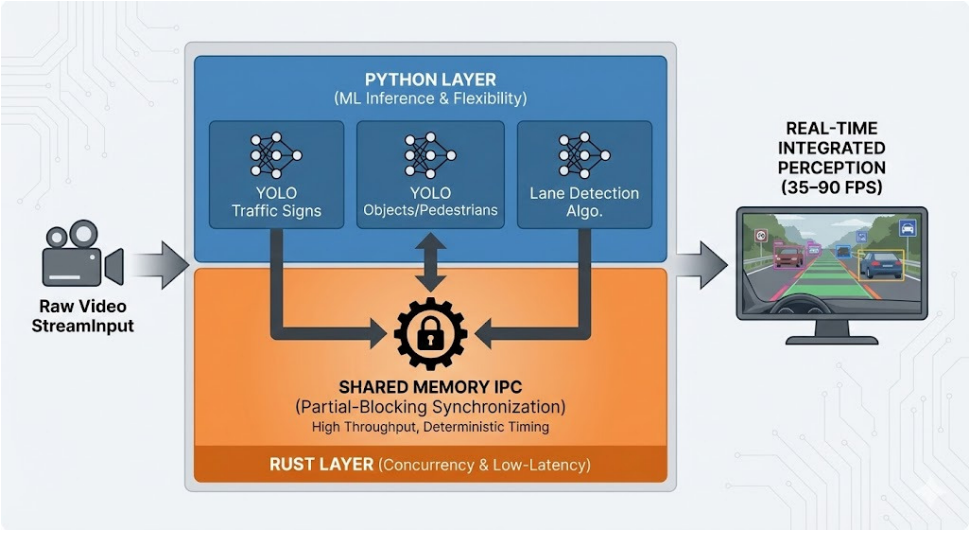
Relevance and importance. Traditional Python pipelines simplify prototyping but struggle with concurrency and latency when several neural networks run in parallel. By embedding these ML modules into a Rust-backed shared-memory infrastructure using a **partial-blocking inter-process communication (IPC)** strategy, the system achieves high throughput without sacrificing development simplicity. It enables the use of advanced deep learning models in a structure suitable for embedded and safety-critical robotics applications.

Intelligent methods used. The perception stack employs **YOLO** models, fine-tuned on *hand-crafted datasets* derived from videos recorded in manual (RC) mode on the autonomous vehicle platform. The collected footage was passed through a custom augmentation and labeling pipeline to increase dataset diversity and accuracy. Each trained model specializes in a different perception task (lane detection, object recognition, sign classification), and all are executed concurrently inside the shared Python–Rust runtime, coordinated by the Partial-Blocking communication policy.

Data used. Training data originates from real-world miniature driving experiments on a 1:10 scale platform. The runtime system was tested on Jetson AGX Orin and desktop CUDA/non-CUDA workstations.

Results. The hybrid framework delivers real-time inference performance between 35–90 FPS while reducing inter-process latency by up to 75% compared to standard multiprocessing. This allowed the vehicle to detect all relevant features with very high accuracy (although not confidence) at 35 FPS, during closed-loop autonomous driving tests. Beyond speed, it provides a seamless interface for ML development, models can be retrained or swapped in Python without modifying the Rust layer, allowing fast experimentation and reliable deployment within a unified environment.

Graphical Abstract



YOLO-based perception modules (object, lane, sign detection) integrated in a hybrid Python–Rust real-time pipeline with shared-memory communication.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 What? Why? How?

What is the problem? Autonomous driving requires real-time understanding of the vehicle's surroundings: detecting lanes, recognizing signs, tracking dynamic obstacles, and fusing multiple sensor inputs. These perception tasks are typically solved using deep learning methods such as convolutional neural networks (CNNs) or transformer-based vision models that must run efficiently. However, executing several ML models concurrently introduces synchronization delays and communication overhead, particularly in Python-based prototypes. The scientific problem addressed here is how to combine the flexibility of ML model development with the performance required for real-time operation.

Why is it important? The quality of perception directly determines driving safety. An accurate but slow object detector can be less useful than a slightly less precise one that operates in real time. Researchers often face a trade-off between rapid experimentation (using Python ML frameworks such as PyTorch or TensorFlow) and deployment efficiency (which requires low-level optimization). A unified framework that keeps Python for development but offloads concurrency and memory management to Rust enables reproducible ML experiments that scale smoothly to embedded platforms.

How do we address it? Our solution integrates multiple trained ML models into a **hybrid Python–Rust architecture** that manages all data flow and synchronization through shared memory. The perception modules, each responsible for a distinct vision task, run as independent processes connected by a *Partial-Blocking Producer–Consumer* communication model. This mechanism allows fast inference branches (e.g., lane detection) to update continuously while slower ones (e.g., sign recognition) operate asynchronously, without stalling the system. Rust provides atomic synchronization primitives and watchdog-based runtime verification, while Python handles model inference, visualization, and logging. Together, they deliver both development agility and real-time performance.

Main findings. The proposed architecture sustains real-time inference across heterogeneous ML modules with up to fourfold higher throughput compared to standard multiprocessing. It simplifies experimentation, ML models can be replaced or retrained in Python without changing the low-level code, while maintaining deterministic timing during deployment. The project demonstrates that intelligent perception does not rely solely on model accuracy: the surrounding architecture can itself embody "intelligence", enabling ML-driven systems to operate efficiently and predictably in real time.

1.2 Paper Structure and Original Contributions

The research presented in this report advances the design and implementation of a modular perception framework for small-scale autonomous vehicles. The focus lies on the integration of modern techniques with resource-constrained runtime environments, enabling real-time detection and decision-making.

The first major contribution is the development of a fine-tuned object detection pipeline based on the YOLO 11s architecture. This model was adapted to the BDD100K dataset and later retrained on a handcrafted, domain-specific dataset derived from real sensor recordings. The result is a robust, lightweight detector capable of real-time inference performance.

The second contribution consists of the construction of a hybrid Python–Rust runtime framework that enables efficient parallel execution of multiple perception models. This architecture mitigates the computational cost of running separate detectors for pedestrians, traffic lights, and signs by leveraging concurrent scheduling and optimized memory transfer mechanisms.

The report is organized into eight main chapters:

- **Chapter 1** introduces the motivation, objectives, and overall context of the project.
- **Chapter 2** defines the scientific problem, detailing the challenges of real-time perception in autonomous driving and the rationale for using intelligent algorithms.
- **Chapter 3** reviews the current state of the art in visual perception and runtime optimization.
- **Chapter 4** describes the proposed perception architecture, including model selection, integration, and software design principles.
- **Chapter 5** presents the fine-tuning and evaluation results of the YOLO-based detectors, supported by quantitative metrics and qualitative analyses.
- **Chapter 6** provides a SWOT analysis, evaluating the strengths, weaknesses, opportunities, and threats associated with the developed system.

- **Chapter 7** discusses the philosophical, ethical, and social implications of AI-based perception and autonomous systems.
- **Chapter 8** concludes the work, highlighting the main findings and outlining potential directions for future research and development.

1.3 Team Roles and Skills

The development and implementation of the AutoVision system were distributed among the team members according to their specific areas of expertise:

David Szilagyi

Role: Lead Machine Learning Engineer & Python Specialist

Responsibilities: Developed the core YOLOv11s perception pipeline, including fine-tuning and domain adaptation. Implemented the custom geometric lane detection algorithm and managed the Python-side integration with the shared memory architecture.

Skills: Python, PyTorch, Ultralytics YOLO, OpenCV, Computer Vision, Data Engineering.

Razvan Filea

Role: Systems Architect & Rust Engineer

Responsibilities: Designed and implemented the high-performance `rs_ipc` shared-memory communication layer and the Partial-Blocking scheduling algorithm. Also developed the native Rust visualization UI (`rs_ui`) using the Iced framework.

Skills: Rust, Systems Programming, IPC (Shared Memory), Concurrency, Iced (GUI), FFI (PyO3).

Armin Torok

Role: Frontend Developer & Web Interface Engineer

Responsibilities: Built the modern web-based monitoring interface using Next.js and React. This frontend provides a flexible and accessible way to visualize pipeline status and metadata remotely.

Skills: TypeScript, Next.js, React, TailwindCSS, Web Development, UX/UI Design.

1.4 Project Timeline

The development lifecycle of the XLR8 Hybrid Perception System spanned from January 2025 to January 2026, encompassing research, initial development, domain adaptation, and final system integration. The detailed timeline is illustrated in Figure 1.1.

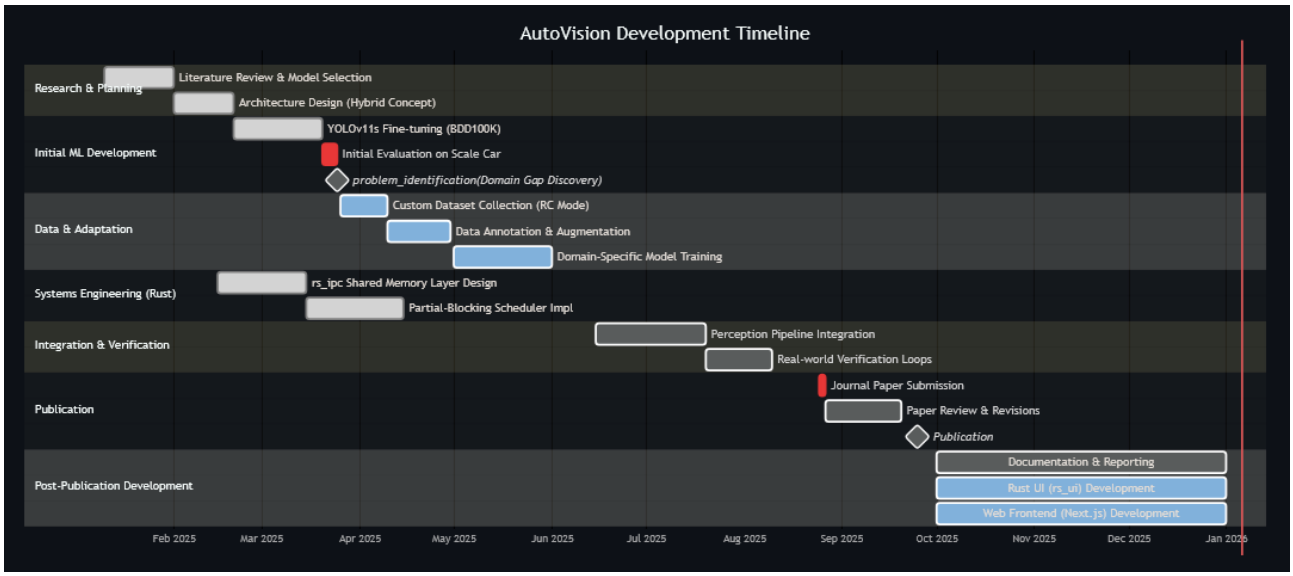


Figure 1.1: Gantt chart illustrating the development timeline, including the domain adaptation phase and post-publication UI development.

Chapter 2

Scientific Problem

2.1 The Need for Intelligent Perception under Latency Constraints

Problem Definition and Environmental Variability. The core challenge addressed in this project is the design of a perception system capable of understanding the driving environment of an autonomous vehicle in real time. Unlike controlled industrial settings, the driving domain is fundamentally unstructured: lighting changes, weather conditions, dynamic occlusions, and variable road textures make static, rule-based algorithms (such as classical thresholding or Hough transforms) insufficient.

To handle this variability, the system must employ adaptive, data-driven methods, specifically Deep Neural Networks (DNNs). In this work, we utilize YOLO models trained on handcrafted datasets derived from real onboard sensor data. These models allow the vehicle to recognize complex semantic cues (traffic lights, pedestrians, signs) that cannot be modeled explicitly.

The Operational Dilemma: Complexity vs. Latency. However, adopting deep learning introduces a critical conflict. To maximize semantic understanding, models require depth and high-resolution input, which linearly (or quadratically) increases computational cost. In a standard sequential pipeline (Capture $\rightarrow$ Inference $\rightarrow$ Control), the system's reaction speed is bottlenecked by the slowest component, usually the DNN inference. If a detector operates at only 10 Hz, the entire vehicle reacts at 10 Hz, which is insufficient for emergency braking at speed. Asynchronous multiprocessing strategies can decouple these rates, but they introduce non-deterministic latency and data serialization overheads that are difficult to bound in safety-critical contexts.

The scientific problem is therefore not just "training a better model," but constructing a runtime environment where *heavy* semantic tasks can coexist with *fast* reactive loops without compromising safety.

2.2 Architectural Challenges in Hybrid Systems

The "Two-Language" Perception Gap. A key aspect of this research is bridging the gap between research flexibility and deployment efficiency.

- **Python** is dominant in AI research (PyTorch, Ultralytics) due to its ease of use and rich ecosystem, but its Global Interpreter Lock (GIL) and garbage collection make it unsuitable for precise real-time orchestration.
- **Rust** offers memory safety and zero-cost abstractions, making it ideal for the low-level systems logic, but it lacks the rapid prototyping capabilities of Python for ML.

Synchronization Variance and Significance. Integrating these two paradigms creates a "Producer-Consumer Variance" problem. The camera produces frames at a fixed rate, while Python-based ML models have stochastic inference times depending on scene complexity. This project proposes a Partial-Blocking hybrid architecture to solve this. By managing synchronization in shared memory, the system ensures that the vehicle always acts on the freshest *completed* perception state. This approach is scientifically significant because it demonstrates that intelligent perception does not rely solely on model accuracy; the surrounding architecture itself must be optimized to allow ML models to function within strict real-time boundaries.

Chapter 3

State of the art/Related work

Research on autonomous vehicle perception has evolved along two major directions: (1) improving learning models for visual understanding, and (2) designing runtime architectures that enable those models to operate under real-time constraints. However, most studies emphasize one aspect while treating the other as secondary, resulting in high-accuracy models that are difficult to deploy efficiently, or optimized systems relying on simplified inference pipelines. Our work bridges this gap by integrating full-precision deep learning models into a framework optimized for low-latency, concurrent processing.

On the perception side, Rosique et al. [6] survey sensor modalities and vision architectures for autonomous driving, while Gu et al. [2] and Shahian Jahromi et al. [8] propose multi-sensor fusion pipelines designed for responsiveness and robustness. Ros et al. [5] highlight hybrid offline–online perception paradigms that improve reliability across varying conditions. These contributions advance detection and fusion accuracy but assume middleware that impose non-negligible communication costs.

System-oriented research, by contrast, addresses those architectural limitations. Early embedded vision works such as Sridharan and Stone [9] examined the trade-off between latency and throughput on constrained platforms, and frameworks like ROS 2 [4] and RViz [3] exemplify distributed modularity and built-in visualization. Yet, these solutions often rely on serialized message passing that limits scalability for high-frame-rate video and multiple concurrent models.

Recent efforts in multi-language and hybrid runtimes aim to merge high-level ML workflows with low-level efficiency. PyO3 [1] demonstrates a robust interface between Python and Rust, enabling flexible development with near-native performance. Scott and Brown [7] establish theoretical foundations for shared-memory synchronization, which inform modern approaches to concurrent data exchange. Building on these principles, our architecture embeds YOLO-based perception models directly within a partial-blocking, shared-memory environment, thereby combining the flexibility of ML research code with the determinism of optimized real-time systems.

Chapter 4

Investigated approach

4.1 Overview

We implement a modular perception stack where multiple *YOLO* models (traffic sign, light and pedestrian detection) and a lane–geometry detection module run in parallel as independent branches. Python hosts the ML code (model loading, pre/post-processing), while a Rust-backed shared–memory layer, called *SharedMessage* [10] coordinates data exchange and synchronization.

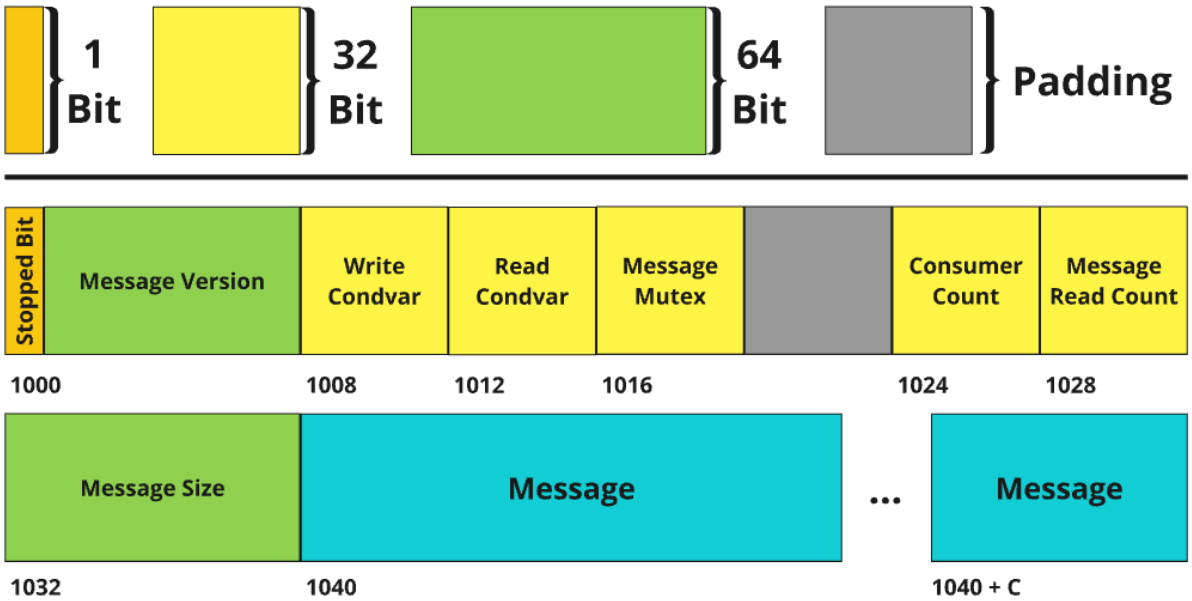


Figure 4.1: Memory layout of `SharedMessage`: fields are aligned to enable atomic access to *Message Version*, while the *Mutex* and condition variables reside in shared memory.

4.2 ML models and data pipeline

- **Models.** YOLO variants, fine-tuned for traffic sign, traffic light, pedestrian/object detection. Lane detection uses a custom geometric algorithm based on edge detection and Hough transforms, which in our testing proved faster and more reliable than learning-based lane segmentation models.
- **Data.** Handcrafted datasets from RC-mode onboard videos, processed by an augmentation & labeling pipeline to increase diversity and accuracy.
- **Inference I/O.** Each branch consumes the latest RGB frame f_t; produces task-specific predictions (boxes/classes for objects/signs; line/curve params for lanes) plus timestamps.

4.3 Producer–Consumer Model

A key component of the perception runtime is the **Producer–Consumer model**, which governs the data flow between the frame capture process (producer) and the perception branches (consumers). Depending on timing and synchronization requirements, the system supports three strategies:

- **Non-blocking:** The producer continuously publishes new frames without waiting for any consumer. Consumers process the most recent frame independently. This mode offers maximum responsiveness but may lead to desynchronized outputs, suitable for live monitoring or non-critical visualization.
- **Full-blocking:** The producer waits until all consumers have finished processing the current frame before publishing the next. This guarantees frame-level consistency across all modules but reduces throughput. It is mainly used in testing, debugging, or analysis scenarios.
- **Partial-blocking (proposed):** The producer may advance once at least one consumer has *started* processing the current frame. This minimizes idle time while ensuring every frame is processed by at least one consumer. It achieves a balance between responsiveness and consistency, ideal for real-time autonomous driving, where some ML branches (e.g., lane tracking) must update faster than others (e.g., sign recognition).

4.3.1 Partial-Blocking Algorithm

Algorithm 1 Partial-Blocking Producer–Consumer Perception Runtime

Variables / Notation
$F = \{f_t\}$: camera frame stream; n: number of consumer branches
$C = \{c_1, \ldots, c_n\}$: consumers (e.g., *objects*, *signs*, *lanes*)
$v \in \mathbb{Z}_{\geq 0}$: atomic version of the current raw frame in shared memory
f: frame written by Producer for version v
$\mathcal{M}_i$: ML model used by branch c_i; y_i: predictions produced by c_i for version v
v_i: version tag attached to y_i; v_{seen}: last version processed by a given consumer
$v^* = \max(v_1, \ldots, v_n)$: latest version available across all branches
$\Delta_i = v^* - v_i$: per-branch staleness (version gap)
P_t: fused perception state at time t;
{— Producer process —}
$v \leftarrow 0$
while running **do**
  $f \leftarrow$ ReadCamera()
  WriteShared(f, v)
  NotifyConsumers()
  if AtLeastOneConsumerStarted(v) **then**
    $v \leftarrow v + 1$ {Partial-Blocking advance criterion}
  end if
end while
{— Consumer process c_i —}
$v_{\text{seen}} \leftarrow -1$
while running **do**
  $(v, f) \leftarrow$ ReadLatest() {Non-blocking: may return same v as before}
  if $v \neq v_{\text{seen}}$ **then**
    $v_{\text{seen}} \leftarrow v$
    $y_i \leftarrow$ Infer($\mathcal{M}_i, f$)
    SignalStarted(v)
    WriteResult(c_i, v, y_i)
  end if
end while
{— Manager process —}
while running **do**
  $\{(v_i, y_i)\}_{i=1}^{n} \leftarrow$ CollectLatest(C) {v_i: version from branch i, y_i: its predictions}
  $v^* \leftarrow \max(v_1, \ldots, v_n)$
  $\Delta_i \leftarrow v^* - v_i \quad \forall i \in \{1, \ldots, n\}$
  $P_t \leftarrow$ MergeLatest($y_1, \ldots, y_n$)
  ResolveCollisions(P_t, *policy*) {e.g., confidence-first, IoU-NMS, or prefer newer v_i}
  Publish(P_t, v^*, Δ)
end while

Behavior summary. Under partial-blocking scheduling, the producer maintains constant frame flow while ensuring that every frame is processed by at least one consumer. Faster branches (e.g., lane detection) publish frequent updates, while slower ones catch up asynchronously. The manager merges

the most recent outputs from all perception pipelines, retaining the latest version available for each branch.

4.4 Worked Example (Trace)

To illustrate how the implemented partial-blocking scheduling behaves in practice, consider three consumer branches: c_1 (objects), c_2 (signs), and c_3 (lanes). Their respective average inference times are:

Objects: 12 ms Signs: 28 ms Lanes: 16 ms Capture: 10 ms

1. At t=0, the PRODUCER captures frame f_{10} and writes it to shared memory with version v=10. All branches are notified and begin reading from the shared buffer.

2. The fastest branch, c_1 (objects), immediately signals that it has started processing version v=10. This satisfies the partial-blocking condition (*at least one consumer started*), allowing the producer to proceed and capture the next frame f_{11} with v=11 after 10 ms.

3. Meanwhile, c_1 publishes its predictions (v_1=10, y_1=O_{10}) after 12 ms, c_3 (lanes) publishes (v_3=10, y_3=L_{10}) after 16 ms, and c_2 (signs), being the slowest, completes (v_2=10, y_2=S_{10}) only after 28 ms. By this time, the producer has already captured f_{11} and possibly f_{12}, so the current global version counter is v=12.

4. c_1 finishes processing f_{11} and publishes (v_1=11, y_1=O_{11}) before c_2 even finishes version 10. c_3 completes (v_3=11, y_3=L_{11}) shortly after. At this point, the manager collects the latest results from all branches:

$$(v_1, y_1) = (11, O_{11}), \quad (v_2, y_2) = (10, S_{10}), \quad (v_3, y_3) = (11, L_{11}).$$

5. The manager determines the newest version v^*=11 and computes the staleness vector Δ=(v^*-v_1, v^*-v_2, v^*-v_3) = (0, 1, 0). It merges the latest predictions using MERGELATEST(O_{11}, S_{10}, L_{11}) and applies optionally applies collision handling where overlapping detections exist (e.g., by choosing the higher-confidence or newer result). The fused perception state P_t is then published as the system's current output.

Chapter 5

Application (numerical validation)

5.1 Methodology

Reliable traffic sign and traffic-light detection forms a core element of the perception pipeline in autonomous and semi-autonomous driving systems. These visual cues provide essential semantic information for high-level decision-making tasks such as intersection handling, right-of-way interpretation, and trajectory planning.

To achieve robust detection performance, this work first explored adapting a state-of-the-art, pretrained detector to the project's domain through targeted fine-tuning. The initial approach focused on the YOLO 11s architecture, which combines a lightweight backbone with efficient spatial feature aggregation, making it suitable for deployment on embedded hardware. The model was fine-tuned on the BDD100K dataset, a large-scale benchmark for real-world driving scenarios. The motivation was to leverage its diverse and well-annotated scenes to improve generalization across varying traffic, illumination, and environmental conditions.

However, when transferred to the project's small-scale physical environment, the resulting detector exhibited reduced reliability due to substantial domain gaps in perspective, object scale, and scene illumination.

These limitations motivated a second, domain-specific approach: pretraining YOLO on a handcrafted dataset captured directly from the vehicle's onboard sensors within the real test environment, followed by targeted augmentation and refinement.

The following sections detail these two complementary strategies, beginning with the fine-tuning of YOLO 11s on the BDD100K dataset, and subsequently the design and use of the custom sensor-based dataset for specialized adaptation.

5.2 Fine-Tuning and Evaluation of the YOLO 11s Detector

5.2.1 Dataset Description

The **Berkeley DeepDrive 100K (BDD100K)** dataset [11] is one of the largest open benchmarks for road-scene understanding. It contains over 100,000 driving video clips with frame-level annotations across a wide range of weather, lighting, and geographic conditions. Each frame is labeled for multiple perception tasks including object detection, lane marking, and drivable area segmentation.

For this work, only the *detection* subset was used, containing image-level annotations for 10 object categories such as cars, buses, trucks, traffic lights, and traffic signs. T he dataset was reformatted into the YOLO file structure, and three disjoint splits were created:

- 7,500 images for training,
- 2,000 images for validation,
- 500 images for testing.

The focus was restricted to the categories *traffic light* and *traffic sign*, as these objects are the most relevant for the project's control and planning modules.

5.2.2 Training Configuration

Fine-tuning was performed using the Ultralytics YOLO 11s implementation, initialized with weights pretrained on the COCO 2017 dataset. The full model was trained end-to-end for 20 epochs. The most relevant settings are summarized in Table 5.1.

Parameter	**Value**
Base model	`yolo11s.pt` (COCO pretrained)
Dataset	BDD100K detection subset (traffic lights/signs)
Input resolution	640×640
Batch size	8
Epochs	20
Optimizer	AdamW
Learning rate	7.1×10^{-4} (cosine schedule)
Weight decay	5×10^{-4}
Augmentation	Color jitter, translation, scaling, horizontal flip
Precision mode	FP32 (AMP disabled)
Hardware	NVIDIA RTX 3090 (24 GB)

Table 5.1: Final YOLO 11s fine-tuning configuration.

5.2.3 Dataset Distribution

Figure 5.1 provides an overview of the class composition and bounding-box statistics for the fine-tuning subset. As expected for a driving dataset, the majority of annotated instances correspond to vehicles, which dominate both the frequency histogram and the spatial density map. The bounding-box center distribution shows a concentration near the image center, reflecting ego-vehicle-aligned scenes where most cars appear in the forward direction.

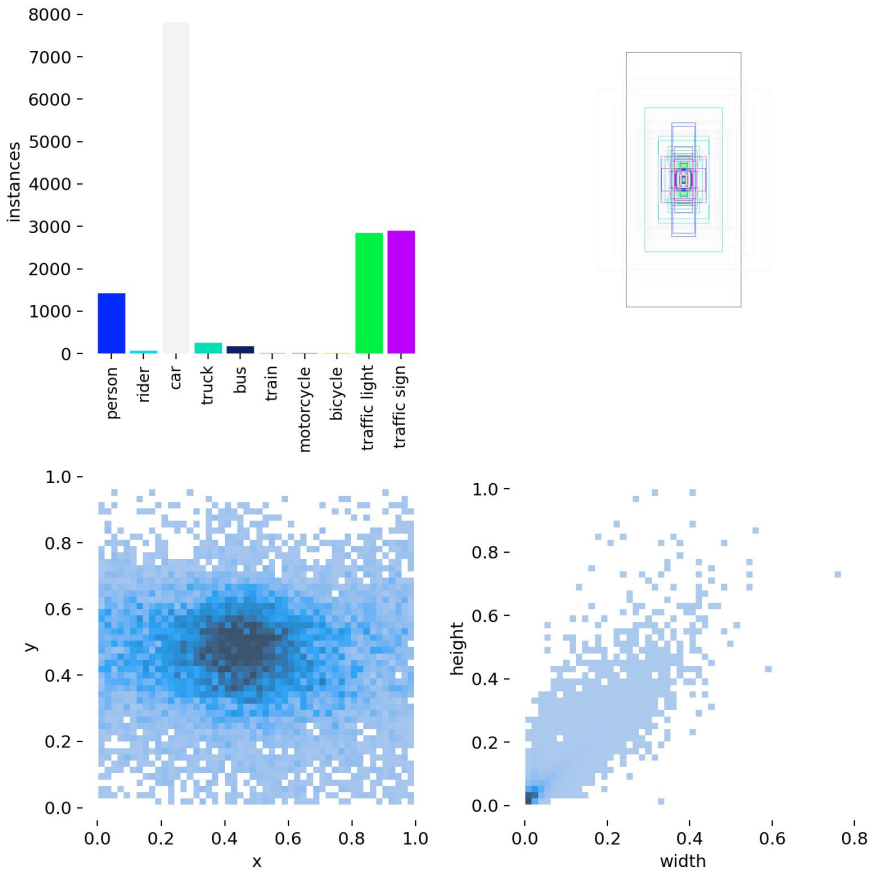


Figure 5.1: Label frequency and bounding-box spatial distribution for all classes in the BDD100K fine-tuning subset.

5.2.4 Training Loss Evolution

Figure 5.2 summarizes the evolution of the primary loss components during the 20 training epochs. The bounding-box regression (box_loss), classification (cls_loss), and distribution focal (dfl_loss) losses all exhibit a steady decrease throughout the first fifteen epochs, indicating stable convergence of both spatial and semantic learning. A mild spike is visible in the final epochs across all loss components, which can be attributed to the temporary removal of mosaic augmentation near the end of training

(`close_mosaic=5`). This shift briefly alters the effective data distribution, slightly increasing training loss before stabilization. The validation curves, however, remain smooth and continue to improve, confirming that no overfitting occurred and that the model generalized well within the given training budget.

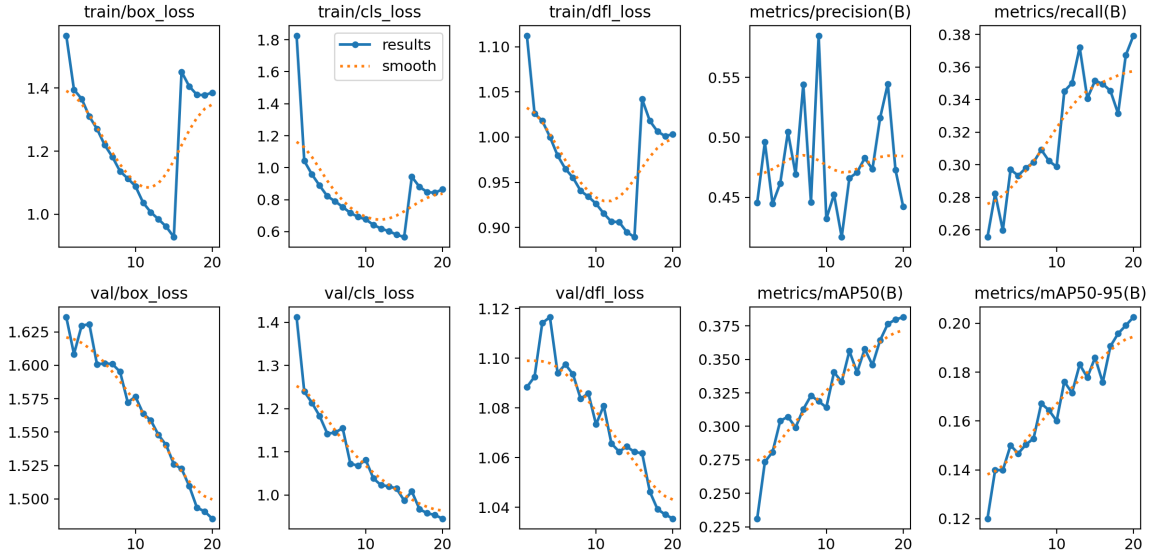


Figure 5.2: Training and validation loss evolution along with precision, recall, and mAP metrics across epochs. The late-epoch fluctuation corresponds to mosaic augmentation being disabled.

5.2.5 Evaluation Results

The fine-tuned YOLO 11s model achieved a mean average precision (mAP@0.5) of approximately **0.38** on the validation split, maintaining stable detection performance across varying illumination and viewing angles. The precision–recall characteristics (Figure 5.3) show that precision remains above 0.4 for recall levels up to 0.8, while the F1 curve (Figure 5.4) indicates an optimal operating confidence threshold around **0.23**. Overall, the model demonstrates reliable real-time detection of traffic lights and signs within the BDD100K domain, with slightly reduced recall for small, distant instances.

Table 5.2 provides a class-wise summary of the quantitative results. As expected, vehicles (particularly cars) achieve the highest detection accuracy due to their abundance and larger bounding-box areas, while traffic lights and signs remain more challenging targets. These results nonetheless confirm the benefit of domain-specific fine-tuning compared to generic COCO-trained baselines.

Class	Precision	Recall	F1	mAP@0.5
Person	0.41	0.64	0.49	0.41
Car	0.90	0.74	0.81	0.67
Truck	0.57	0.48	0.52	0.30
Bus	0.66	0.54	0.59	0.43
Traffic Light	0.53	0.39	0.45	0.38
Traffic Sign	0.56	0.37	0.45	0.40
Mean	**0.60**	**0.53**	**0.54**	**0.38**

Table 5.2: Per-class detection metrics computed on the BDD100K validation split.

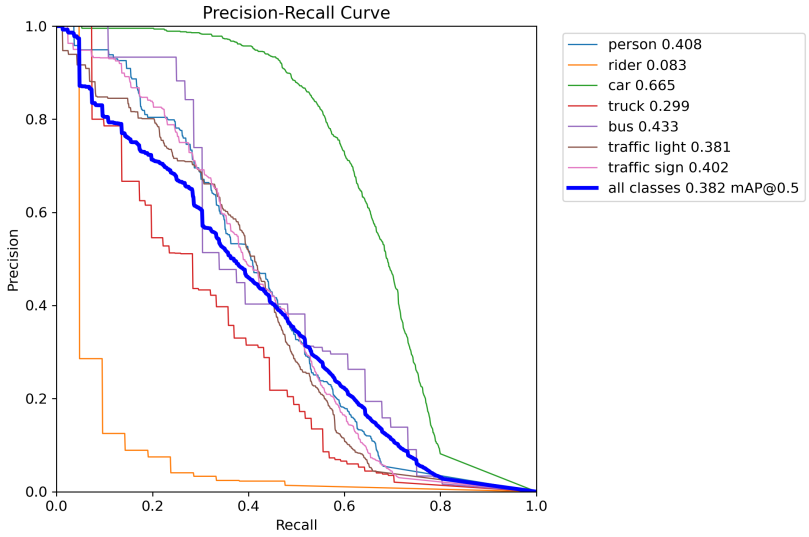


Figure 5.3: Precision–recall curve of the fine-tuned YOLO 11s detector.

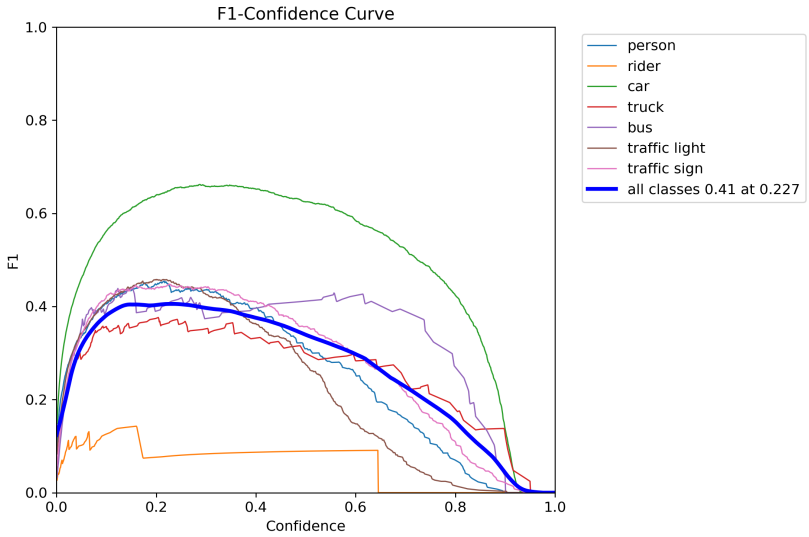


Figure 5.4: F1-score versus confidence threshold.

5.2.6 Error Analysis

The confusion matrices in Figure 5.5 reveal that the primary misclassifications occur between *traffic light*, *traffic sign*, and background regions. These errors are mainly caused by distant objects and low-contrast lighting conditions, leading to partial detections or missed small-scale targets. Nevertheless, the normalized confusion matrix confirms a high true-positive ratio and minimal cross-category leakage.

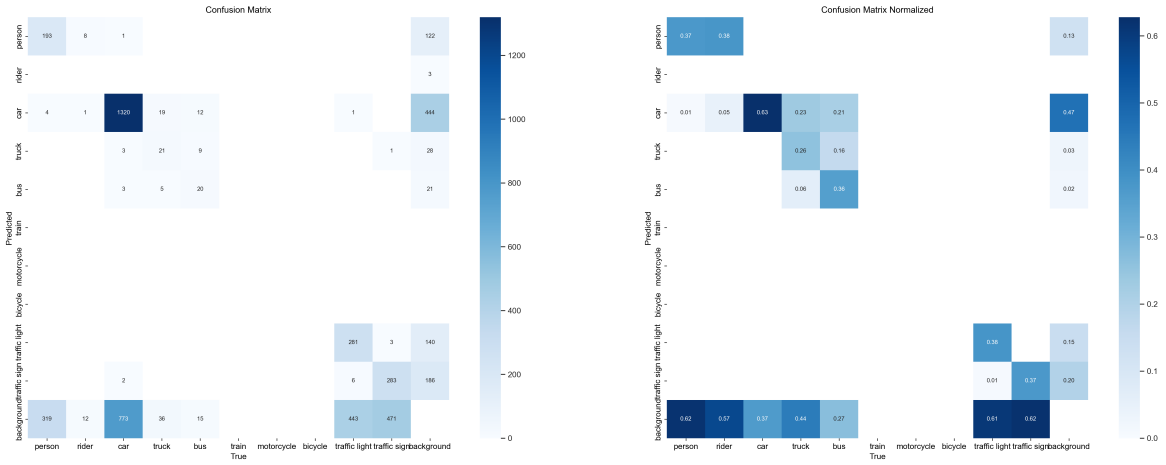


Figure 5.5: Confusion matrices of the fine-tuned YOLO 11s model.

5.2.7 Discussion

The fine-tuned YOLO 11s model performed reliably on the BDD100K test split, achieving accurate detection of traffic lights and signs across diverse driving conditions. Figure 5.6 illustrates representative inference results, where the model successfully localizes multiple traffic lights, even under challenging illumination and partial occlusion. These results confirm that the fine-tuning process effectively adapted the pretrained COCO features to the BDD domain, enabling stable real-time performance on urban imagery.

When evaluated in the scaled physical environment, however, the detector's behavior diverged markedly from its performance on the BDD dataset (Figure 5.7). The model frequently misidentified background elements such as chairs, posters, or fire extinguishers as traffic signs or lights, while overlooking the actual miniature props used in the scene. This degradation stems not from scale alone but from the fundamental visual mismatch between the real-world urban imagery of BDD100K and the toy-like textures, shapes, and color contrasts present in the scaled setup. Objects such as printed paper traffic lights or plastic figurines bear little resemblance to the materials, geometry, and illumination patterns on which the network was originally trained. Consequently, the learned visual features fail to activate meaningfully in this artificial domain, leading to a high rate of false positives and missed

Figure 5.6: Qualitative detection results on the BDD100K test set using the fine-tuned YOLO 11s model.

detections.

Since reliable detection serves as a prerequisite for any downstream classification or semantic reasoning, the observed instability rendered subsequent traffic-light and sign classification infeasible. Consequently, the following section investigates a more targeted approach, retraining the detector using a handcrafted dataset collected from the real vehicle's onboard sensors.

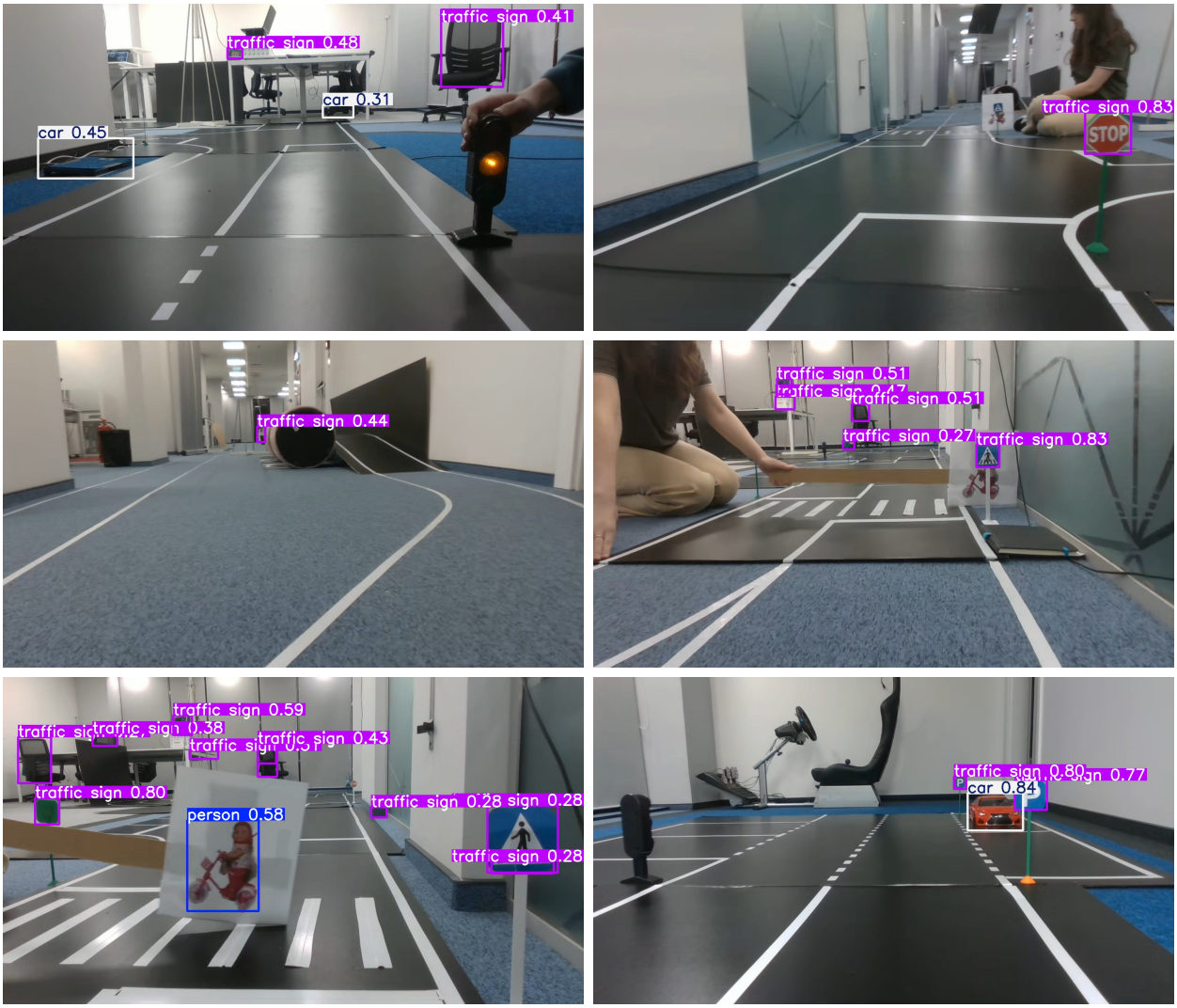


Figure 5.7: Detection performance in the real test environment.

5.3 Pretraining on a Domain-Specific Dataset

5.3.1 Motivation and Dataset Construction

Given the severe visual mismatch between the BDD100K dataset and the scaled laboratory environment, a new dataset was assembled directly from the perception data of the miniature test vehicle. The objective was to provide the detector with domain-relevant visual cues, such as lighting reflections, surface textures, and color contrasts,that were absent in large-scale real-world datasets.

To this end, several short driving sessions were recorded using the vehicle's onboard RGB camera under varying ambient light conditions. Individual frames were extracted and manually annotated to include only the relevant categories observed in the test environment. For simplicity and improved supervision, traffic lights of different states (*red*, *amber*, *green*) were treated as separate classes, as

were distinct traffic sign types. This formulation eliminated the need for a secondary classification step while maintaining direct semantic interpretability at detection time.

The resulting dataset was expanded through targeted augmentations including brightness jitter, Gaussian noise injection, random rotations, and mild perspective distortion, to simulate the diversity of viewpoints and lighting conditions encountered in operation. An example of the annotated training data is shown in Figure 5.8.

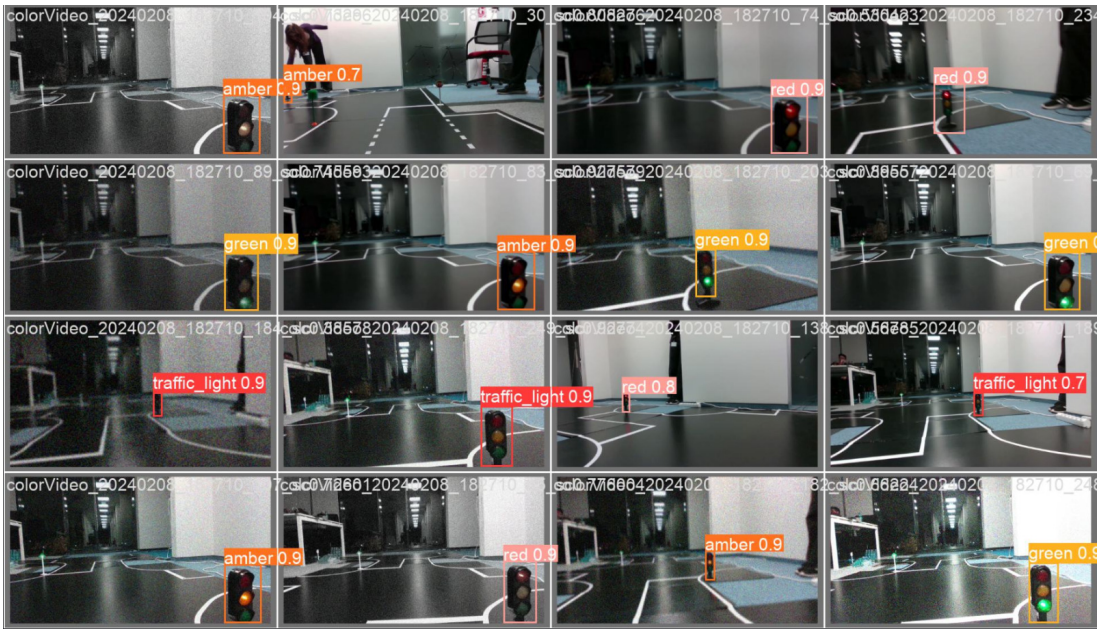


Figure 5.8: Representative sample from the handcrafted training dataset, captured from the onboard camera.

5.3.2 Model Specialization Strategy

Rather than training a single monolithic detector to recognize all object categories simultaneously, the perception stack employs three specialized YOLO 11s models: one for pedestrian detection, one for traffic signs, and one for traffic lights. This design decision was motivated by the strong visual and geometric disparities between these object types in the scaled environment.

Traffic lights are small, high-contrast, and color-dependent; traffic signs are planar and textural; pedestrians, on the other hand, are articulated and comparatively large. A unified model would therefore require compromises in input resolution, anchor configuration, and augmentation parameters, often leading to degraded performance on the most challenging classes. By contrast, class-specific models allow for independent tuning of hyperparameters and augmentation pipelines, enabling higher precision with smaller network backbones.

The computational overhead of running three detectors in parallel was mitigated by the hybrid

processing architecture discussed in previous chapters, which distributes inference across dedicated CPU and GPU threads. This ensures near real-time performance while preserving the benefits of specialization. As a result, each model can operate within its optimal domain, achieving improved per-class accuracy without compromising overall system responsiveness.

5.3.3 Qualitative Evaluation

Inference results, shown in Figure 5.9, indicate that the network successfully adapted to the visual characteristics of the scaled environment. Traffic lights are consistently detected and correctly classified according to their color state, even under moderate illumination changes or motion blur. The detections align with the physical geometry of the scene and exhibit significantly fewer spurious activations than the BDD-trained model.

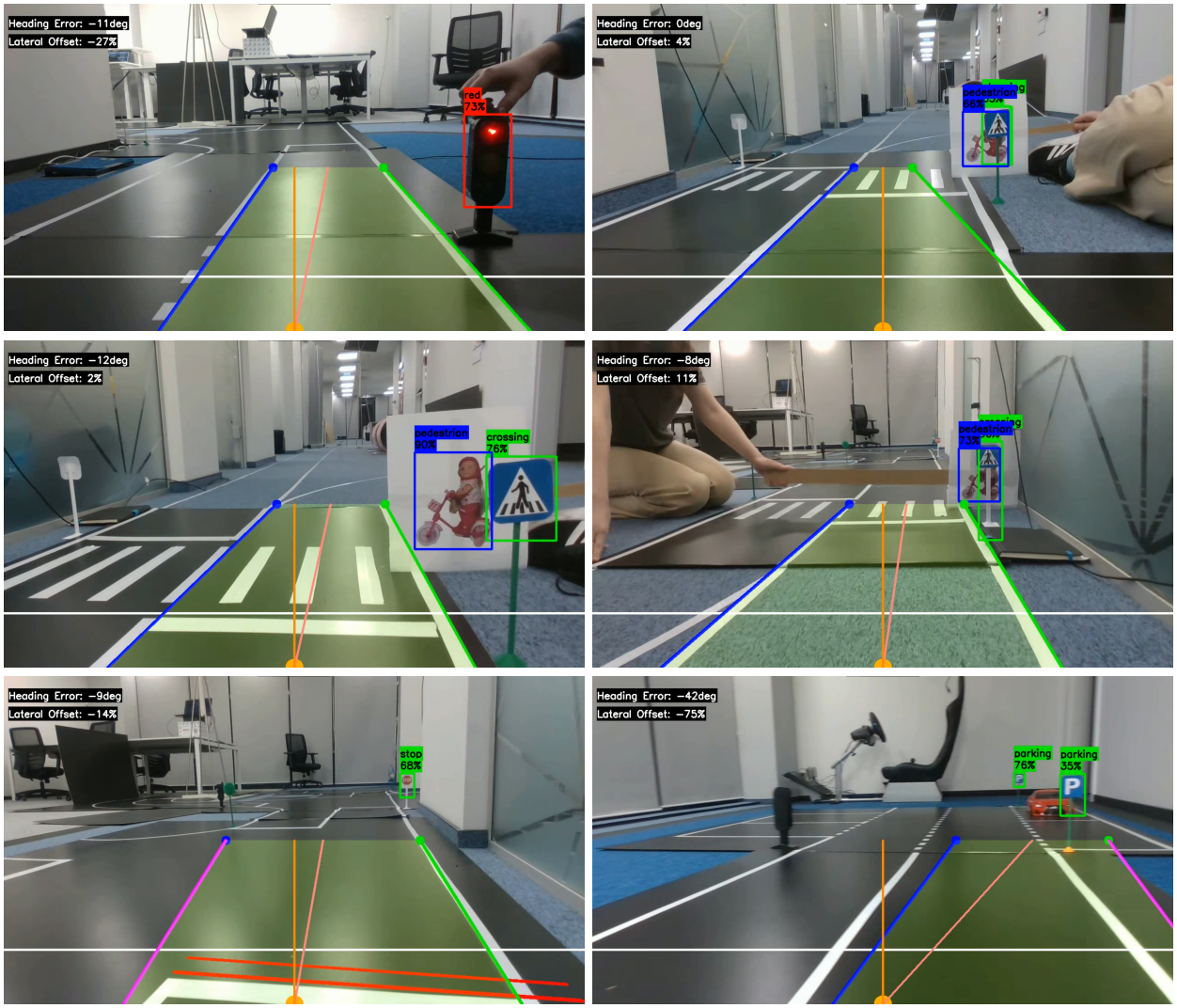


Figure 5.9: Qualitative detections in the scaled test environment after pretraining on the handcrafted dataset.

Chapter 6

SWOT Analysis

The following section summarizes the key strengths, weaknesses, opportunities, and threats identified throughout the development and evaluation of the perception framework. It considers not only model performance but also hardware integration, scalability, and maintainability aspects within the broader context of autonomous and semi-autonomous vehicle systems.

6.1 Strengths

- Modular perception pipeline with easy extensibility and independent model updates.
- Robust object detection performance achieved through YOLO fine-tuning and domain-specific pretraining
- High inference efficiency enabled by the hybrid Python–Rust architecture and parallelized model deployment.
- Successful domain adaptation through custom dataset pretraining, achieving robust detection in the scaled test environment.
- Real-time performance on commodity hardware without dependency on external cloud computation.

6.2 Weaknesses

- Limited generalization to unseen or non-laboratory environments due to restricted dataset diversity.

- Manual annotation of training samples remains time-intensive and prone to subjective bias.
- Lack of unified cross-model calibration, requiring additional synchronization between the three specialized detectors.
- Absence of long-term stability testing under continuous operation.

6.3 Opportunities

- Expansion of the handcrafted dataset with synthetic and simulation-generated imagery to improve robustness.
- Integration of lightweight semantic segmentation for contextual awareness and lane-level reasoning, replacing the geometric lane detection module.
- Exploration of model compression techniques (e.g., quantization, pruning) to further reduce inference latency.
- Adoption of emerging vision-language models (VLMs) for enhanced scene understanding and zero-shot recognition capabilities.

6.4 Threats

- Rapid evolution of deep learning architectures may render the current YOLO-based approach outdated without continuous adaptation. This is already probably the case with the rapid adoption of VLM/VLA architectures.
- Linux-specific optimizations limit portability across different systems.

Chapter 7

Philosophical, Ethical, and Social Implications

7.1 Social Impact of AI-Based Perception

Although this project is evaluated on a scaled autonomous vehicle platform, the developed perception architecture directly reflects principles used in full-scale autonomous driving. As such, its implications extend beyond the laboratory environment.

Real-time perception systems shift situational awareness from human drivers to algorithmic pipelines. In classical driving, perception errors are interpreted and corrected by human judgment. In autonomous systems, however, perceptual errors directly affect planning and control. This redistribution of responsibility fundamentally changes the structure of accountability, as decisions emerge from layered technical systems rather than from a single human agent.

In the long term, perception-driven automation also reshapes the labor market. While new highly specialized engineering roles appear, traditional driving-related professions face gradual displacement. This contributes to technological polarization between high-skill development roles and disappearing routine occupations.

In addition, autonomous perception normalizes large-scale visual data acquisition. Even though this project does not process personal data, the same technologies are used in smart cities, traffic monitoring, and surveillance systems, raising concerns related to privacy erosion and the growing societal acceptance of continuous algorithmic observation.

7.2 Ethics of Automated Perception

The perception pipeline developed in this work forms the sensory basis for autonomous decision-making. Ethical concerns therefore arise already at the perception level, prior to any explicit control actions.

A central ethical issue is responsibility attribution. If a detector fails to recognize a pedestrian or traffic light, the failure is not a single human mistake but the outcome of interacting technical choices: dataset construction, annotation policies, model architectures, optimization goals, and runtime scheduling. Responsibility is thus structurally distributed across multiple roles, making accountability inherently fragmented.

Perception errors are probabilistic in nature. The system relies on confidence thresholds and statistical approximations rather than certainty. Lower thresholds improve sensitivity but increase false positives, while higher thresholds reduce false detections at the risk of critical misses. These thresholds are not purely technical parameters but implicit ethical decisions about acceptable operational risk.

The use of handcrafted datasets further introduces subjectivity. Manual annotation inevitably reflects human interpretation, which may bias the learned models toward specific visual patterns, lighting conditions, or object appearances.

7.3 Objectivity, Reliability, and Trust

Machine learning systems are often perceived as objective due to their mathematical formulation. In practice, however, objectivity is limited by the subjectivity embedded in data selection, labeling, and model design.

The strong domain gap observed between the BDD100K-trained model and the handcrafted dataset demonstrates that the model does not learn universal visual truth. Instead, it learns statistical regularities bound to the conditions of its training environment. What the system perceives as “reality” is therefore a data-dependent approximation.

The perception system does not possess semantic understanding. It correlates pixel patterns with learned labels. The indoor misclassification of objects as traffic signs directly illustrates the limits of this statistical perception. Errors arise not from flawed reasoning, but from missing relevant visual experience.

Trust in such systems therefore cannot rely solely on detection accuracy. It must be grounded in continuous validation, domain adaptation, redundancy, and system-level safeguards. While the hybrid Python–Rust runtime improves execution determinism and temporal reliability, it cannot eliminate the uncertainty inherent in data-driven perception.

7.4 Developer Accessibility and Ethical Abstraction

A core design goal of the proposed hybrid Python–Rust architecture is to enable developers with strong machine learning expertise to deploy real-time perception systems without requiring deep knowledge of low-level infrastructure.

By keeping model development, training, and inference fully within the Python ecosystem, the system allows developers to focus on data, models, and evaluation instead of memory management, synchronization, or real-time scheduling. These low-level responsibilities are delegated to the Rust-based runtime layer.

This abstraction has clear ethical benefits. It lowers the barrier of entry to autonomous systems research, supports faster academic experimentation, and democratizes access to real-time AI deployment beyond specialized industrial environments.

At the same time, abstraction introduces ethical risk. When developers are shielded from infrastructure behavior, they may lose awareness of hidden latency effects, synchronization delays, or rare failure modes. Safety can shift from explicit verification to implicit trust in the runtime layer.

Philosophically, the architecture reflects a transition from low-level optimization as the primary gatekeeper of innovation toward conceptual and data-centric competence. Ethical responsibility is redistributed rather than removed.

7.5 Limits of AI Autonomy

The developed perception system operates autonomously at the sensory level: it captures, processes, and publishes environmental information without direct human intervention. However, this autonomy remains strictly functional.

The system can optimize detection speed and accuracy, but it cannot evaluate the moral significance of its errors. It does not distinguish ethically between a missed pedestrian and a missed traffic sign—both appear as numerical deviations in an optimization process.

For this reason, full moral responsibility must always remain external to the system, with human designers, operators, and regulatory institutions. Autonomous perception does not imply autonomous moral agency. The system can assist and accelerate perception, but it cannot replace human ethical judgment.

Chapter 8

Conclusion and Future Work

This work presented a modular perception framework for traffic element detection and classification in both simulated and real-world scaled environments. The approach combined pretrained YOLO 11s detectors with targeted fine-tuning, followed by domain-specific pretraining on a handcrafted dataset derived from the vehicle's onboard camera. The resulting system demonstrated that lightweight neural architectures can achieve stable and real-time performance when properly adapted to their operational domain.

Summary of Findings

The experiments confirmed that fine-tuning on the BDD100K dataset yields reliable traffic-light and traffic-sign detection under realistic urban conditions but fails to generalize to non-standard, toy-scale environments. This limitation motivated the construction of a custom dataset tailored to the visual properties of the laboratory setup, resulting in significantly improved robustness and a marked reduction in false positives. Moreover, the decision to employ three specialized YOLO models, each optimized for pedestrians, signs, or lights, proved effective in balancing computational efficiency and task-specific accuracy when executed within the hybrid Python–Rust architecture.

Overall, the results illustrate that careful domain adaptation and architectural specialization can achieve a strong trade-off between performance and hardware constraints. The system fulfills real-time requirements on consumer-grade GPUs, maintains modularity for future extensions, and provides a reproducible baseline for small-scale autonomous vehicle research.

Limitations and Proposed Improvements

Despite its strengths, several shortcomings were identified:

- **Limited generalization.** The handcrafted dataset, though effective within the toy environment, remains narrow in scope. To enhance robustness, future work should incorporate synthetic data generation using simulation tools or domain randomization to emulate broader visual variability.
- **Manual annotation overhead.** The current labeling process is labor-intensive and susceptible to human inconsistency. Semi-automated annotation pipelines and active learning could significantly reduce this burden.
- **Independent model calibration.** Running three separate detectors requires manual threshold tuning and synchronization. A unified multi-head training framework or shared feature backbone could simplify deployment while preserving specialization.
- **Lack of temporal consistency.** All current detections are frame-based. Integrating lightweight temporal tracking or recurrent filtering would stabilize outputs and improve robustness under motion blur.

Outlook

The presented system establishes a solid foundation for future research in resource-constrained perception pipelines. Future extensions will focus on fusing additional sensory modalities, such as depth and inertial data, to enhance spatial reasoning, while exploring model compression and quantization for embedded deployment. Furthermore, the developed datasets and training procedures can support continued experimentation with federated or compute-aware learning paradigms, enabling scalable training across heterogeneous robotic platforms.

In conclusion, this work demonstrates that targeted fine-tuning, domain-specific pretraining, and architectural specialization can collectively yield a practical, efficient, and extensible perception system. These results not only validate the feasibility of deploying modern deep detectors on small-scale robotic platforms but also provide a blueprint for advancing real-world autonomous perception research.

Bibliography

[1] Maxwell Flitton. Speed up your python with rust: Optimize python performance by creating python pip modules in rust with pyo3. 2022.

[2] Yongxiang Gu, Qianlei Wang, and Xiaolin Qin. Real-time streaming perception system for autonomous driving. In *2021 China Automation Congress (CAC)*, pages 5239–5244, 2021.

[3] Hyeong Ryeol Kam, Sung-Ho Lee, Taejung Park, and Chang-Hun Kim. Rviz: a toolkit for real domain data visualization. *Telecommunication Systems*, 60:337–345, 2015.

[4] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science robotics*, 7(66):eabm6074, 2022.

[5] German Ros, Sebastian Ramos, Manuel Granados, Amir Bakhtiary, David Vazquez, and Antonio M Lopez. Vision-based offline-online perception paradigm for autonomous driving. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 231–238. IEEE, 2015.

[6] Francisca Rosique, Pedro J Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3):648, 2019.

[7] Michael Lee Scott and Trevor Brown. *Shared-memory synchronization*. Springer, 2013.

[8] Babak Shahian Jahromi, Theja Tulabandhula, and Sabri Cetin. Real-time hybrid multi-sensor fusion framework for perception in autonomous vehicles. *Sensors*, 19(20):4357, 2019.

[9] M. Sridharan and P. Stone. Real-time vision on a mobile robot platform. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2148–2153, 2005.

[10] Dávid Szilágyi, Răzvan Filea, and Kuderna-Iulian Bența. Accelerating intelligent vehicle vision: A hybrid python–rust architecture with partial-blocking inter-process communication. In *2025 RRIA, Romanian Journal of Information Technology and Automatic Control*, 2025. in press.

[11] Ye Xia, Danqing Zhang, Jinkyu Kim, Ken Nakayama, Karl Zipser, and David Whitney. Predicting driver attention in critical situations. In *Asian conference on computer vision*, pages 658–674. Springer, 2018.