Robust Multimodal Perception Stack for High-Speed Autonomous Racecars

Kaushik Singh

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2025-110
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-110.html

May 16, 2025

Acknowledgement

I would like to thank the entire AI Racing Tech team, especially the Perception Team, including Chris Lai, Tianlun Zhang, Lawrence Shieh, Edward Lee, Jiaming Zhang, Eric Berndt, Kevin Chow, Annabel Ng, Ashwat Chidambaram, Jeff Liu, and Timothy Park. The perception stack detailed in this thesis is a result of all of their hard work and contributions. I'm also grateful to our senior members, Siddharth Saha and Haoru Xue, whose foundational work underpins the complete software stack. My deepest thanks go to Dr. Allen Yang and Professor Shankar Sastry for their mentorship, guidance, and for giving me the opportunity to research autonomous racecars.

Robust Multimodal Perception Stack for High-Speed Autonomous Racecars

by Kaushik Kunal Singh

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor S. Shankar Sastry
Research Advisor

5-16-2025

(Date)

* * * * * * *

Dr. Allen Y Yang
Second Reader

5-16-2025

(Date)

Robust Multimodal Perception Stack for High-Speed Autonomous Racecars

by

Kaushik Kunal Singh

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor S. Shankar Sastry, Chair
Dr. Allen Yang, Co-chair

Spring 2025

Robust Multimodal Perception Stack for High-Speed Autonomous Racecars

Abstract

Robust Multimodal Perception Stack for High-Speed Autonomous Racecars

by

Kaushik Kunal Singh

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor S. Shankar Sastry, Chair

Dr. Allen Yang, Co-chair

Autonomous racing presents a uniquely constrained yet demanding testbed for perception systems: cars compete at high speeds on fixed circuits with known boundaries, but must reliably detect and track only their opponents under stringent real-time constraints. This thesis addresses the challenge of robust multi-modal perception for autonomous racecars by developing, analyzing, and experimentally validating modular fusion architectures that leverages LiDAR, radar, and camera sensors.

We begin by formulating the problem of opponent detection - estimating the two-dimensional position, orientation, and velocity of other vehicles - under assumptions of a separate localization system and a predefined track. After surveying classical and end-to-end learning approaches, we motivate a classical "early-stage" fusion pipeline based on perspective projection and extrinsic calibration, alongside a "late-stage" fusion design that independently processes each modality before combining outputs via an Extended Kalman Filter.

Preliminary experiments - benchmarked against transponder-derived ground truth - evaluate positional accuracy and computational load for both fusion methods. Results demonstrate promising indications that our late-stage fusion method achieves superior robustness to misclassification and miscalibration, while maintaining real-time performance on the racecar's onboard compute.

To my Family

Mom, Dad, Sachin, and Snoopy - thank you for your unwavering love and support. It means the world to me to have you by my side.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank the entire AI Racing Tech team, especially the Perception Team, including Chris Lai, Tianlun Zhang, Lawrence Shieh, Edward Lee, Jiaming Zhang, Eric Berndt, Kevin Chow, Annabel Ng, Ashwat Chidambaram, Jeff Liu, and Timothy Park. The perception stack detailed in this thesis is a result of all of their hard work and contributions. I'm also grateful to our senior members, Siddharth Saha and Haoru Xue, whose foundational work underpins the complete software stack.

My deepest thanks go to Dr. Allen Yang and Professor Shankar Sastry for their mentorship, guidance, and for giving me the opportunity to research autonomous racecars. If someone had told me a few years ago that I'd be working on this, I wouldn't have believed it. As a child I loved racecars, and my passion for engineering soon followed. Being able to combine both has truly been a dream come true.

Chapter 1

Introduction

1.1 Competitions in Autonomous Driving

Darpa Grand Challenge

In the early 2000s, the Defense Advanced Research Projects Agency (DARPA) launched the Grand Challenge, a pivotal event in the evolution of autonomous driving. The first competition in March 2004 tasked autonomous vehicles with navigating a 142-mile off-road course through the Mojave Desert, with a $1 million prize [37]. Fifteen vehicles participated, with retrofitted off-road vehicles (see 1.1), yet none finished. Despite this outcome, the event galvanized the autonomous driving community and inspired further research and innovation in self-driving technology.

These advancements became clear in October 2005, when DARPA held a second Grand Challenge, and out of 195 entrants, five vehicles completed a 132-mile desert route. Subsequently, DARPA raised the complexity with the Urban Challenge in November 2007, where autonomous vehicles had to navigate traffic scenarios and urban environments. Here, six of eleven finalists completed the challenge.

In just a few years, the challenges had gotten more difficult [6], yet more teams were successfully completing them. These ambitious competitions accelerated innovation in robotic perception, planning, and control, directly influencing practical applications in industries. They bridged academic research with real-world implementation, laying essential groundwork for today's autonomous vehicle industry. DARPA demonstrated that ambitious competition models could rapidly mobilize interdisciplinary teams, establishing a foundation that continues to influence specialized autonomous driving challenges today [3].

Figure 1.1: "Sandstorm" by Carnegie Mellon Red Team, DARPA Grand Challenge 2005

Indy Autonomous Challenge

More than a decade after the DARPA Grand Challenge, the autonomous systems community has turned to high-speed autonomous racing as a new frontier. The Indy Autonomous Challenge (IAC), launched in 2019 by the Energy Systems Network (ESN), is a prominent competition featuring university teams racing autonomous Dallara IL-15 chassis (see 1.2) at speeds up to 170 mph. The inaugural IAC race took place at the Indianapolis Motor Speedway in October 2021, marking the first autonomous oval-track race with multiple vehicles. The competition has since expanded to Texas Motor Speedway, Las Vegas Motor Speedway, and complex road courses such as Monza Speedway, Putnam Road Course, Las Vegas Road Course, and Laguna Seca Raceway.

With standardized hardware provided to all teams, the IAC emphasizes software innovation [4, 41, 32, 23], challenging participants to develop advanced perception and decision-making algorithms. Racing under real-world conditions at high speeds demands solutions to critical scenarios like obstacle avoidance and precise overtaking. Unlike the DARPA challenges, the IAC features a different vehicle platform and driving environment, presenting unique and complex technical challenges.

Figure 1.2: AI Racing Tech (ART) AV24, Indy Autonomous Challenge 2025

1.2 Motivation

In explaining my research to others, racing a "million-dollar car" autonomously, I'm often met with the same question: Why? Why push autonomy into the high-stakes world of motorsport?

At racing speeds, the margin for error vanishes: what feels like half a second to us is an eternity at 160 mph, and centimeter-level uncertainty can turn a textbook pass into a spin-out. Every component of the software stack is pushed to its limits. For perception, the challenges include, but aren't limited to:

- **Low-latency demands**: Perception and control loops must complete in tens of milliseconds to stay competitive.
- **High G-loads, heat, and vibration**: Sharp turns and engine/exhaust pulses constantly shift sensors, degrading calibration, introducing noise and dropout.
- **Rapidly varying lighting and track conditions**: Sun glare, deep shadows, and sudden changes in ambient light can blind cameras.
- **Close-quarters dynamics**: Multiple cars in tight formation lead to frequent occlusions.
- **Data scarcity**: Unlike consumer driving, there's very limited real-world racing data to train large-scale models.

These factors strip perception down to its essentials: can we detect, locate, and track every opponent reliably enough that the downstream planner and controller can execute high-speed maneuvers without fail?

By tackling perception in this unforgiving arena, we develop algorithms and systems whose robustness, precision, and efficiency elevate performance and safety - not only on the race-track, but across any high-speed, high-reliability autonomy domain.

1.3 Problem Statement

Assumptions

We consider the task of real-time opponent detection and state estimation on a fixed, known race track, under the following environment and assumptions:

- Track and Ego-Localization
  - The circuit layout and drivable boundaries are predefined and available offline.
  - Ego-vehicle pose (position and heading) is provided by a high-accuracy localization system.
- Detected Objects
  - Only other racecars will occupy the drivable surface - no pedestrians, traffic cones, or unexpected obstacles.

Detection Objectives

For every opponent car within sensor range, estimate in real time:

- 3D Position (x, y, z)
- Orientation (heading)
- Velocity (speed and direction)

The goal is to maintain a stable estimate of the above information for multiple opponent cars simultaneously, accounting for the noisy sensor environment while ensuring low latency, a relatively high frequency (approximately 20 Hz), and minimal occurrences of false detections or missed targets.

Research Objectives

This thesis investigates the design decisions, fusion architectures, and engineering practices required to build a robust, multi-modal perception stack that synthesizes LiDAR, radar, and camera data (with possible sensor hardware variation) into a semantically simple yet reliable output - opponent car state estimates - that downstream planners and controllers can trust at racing speeds.

1.4 Contributions

The perception stack presented in this thesis reflects the collaborative efforts of the entire ART Perception Team. Over the past three years, I have driven the Perception development for the AV-24 (serving as Perception Team Lead during the final year) and personally led the design, implementation, and on-track deployment across every stage of the stack:

- **Shared Modules (chapter 4)**
  - Extended the EKF tracker with Frenet frame-based lane-keeping and dynamic confidence scoring to maintain stable object tracks through prolonged dropouts.
  - Conducted evaluation of YOLOv8, including model selection, determining training data distributions, and tuning hyperparameters, to maximize detection accuracy.
- **Early-Stage Fusion (chapter 5)**
  - Designed and implemented a target-based LiDAR–camera calibration pipeline achieving sub-pixel reprojection errors.
  - Developed a high-throughput LiDAR-camera projection node with hybrid frustum prefiltering and mask-based refinement.
- **Late-Stage Fusion (chapter 6)**
  - Architected and trained a radar-only detection stack, including Doppler-to-parallel-velocity conversion, decision-tree classification, and specular-reflection suppression.
  - Redesigned and optimized the LiDAR-only pipeline, integrating Autoware modules with custom filtering nodes.
- **On-Track Deployment & Data Collection**
  - Led on-track integration, real-time debugging, and performance tuning of the perception stack under race conditions.

– Configured sensor hardware (eg: LiDAR scan patterns, camera PTP synchronization, network and buffer architectures) to ensure reliable data throughput.
– Engineered the electronics for the "Perception Jeep" testbed (Figure 1.3), a human-driven platform with identical sensors for data acquisition.

Figure 1.3: "Perception Jeep," with front sensor mounting visible.

Chapter 2

System Overview

2.1 Software

The software stack operates on Ubuntu 22.04, utilizing ROS2 Iron and Cyclone DDS middleware. The architecture is divided into the following main components:

- **Simulation:** Development of digital twins, real-time vehicle dynamics simulation, race line optimization, and satellite and sensor imagery analysis.
- **Perception:** Sensor ingestion, fusion, and filtering for world-state estimation, including object detection, classification, pose estimation, and predictive modeling.
- **Controls:** Low-level hardware management (steering, braking, gear shifting, accelerator) implemented using model predictive control algorithms.
- **Planning:** High-level strategic navigation, path planning, and obstacle avoidance.
- **Localization:** Integration of GPS and IMU data through Kalman filtering, pose estimation, and handling of faults and failures.

The real-time software used during races is primarily developed in C++, chosen for its compiled efficiency, type safety, manual memory management, and low-latency performance. Offline software, such as sensor calibration and data analysis scripts, is developed in Python due to its ease of prototyping, extensive libraries, and scripting capabilities. [31, 17]

Figure 2.1 illustrates the interactions between these components. This architecture follows a classical autonomous vehicle stack approach [27], with motivations discussed in the related section.

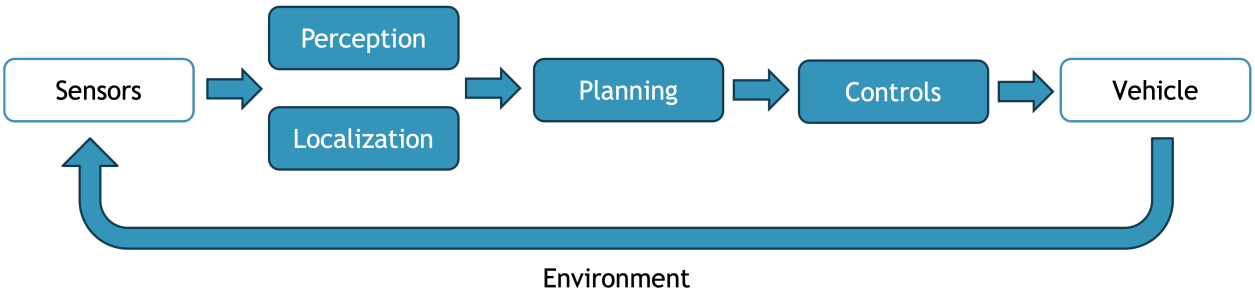


Figure 2.1: Autonomous Vehicle Stack

2.2 Hardware

The competition utilizes an IL-15 chassis manufactured by Dallara, modified by removing the driver's seat and replacing it with onboard computing hardware, sensors, and electronics required for autonomous operation (seen in Figure 2.2). The term AV-21 and AV-24 throughout this paper refer to specific configurations of sensors and hardware integrated into the IL-15 chassis. The current implementation used in competition is the AV-24 configuration.

Figure 2.2: Autonomous electronics where the driver would usually sit.

Chassis:

- Dallara IL-15
- Drive-by-wire, steer-by-wire, brake-by-wire
- Honda K20C engine
- 6-speed transmission
- Bridgestone tires

Onboard Computing:

- dSPACE Autera Autobox (14TB)
- Nvidia A5000 GPU
- ROS2 Iron on Ubuntu 22.04

Sensor Suite:

- **LiDAR:** Luminar Iris (x3) - high-resolution 3D spatial data
- **Radar:** Continental ARS548 (x2) - coarse 3D spatial data with direct Doppler velocity measurements
- **Cameras:** AlliedVision Mako G-319C (x6) - 2D semantic visual information
- **GNSS Antenna:** VectorNav VN-310 (x2) and PointOne RTK (x2) - 3D localization

While additional sensors such as tire temperature sensors, slip angle sensors, and wheel encoders are available, the above 4 sensor modalities are directly relevant to the functioning of the perception stack performance. Onboard computing capabilities emerged as a significant constraint, impacting overall system performance due to limited computational resources.

Sensor Variation

The sensors used by the perception stack have undergone significant hardware evolution over the past three years:

- **LiDAR:** Transitioned from Luminar Hydra units to Iris units across successive racing seasons.
- **Radar:** Transitioned from Adeptif ESR 2.5 (operating in object mode) to Continental ARS-548 and subsequently ZF ProWave radars (both operating in pointcloud mode).
- **Cameras:** Allied Vision MAKO G319 with different f-stops, exposure settings (fixed v/s dynamic) and mounting configurations.

Additionally, in the shorter term, sensors were frequently removed for specific races due to mechanical constraints. For instance, rear-mounted LiDAR and radar sensors were occasionally removed to mitigate issues related to exhaust pipe vibration and thermal damage (as they were mounted just above the diffuser). These hardware variations introduced additional complexities for the perception stack. Balancing the development of a software stack that remained largely hardware-agnostic while still extracting maximum performance from the available sensors proved challenging.

Sensor Field-of-View Coverage

Figure 2.3 shows the combined fields of view (FOV) of our AV-24 sensor suite (excluding two front stereo cameras, which are not used). All modalities - LiDAR, radar, and mono cameras - provide full frontal coverage, with partial overlap on the flanks. Notably, the rear of the vehicle experiences extended gaps in coverage when rear-mounted sensors are removed or fail.

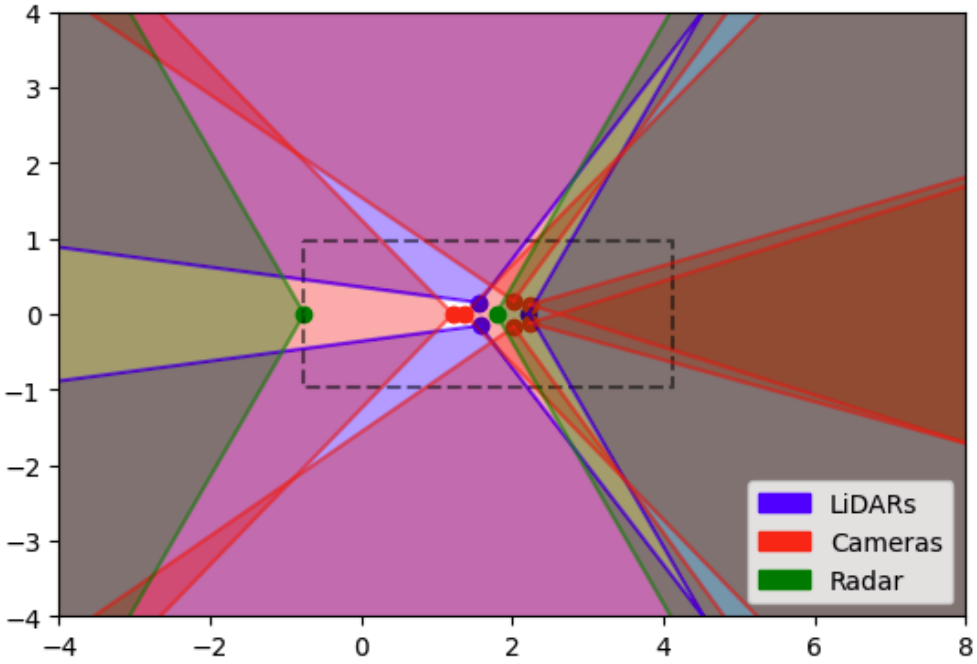
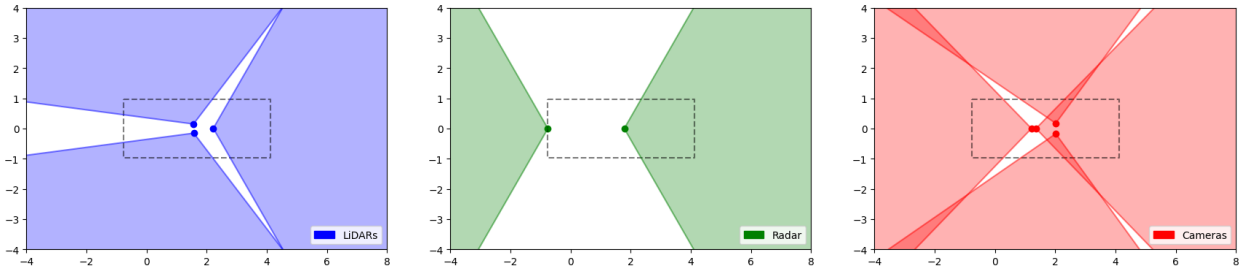


Figure 2.3: Combined sensor FOV for the AV-24 (car pointing to the right).



(a) LiDAR FOV (b) Radar FOV (c) Camera FOV

Figure 2.4: Individual sensor fields of view.

Chapter 3

Related Work

3.1 Introduction

Autonomous driving perception has evolved from classical modular pipelines - where distinct detection, tracking, and mapping modules process sensor data - to end-to-end deep learning approaches that learn mappings from raw inputs to driving outputs. Early successes like Stanford's "Stanley" in the 2005 DARPA Grand Challenge championed the modular paradigm [37], demonstrating reliable obstacle detection and path planning through separate LiDAR, radar, and vision modules fused via an Extended Kalman Filter. In contrast, recent advances in neural networks have made end-to-end systems increasingly viable: systems such as NVIDIA's PilotNet [5] and Tesla's FSD Beta directly map camera streams to driving decisions, bypassing handcrafted intermediate representations. This section frames that evolution, setting the stage for our investigation into perception for high-speed autonomous racing.

3.2 Sensor Modalities and Setups

Autonomous vehicles rely on complementary sensor modalities [44]:

- **Cameras (RGB/monocular/stereo):** provide rich color and texture information, necessary for semantic tasks [13] like traffic-light recognition and lane marking detection; sensitive to lighting and glare.
- **LiDAR (spinning/solid-state):** generates precise 3D point clouds with accurate range measurements in all lighting conditions [25]; expensive and produces high data rates; performance can degrade in heavy precipitation.
- **Radar (Doppler/imaging):** measures relative velocity directly and operates robustly in adverse weather [35]; lower spatial resolution makes fine classification and localization challenging.

- **Infrared/Thermal Cameras:** capture heat signatures, useful for night and poor-visibility scenarios; limited by lower resolution and higher noise [18].
- **Ultrasonic Sensors:** short-range proximity detection, used primarily for parking and low-speed maneuvers [45].

The IAC standardized the sensor suite for our racing platform to include solid-state LiDARs, Doppler radars, and monocular cameras. Therefore, the design challenge was not selecting sensor hardware but rather developing software capable of effectively fusing these predetermined modalities.

3.3 Case Studies in Industry

Tesla

Tesla pursues a *vision-only* strategy: eight surround cameras feed a shared convolutional backbone (e.g. RegNet) whose multi-scale features are transformed via learned BEV (bird's-eye-view) modules into a unified "vector space." [29] Multiple network heads then output lanes, drivable space, object trajectories, and traffic signals. Radar was phased out in 2021, shifting the burden of depth and velocity estimation entirely onto neural depth inference and temporal cues. This end-to-end-oriented design scales via Dojo-trained datasets but sacrifices hardware redundancy.

Waymo

Waymo employs a *multi-modal modular* stack: high-resolution LiDARs (long- and mid-range), a 360° camera array, and imaging radars [36]. Dedicated deep networks perform 3D detection on LiDAR and semantic labeling on images; an HD map provides context. Outputs feed into a probabilistic fusion and Extended Kalman Filter-based tracker, yielding robust object state estimates. This architecture prioritizes redundancy and interpretability, achieving industry-leading performance in diverse urban conditions.

Aurora

Aurora's Driver stacks Frequency-Modulated Continuous Wave (FMCW) LiDAR, directly measuring per-point velocity, with cameras and boresight radars. Their perception pipeline uses learned detectors on each modality, continuous self-calibration to counter vibration, and a Kalman filter–style fusion that integrates velocity and positional cues in real time. Aurora emphasizes long-range detection (over 400 m) for highway safety while maintaining a modular structure for validation and safety assurance.

3.4 Classical Modular Perception Pipeline

In the classical paradigm, perception is decomposed into sequential modules:

1. **Sensor-specific feature extraction** (e.g. color thresholding, HOG+SVM, point-cloud clustering).
2. **Object detection** via deep CNNs (R-CNN [13], SSD [26], YOLO [34]) on images and voxel- or point-based networks on LiDAR.
3. **Tracking** by Kalman Filter [21] variants (EKF, UKF, IMM) with data association (Hungarian algorithm, gating).
4. **Mapping and lane detection** against HD maps or occupancy grids.
5. **Sensor fusion** at the decision level (late fusion) or state level (filter updates) [14].

This modular approach offers interpretability and per-module validation but requires careful hand-tuning for fusion rules and interfaces. Modern systems augment it with deep learning in each component, preserving the pipeline structure for safety and clarity.

3.5 End-to-End Deep Learning Pipelines

End-to-end frameworks collapse perception, prediction, and planning into unified neural models. Early work (ALVINN [30], PilotNet [5]) demonstrated direct camera-to-steering mapping. Contemporary systems, such as Tesla's multi-camera BEV transformer and Wayve's map-free urban driver, ingest raw video (and sometimes radar) and output trajectories or controls. These models benefit from holistic optimization and can implicitly learn intermediate tasks (e.g. lane detection), but demand massive training data, introduce opaque failure modes, and complicate safety validation. Hybrid variants extract mid-level representations (BEV grids [29], object heatmaps [47]) within an end-to-end framework to balance interpretability with learning capacity.

3.6 Summary

The related work reveals a clear convergence toward *hybrid architectures*: modular pipelines embedding deep learning components, and end-to-end models augmented with interpretable intermediate outputs [39, 43]. Sensor fusion has progressed from hand-engineered early/late fusion toward learned attention-based methods, yet classical techniques persist where safety, predictability, and incremental validation are paramount.

Consequently, in the constrained context of autonomous racing, where training data is scarce and any perception error can have catastrophic consequences, hybrid architectures offer the

best of both worlds. The fixed track layout and known number of vehicles let us bake in strong geometric and kinematic priors, maximizing predictability and safety. At the same time, we need to adapt quickly to new circuit configurations or slight sensor variations without hand-crafting every parameter; embedding learning-based modules lets the system generalize beyond its original conditions. By combining deterministic classical filters with data-driven neural components, we demonstrate both the reliability required at high speeds and the flexibility to handle novel scenarios with minimal manual tuning.

Chapter 4

Perception System Design

4.1 Overview

In tackling robust opponent detection at racing speeds, we explored two distinct fusion paradigms: early-stage and late-stage fusion. The next two chapters delve into the details of each. This chapter introduces the *common building blocks* shared by both approaches - namely, the YOLOv8 segmentation model and the multi-modal tracker.

4.2 Methodology

Method #1: Early-Stage Fusion

Our initial implementation adopted an early-stage fusion paradigm. In general, early-stage fusion transforms each sensor's raw measurements into a common coordinate frame (the camera image plane or a shared 3D map) before any semantic processing, allowing downstream algorithms to work on fused, denser data [8]. Our implementation of this is outlined by Figure 4.1, and is covered in depth in chapter 5.

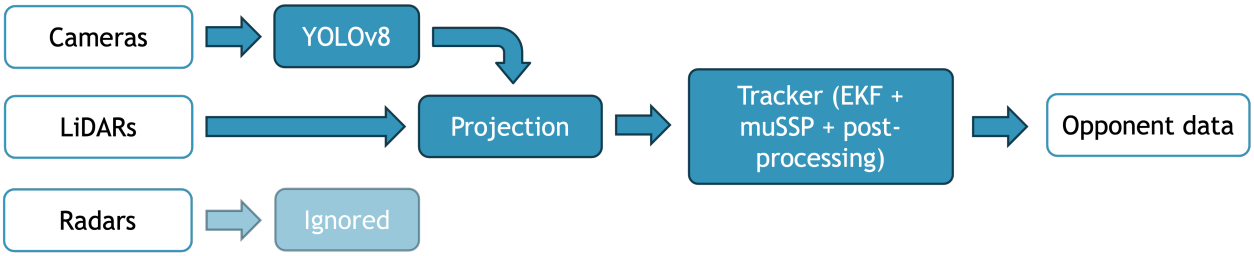


Figure 4.1: Early-stage sensor fusion pipeline

Method #2: Late-Stage Fusion

Faced with inconsistent detection quality and frequent "hallucinated" objects in the early-stage system, we transitioned to late-stage fusion. In this paradigm, each sensor modality maintains its own 3D detection pipeline, and the tracker fuses these independent hypotheses at a higher semantic level [2]. Figure 4.2 outlines our design, which is covered in depth in chapter 6.

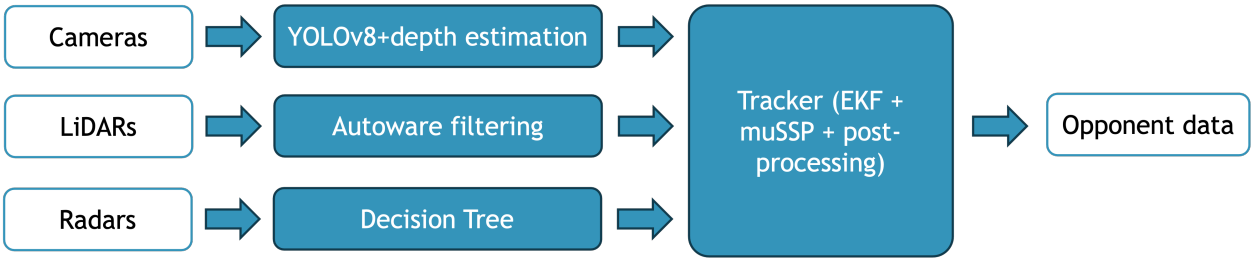


Figure 4.2: Late-stage sensor fusion pipeline

4.3 YOLOv8

Introduction

We selected YOLOv8 [38] as our primary vision detector because it delivers real-time instance segmentation and classification with state-of-the-art speed-accuracy trade-offs. Both our early-stage and late-stage fusion pipelines depend on YOLOv8 to localize opponents in the 2D image plane: early fusion projects LiDAR returns into the image and filters points by the YOLO mask, while late fusion uses YOLO's bounding boxes and masks alongside depth estimation to lift detections into 3D.

YOLOv8 is pretrained on the COCO dataset (80 common classes such as `bicycle`, `dog`, `car`, `person`, etc.), and we specifically leverage its `car` class. Architecturally, YOLOv8 is built on a convolutional neural network backbone optimized for high throughput on embedded GPUs.

Figure 4.3: Sample YOLO segmentation in the pitlane

Finetuning

Out-of-the-box, YOLOv8's `car` class is trained on street vehicles under everyday conditions. To adapt it to our racing domain, where cars feature unique liveries, extreme motion blur, and variable lighting across different tracks, we finetune on tens of thousands of racecar images. Key drivers for finetuning include:

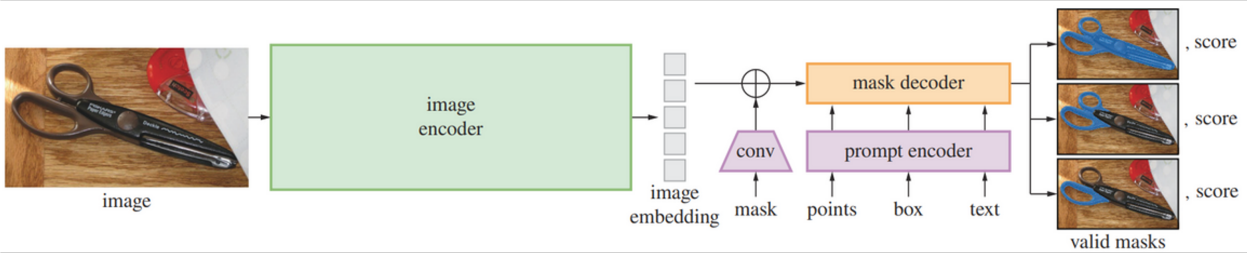
- Diverse track environments (day/night, sun glare, shadows)
- Multiple vehicle liveries and shapes
- Motion artifacts at high speeds (blur, distortion)
- Limited availability of real racing footage—necessitating a large, tailored dataset

By retraining the final layers and adjusting anchor priors on our domain-specific data, we significantly improve detection confidence and mask accuracy for racing cars.

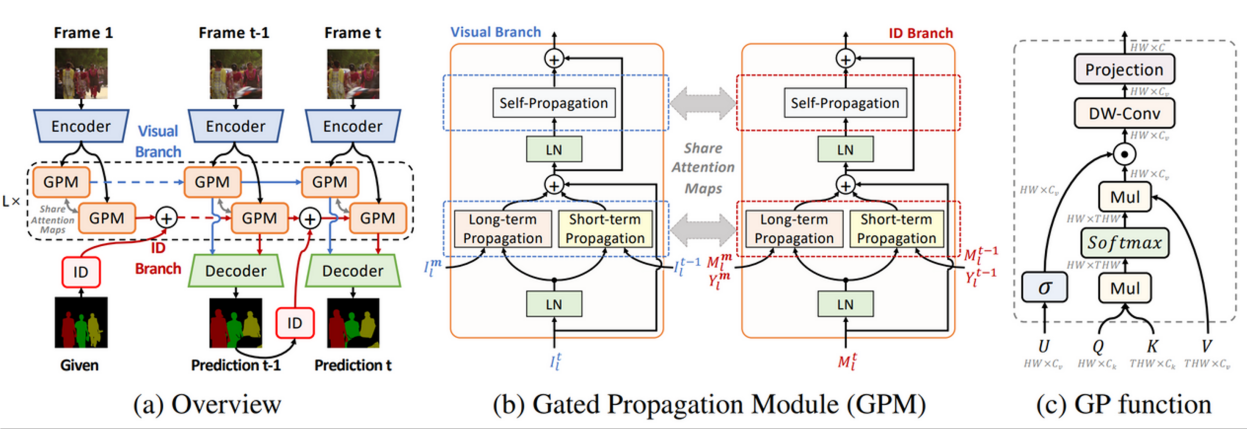
Data Labeling

To generate the required labeled images efficiently, we employ a semi-supervised pipeline built around Segment Anything Model (SAM) tools. Initially, we used SAM1 within CVAT to annotate keyframes (Figure 4.4a), then propagated masks over adjacent frames via a separate tracking algorithm (Figure 4.4b). After SAM2's release in late 2024, we switched to its built-in video segmentation capability, which provides temporal consistency and obviates the need for an external tracker. This workflow lets us produce on the order of 300 auto-labeled

frames per human-annotated keyframe, focusing human effort on high-quality annotations while automating the bulk of the dataset generation.



(a) SAM1 segmentation pipeline



(a) Overview (b) Gated Propagation Module (GPM) (c) GP function

(b) SAM1 segment propagation pipeline

Figure 4.4: Data labeling pipeline

Training with Simulated Data

Given the scarcity and expense of real multi-car racing data, we explored alternative sources, including public online datasets such as KITTI Vision Benchmark Suite [12] and the nuScenes dataset [7], but found none that matched our domain. Instead, we turned to simulator-generated images, which offer virtually unlimited, perfectly labeled samples (with exact ground-truth vehicle positions). However, simulated sensor data suffer from:

- Unrealistic lighting, textures, and reflections
- Absence of camera noise, vibration blur, and lens distortions
- A feature distribution shift relative to real footage

To mitigate this reality gap, we finetune YOLOv8 on stylized sim data - applying a photorealistic style transfer (Deep Translation Prior) to better match real-world textures and noise

patterns [9, 22]. This approach improves robustness to unseen track conditions, though care must be taken to avoid overfitting to synthetic artifacts.

Deployment

For deployment on our on-board NVIDIA A5000, we convert the PyTorch checkpoint (`.pt`) - ideal for research but too slow for real time - into an intermediate ONNX (`.onnx`) representation, and then compile it to a TensorRT engine (`.engine`). The ONNX format provides framework-agnostic portability, while the TensorRT engine is quantized (FP16), hardware-accelerated, and optimized for low-latency inference on our GPU, meeting our sub-30 ms perception budget per frame.

4.4 Tracker

The tracker ingests `DetectedObject` ROS2 messages from individual perception modalities (camera, LiDAR, radar) and outputs fused, temporally consistent `TrackedObject` messages. A `DetectedObject` consists of a bounding volume and pose (position and orientation). In contrast, `TrackedObject` messages include velocity (twist) and a unique, temporally consistent identifier.

While the preceding perception modules operate on individual frames independently, the tracker introduces *temporal* consistency by associating detections across successive frames into persistent object tracks. Our approach uses Autoware's `autoware_multi_object_tracker` as a baseline, which we further enhance to meet our performance and accuracy requirements.

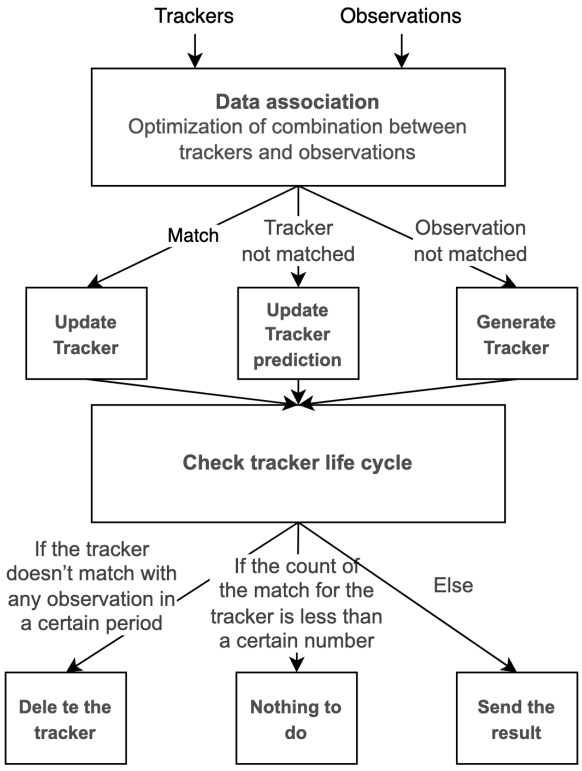


Figure 4.5: Autoware multi object tracker, consisting of EKF and data association.

Extended Kalman Filter (EKF)

We adopt an Extended Kalman Filter (EKF) to propagate each tracked object's state over time and fuse asynchronous detections:

- **Nonlinear dynamics:** we model each car with a bicycle-model motion (nonlinear kinematics), linearized at each step.
- **Multi-modal fusion:** the EKF update step ingests observations from camera (2D reprojections), radar (range + Doppler velocity) and LiDAR (3D position), weighting each by its measurement covariance.
- **Noise smoothing and dropout handling:** the predict step fills in short detection dropouts (up to ∼1 s) by propagating the state; the filter naturally smooths sensor noise.
- **Data association:** we use the Minimum-update Successive Shortest Path (muSSP) [1] algorithm to associate new detections with existing tracks by solving a global assignment problem at each timestep.

This EKF + muSSP core is very efficient, suitable for our 20 Hz perception loop, but in practice we observed two shortcomings:

1. **Track dropout:** during extended occlusions, tracks faded out or became highly uncertain after ∼1 s.
2. **Track hallucination/duplication:** noisy detections resulted in momentary 'blips' of a track. Additionally, muSSP occasionally split a single car into multiple `TrackedObjects` that could diverge and cause further hallucination.

EKF Post-Processing

To address these limitations, we exploit racing-track constraints, such as fixed track layout and a known number of opponents, through two post-processing strategies applied to the raw EKF outputs: lane-keeping hallucination and customized confidence thresholding.

Lane-Keeping Hallucination

Dynamics-based trackers alone struggle to reliably "hallucinate" an opponent's position during sensor dropouts, especially at racing speeds of up to 160 mph, where even brief interruptions are critical. Observing that racecars closely follow established racing lines provides a valuable higher-level cue.

Leveraging this observation, we implemented a lane-keeping hallucination method. When EKF tracking momentarily fails, we estimate the vehicle's position based on its adherence to the racing line rather than solely relying on a dynamics-based model. We assume the opponent vehicle maintains its lane during these dropouts. While this assumption is strong and occasionally incorrect, it proved sufficiently effective in practice.

To facilitate this lane-keeping approach, we introduce a *Frenet frame* [40], which simplifies motion planning by expressive the car's position relative to the road. We convert each EKF state (x, y) in the global frame into Frenet coordinates (s, d):

- $s \in [0, 1]$: longitudinal progress along the centerline of the road (laps wrap at $s = 1 \to 0$, the start/finish line).
- $d \in [-1, 1]$: lateral offset from centerline, where $d = 0$ is the center and $|d| = 1$ hits the track boundaries.

In order to implement lane keeping in this Frenet frame, we freeze d (no lateral drift) and propagate s with a constant velocity or acceleration model until the next observation. See Algorithm 1 for a high-level overview of how tracks from the EKF are 'lane-kept' in the case of dropout.

Algorithm 1 Lane Keeping Hallucination

```
procedure onTrackedObjects(msg)
    for detectedObject in msg.objects do                          ▷ EKF update
        track ← find_existing_track(detectedObject)      ▷ by ID or spatial proximity
        if track is None then
            track ← init_new_track()
        end if
        track.EKF_pos ← detectedObject.pos
        track.EKF_vel ← detectedObject.vel
        track.updated_by_EKF ← true
    end for
    for track in tracked_tracks do
        if track.updated_by_EKF then                              ▷ Store frenet coords
            track.frenet_pos ← to_frenet(track.EKF_pos)
            track.frenet_vel ← to_frenet(track.EKF_vel)
        else                                                  ▷ Lane-keeping hallucination
            track.frenet_pos.s ← track.frenet_pos.s + track.frenet_vel.s ×Δt
            track.EKF_pos ← to_euclidean(track.frenet_pos)
        end if
    end for
    publish(tracked_tracks)
end procedure
```

Confidence Tracking

By default, Autoware's multi-object tracker hallucinates EKF tracks for a fixed timeout (1 s), after which they can diverge. With our lane-keeping hallucination, we can safely extend this period. However, rather than using a larger constant timeout, we compute a *dynamic* confidence C for each track. Tracks whose C falls below a threshold are removed; others persist through moderate sensor outages.

We define

$$C = \sqrt[3]{C_t^{2.2} \times C_r^{0.3} \times C_\Sigma^{0.5}},$$

where each sub-confidence ranges from 0 to 1:

- C_t ("time"): decays sigmoidally with time since last EKF update.
- C_r ("range"): decreases as the track's distance from the ego vehicle grows.
- C_Σ ("covariance"): decreases with increasing EKF state covariance (position, velocity, and yaw uncertainty).

Each sub-confidence is computed via a saturating sigmoid using the following function.

```
double applySigmoid(double x, double x_sat, bool mirrored=false) {
  if (mirrored)
    return 1.0 / (1.0 + exp(6*x/x_sat - 3.0));
  else
    return 1.0 / (1.0 + exp(3.0 - 6*x/x_sat));
}
```

This function approximately returns 1 at $x = 0$ and approaches 0 as $x \to x_{\text{sat}}$ when `mirrored` is `true`.

Example. For time confidence C_t, with $x_{\text{sat}} = 2.3$ s (so 2.3 seconds since the track was last updated by the EKF), the curve in Figure 4.6 shows $C_t(2.3 \text{ s}) \approx 0.56$.

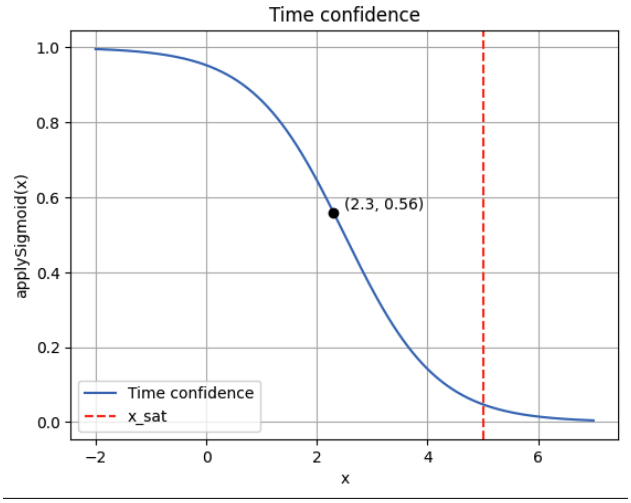


Figure 4.6: Sigmoidal decay of time-since-update confidence C_t.

The range confidence C_r uses the same logic with distance as x, and C_Σ combines three mirrored Sigmoids on the EKF's position, velocity, and yaw covariances.

Together, these EKF, lane-keeping, and confidence mechanisms yield a robust multi-modal tracker that is cognizant of both vehicle dynamics and track constraints.

Chapter 5

Method #1: Early-Stage Fusion

5.1 Introduction

Early-stage fusion tightly couples raw sensor data at the measurement level, projecting all modalities into a common reference frame before any higher-level processing [8]. In our implementation, we fuse YOLOv8's 2D instance-segmentation output with LiDAR point clouds to obtain precise 3D opponent locations:

YOLOv8 instance segmentation Each monocular camera image is processed by YOLOv8 to generate pixel-accurate masks and class labels for every detected car.

Perspective projection of LiDAR points Every LiDAR return $\mathbf{X}_{LiDAR} = [X, Y, Z]^\top$ is first transformed into the camera frame via the extrinsic rotation $\mathbf{R}$ and translation $\mathbf{t}$:

$$\mathbf{X}_{cam} \;=\; \mathbf{R}\,\mathbf{X}_{LiDAR} + \mathbf{t} \quad \text{(all in ENU coordinates)}.$$

It is then projected into image coordinates (u, v) using the calibrated intrinsic matrix $\mathbf{K}$ and distortion $\mathbf{D}$.

Mask-based filtering A LiDAR point is retained if and only if its projected (u, v) lies inside one of the YOLOv8 car masks. This produces a 3D point-cloud slice corresponding exactly to each opponent's surface.

1. *YOLOv8 instance segmentation:* Each monocular camera image is processed by YOLOv8 to generate pixel-accurate masks and class labels for every detected car.
2. *Perspective projection of LiDAR points:* Every LiDAR return $\mathbf{X}_{LiDAR} = [X, Y, Z]^\top$ is first transformed into the camera frame via the extrinsic rotation $\mathbf{R}$ and translation $\mathbf{t}$:

$$\mathbf{X}_{cam} \;=\; \mathbf{R}\,\mathbf{X}_{LiDAR} + \mathbf{t} \quad \text{(all in ENU coordinates)}.$$

It is then projected into image coordinates (u, v) using the calibrated intrinsic matrix $\mathbf{K}$ and distortion $\mathbf{D}$.

3. *Mask-based filtering:* A LiDAR point is retained if and only if its projected (u, v) lies inside one of the YOLOv8 car masks. This produces a 3D point-cloud slice corresponding exactly to each opponent's surface.

Because we perform per-point matrix multiplications at up to 1.2 million points per second, the efficiency and accuracy of our projection routine are paramount. Moreover, any misalignment in the extrinsic calibration, i.e. errors in $\mathbf{R}$ or $\mathbf{t}$, directly translate to mis-projected points and degraded detection performance. Thus, precise LiDAR–camera calibration is a prerequisite for reliable early-stage fusion.

5.2 LiDAR-Camera Calibration

Accurate extrinsic calibration between LiDAR and camera sensors is essential for robust sensor fusion in autonomous racing. Misaligned calibrations can lead to significant projection errors, resulting in incorrect object detections and ultimately tracking failures.

Existing Methods

Existing approaches to LiDAR-camera calibration generally fall into two categories:

- **Target-based methods:** Use known calibration targets (e.g., checkerboards or ArUco boards) to establish correspondences between 3D points and 2D image features [46, 28].
- **Feature-based (automatic) methods:** Detect natural features (edges, corners) in overlapping fields of view, then optimize alignment [24].

While target-based techniques offer high accuracy under controlled conditions, they require manual setup and may not adapt to shifts during racing. Feature-based methods remove the need for explicit targets but often struggle in low-texture or dynamic environments.

Custom Calibration Pipeline

To address the specific demands of autonomous racing, we implemented a custom target-based calibration pipeline in Python using OpenCV and Open3D. We use an ArUco board (a rigid planar target with a central ArUco marker) as our calibration object which could be independently detected by the camera and LiDAR (see Figure 5.1).

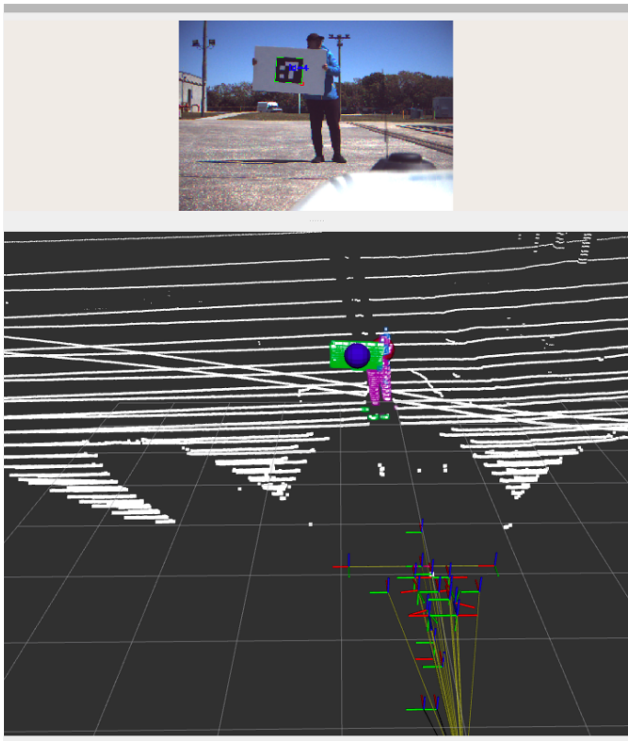


Figure 5.1: LiDAR–camera calibration setup. The blue dot represents the board detected in the image frame; green points correspond to the board plane in the LiDAR frame.

Board Detection in the Camera Frame

We detect the ArUco marker in each image using OpenCV's `cv2.aruco.detectMarkers()` and `cv2.aruco.estimatePoseSingleMarkers()`, which returns the 3D corner coordinates relative to the camera. By averaging these four corners, we obtain the board's center in the camera frame, assuming known intrinsics (calibration matrix $\mathbf{K}$ and distortion coefficients $\mathbf{D}$).

Board Detection in the LiDAR Frame

To localize the board in the point cloud, we apply:

1. **Radius-based prefiltering.** Let $\mathbf{p}_{\text{cam}}$ be the board center in coordinates (projected from the camera pose). We retain all points $\mathbf{p}$ satisfying

$$\|\mathbf{p} - \mathbf{p}_{\text{cam}}\| \leq r\,.$$

   where r is a hyperparameter set to a value that exceeds the physical distance between the LiDAR and camera plus the board's diagonal.

2. **Ground-plane removal.** We estimate normals via `o3d.estimate_normals()` and discard points whose normals are nearly co-planar with the ground.

3. **Plane segmentation.** Using `o3d.segment_plane()`, we extract planar regions. The largest plane is assumed to be the board (other planes, like of the person holding the board, are assumed to be smaller). In Figure 5.1, the board plane appears as the green pointcloud while the person's plane is shown in pink.

4. **ICP alignment.** We generate a synthetic point cloud of the board at the origin (grid of the board's real dimensions) and apply Open3D's ICP (`registration_icp()`) to align it to the segmented plane. The resulting rotation $\mathbf{R}$ and translation $\mathbf{t}$ locate the board center in the LiDAR frame.

Estimating the Extrinsics

With N correspondences $\{(\mathbf{X}_i, \mathbf{X}'_i)\}$, where $\mathbf{X}_i$ is the board center in LiDAR coordinates and $\mathbf{X}'_i$ in camera coordinates, we compute the affine transform $\mathbf{M} \in \mathbb{R}^{3\times4}$ by calling

$$\texttt{cv2.estimateAffine3D}(\{\mathbf{X}_i\}, \{\mathbf{X}'_i\})\,.$$

Under the hood, this proceeds as follows:

1. *RANSAC sampling:* Repeatedly select 4 non-coplanar correspondences $(\mathbf{X}_i, \mathbf{X}'_i)$, $i = 1, \ldots, 4$.

2. *Initial linear solve:* Solve for the 3×4 matrix M in

$$M \begin{bmatrix} \mathbf{X}_i \\ 1 \end{bmatrix} = \mathbf{X}'_i, \quad i = 1, \ldots, 4$$

   by direct least-squares on this small system.

3. *Inlier counting:* For each candidate M, compute the reprojection error

$$\left\| M \begin{bmatrix} \mathbf{X}_j \\ 1 \end{bmatrix} - \mathbf{X}'_j \right\| \; < \; \text{ransacThreshold}$$

   and count inliers; retain the model with the maximum inlier set.

4. *Refinement:* Given the final inlier index set I, refine M by

$$\min_M \sum_{i \in I} \left\| M \begin{bmatrix} \mathbf{X}_i \\ 1 \end{bmatrix} - \mathbf{X}'_i \right\|^2,$$

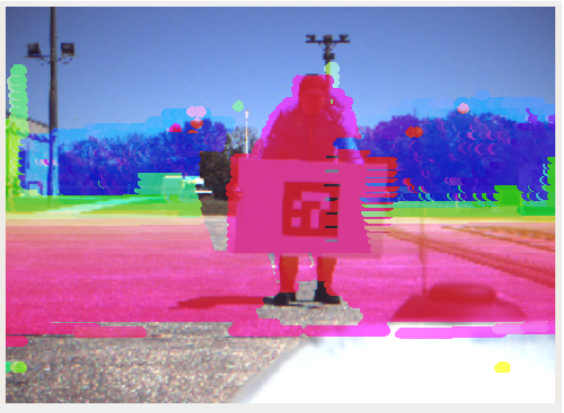
   which is solved via SVD/QR-based least squares to produce the final affine matrix M.

Candidate-Pair Validation

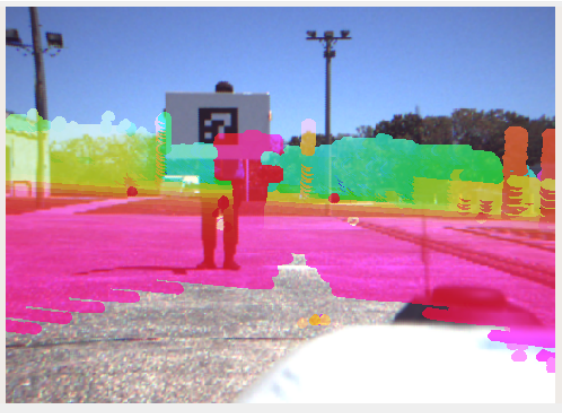
In practice, we found that simply running the steps above was extremely slow on the DSpace computer and provided insufficient results (as seen in Figure 5.2b. To ensure stable, well-conditioned calibration, we accept a board detection pair only if it satisfies some additional conditions:

1. **Aruco stability:** low variance in the marker's location over a 3 s window.
2. **Timestamp sync:** image and LiDAR stamps differ by less than a set tolerance.
3. **Spatial diversity:** new pairs must be at least a minimum 3D distance from existing ones to avoid rank deficiency.
4. **ICP quality:** the ICP RMSE must be below a threshold to avoid false segment matches.
5. **URDF consistency:** the measured offset between frames must approximately match the sensor offsets recorded in the vehicle's URDF.

Enforcing these criteria greatly reduced reprojection error and improved the solve time (improved reprojection results seen in Figure 5.2a). Achieving stable performance of this pipeline required tuning over 24 parameters (such as the r in radius-based filtering or the RANSAC threshold), but once determined this single parameter set generalized across different LiDAR-camera pairs and maintained accuracy even under small sensor misalignments.



(a) Example of good calibration (depth aligned with objects)



(b) Example of bad calibration (depth misaligned with objects)

Figure 5.2: Calibration quality comparison. The colored overlay represents LiDAR points projected into the 2D image plane using the estimated extrinsics.

Limitations

Despite its accuracy, our pipeline has drawbacks:

- **Re-calibration required** whenever sensors shift (e.g. after installing new sensor shocks).
- **Pairwise calibration** needed for each LiDAR-camera pair with overlapping fields of view.
- **Time-consuming:** each valid pair takes 15–20 s to collect, requiring ∼10 min for 30 pairs. This has to be done for each LiDAR-camera pair.
- **Physical space requirements:** capturing well-spaced pairs require up to 20 m of clearance from the car, which is often unavailable in the garages.

5.3 LiDAR-Camera Projection

Objective

The goal of LiDAR–camera projection [8] is to extract, for each detected opponent car, the subset of LiDAR points that lie on that car's surface. Given the YOLOv8 instance-segmentation masks in the image plane, we project each LiDAR point into the camera frame and test whether it falls inside the corresponding mask. This requires accurate camera intrinsics $\mathbf{K}$ and distortion $\mathbf{D}$, plus the extrinsic transform $(\mathbf{R}, \mathbf{t})$ between LiDAR and camera. This will then be passed onto the tracker to introduce temporal consistency.

Perspective Projection Mathematics

Let $\mathbf{X} = [X, Y, Z, 1]^\top$ be a homogeneous LiDAR point in the LiDAR frame. We first transform into the camera frame:

$$\mathbf{X}_{cam} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{X} = \big[X_c, Y_c, Z_c, 1\big]^\top.$$

We then project onto the image plane via

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \big[X_c/Z_c,\ Y_c/Z_c,\ 1\big]^\top,$$

and apply distortion correction using the parameters in $\mathbf{D}$. A point (u, v) that lies within a YOLO mask is considered to belong to that detected object.

Baseline Implementation and Its Limitation

A naive approach uses OpenCV's `cv::projectPoints()` on all LiDAR points (∼20 000 points per sensor × 3 sensors × 20 Hz ≈ 1.2 M points/s). The required matrix multiplications make this infeasible for real-time deployment. Conversely, transforming the high-resolution segmentation mask into 3D for direct filtering in the LiDAR frame is computationally intractable. Hence, we introduce a hybrid approach that combines a bounding-box frustum filter with a segmentation mask refinement.

Coarse Bounding Box Frustum Filtering

To reduce the candidate set, we perform a coarse 3D frustum check with the following steps:

1. **Back-project image corners.** For each 2D bounding box $[u_{\min}, v_{\min}, u_{\max}, v_{\max}]$, form the homogeneous corner vectors

$$\mathbf{b}_i = \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}, \quad i = 1, \ldots, 4.$$

   Let $\mathbf{t}' = \mathbf{R}^{-1}\,\mathbf{t}$ where t'_x, t'_y, t'_z are its components. Compute the 3D rays in the LiDAR frame via

$$\mathbf{p}_{\ell,i} = \mathbf{R}^{-1}\,\mathbf{K}^{-1}\,\mathbf{b}_i \;-\; \mathbf{t}'.$$

   where $\mathbf{K}$ is the camera intrinsics matrix and $(\mathbf{R}, \mathbf{t})$ the LiDAR-camera extrinsic.

2. **Approximate planar edges.** Write each ray $\mathbf{p}_{\ell,i} = (x_i, y_i, z_i)^\top$. We then fit two line-planes in the y–x and z–x projections:

$$y = a_{y,i}\,x + b_{y,i}, \quad z = a_{z,i}\,x + b_{z,i},$$

   with coefficients

$$a_{y,i} = \frac{y_i + t'_y}{x_i + t'_x}, \quad b_{y,i} = a_{y,i}\,t'_x - t'_y,$$

   and analogously for $a_{z,i}, b_{z,i}$ using the z–component.

3. **3D half-space tests.** For each LiDAR point $\mathbf{p} = (x, y, z)$, enforce the four linear inequalities

$$y_{\min}(x) \le y \le y_{\max}(x), \quad z_{\min}(x) \le z \le z_{\max}(x),$$

   where $y_{\min}, y_{\max}$ (resp. $z_{\min}, z_{\max}$) come from the lines of the $u_{\max}, u_{\min}$ (resp. $v_{\max}, v_{\min}$) edges. Points satisfying all four constraints lie inside the projected 3D frustum.

This bounding box frustum filter reduces the candidate LiDAR points from $\mathcal{O}(10^4)$ to $\mathcal{O}(10^2)$ per detection.

Segmentation-Mask Refinement

On the reduced set of frustum points, we now perform the accurate mask check:

1. Use `cv::projectPoints()` to project each candidate $\mathbf{X}$ into (u, v).
2. Check whether (u, v) lies inside the YOLOv8 segmentation mask.

Since only $\sim$100 points are tested, this step is real-time feasible.

Further Refinements for Calibration Sensitivity

Because even small calibration errors in $\mathbf{K}, \mathbf{D}, \mathbf{R}, \mathbf{t}$ can mis-project points, we add the following robustness measures:

Range-based Clustering

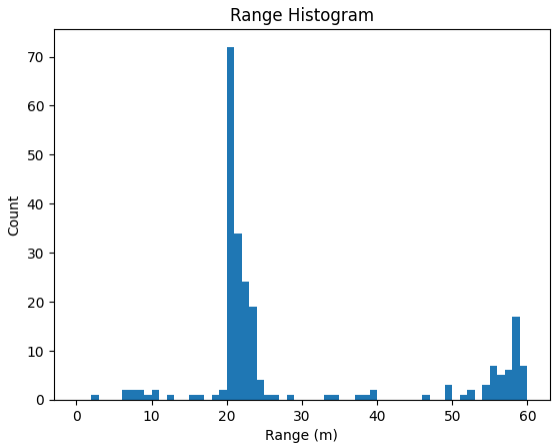
We first cluster candidate LiDAR points by their Euclidean range $\rho_i = \|\mathbf{X}_i\|$. We choose a bin width $\Delta\rho \approx 1$m and assign each point to two overlapping bins:

$$b_i = \left\lfloor \tfrac{2\,\rho_i}{\Delta\rho} \right\rfloor, \quad \text{assign to bins } b_i \text{ and } b_i - 1.$$

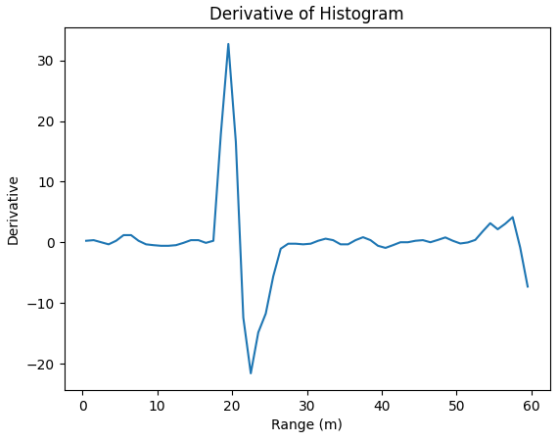
This overlap smooths transitions between adjacent bins. We tally counts $c[b]$ per bin, then convolve with the derivative kernel (sample plot in Figure 5.3)

$$k = [\,0.24197072,\ 0.34495131,\ 0.0,\ -0.34495131,\ -0.24197072\,].$$

Scanning the resulting derivative $d[b]$, we locate the first rising edge $b_{start} = \min\{b \mid d[b] > 0\}$, then continue until $d[b]$ falls below zero and subsequently rises again at b_{end}. The range interval $[\rho_{start}, \rho_{end})$ (where $\rho_{start} = b_{start}\,\Delta\rho/2$, etc.) brackets a dense cluster, typically the car surface, while discarding sparse ground or background points. The effect of this range-based clustering can be seen in Figure 5.4.

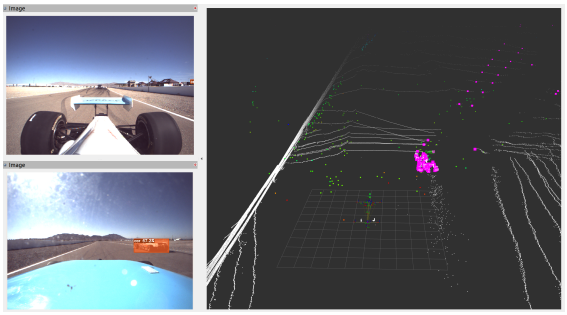


(a) Range histogram. The peak at 20m is a car, while the peak at 55m is the wall.

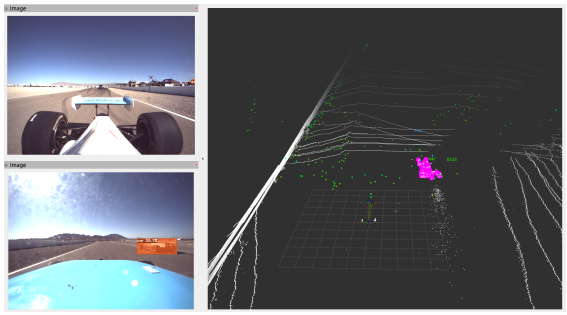


(b) Corresponding derivative histogram. The car can be easily extracted from this.

Figure 5.3: Binned histograms for cluster detection.



(a) Output of projection without binning. Note the presence of ground/wall points.



(b) Output of projection after binning. The ground/wall points have disappeared.

Figure 5.4: Output of LiDAR-camera projection. YOLOv8 mask (orange) used to select LiDAR points (pink).

Singular-Value Decomposition (SVD) Ground Rejection

Within each cluster of N points, we form the $3 \times N$ matrix $\mathbf{X} = [\, x_i, y_i, z_i \,]_{i=1}^{N}$ and compute its SVD:

$$\mathbf{X} = \mathbf{U}\,\Sigma\,\mathbf{V}^{\top}, \quad \Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \sigma_3),\ \sigma_1 \geq \sigma_2 \geq \sigma_3.$$

Planar surfaces (ground, walls) exhibit very small third singular value σ_3. We reject any cluster with $\sigma_3 < \tau_{ground}$, ensuring only volumetric (car) clusters remain.

Bounding-Box and Mask Dilation

To absorb minor reprojection drift, we dilate the 2D bounding box and segmentation mask by a small margin k pixels in all directions [10]. Although this admits more false-positive candidates, the subsequent binning and SVD filters remove spurious ground points while recovering all true car points. An example of this is in Figure 5.5.

(a) Default YOLOv8 segmentation mask.

(b) Dilated YOLOv8 segmentation mask.

Figure 5.5: Comparison of default v/s dilated YOLOv8 segmentation mask.

Optimizations

To maintain real-time throughput while preserving sufficient point density on targets, we apply two key optimizations:

Dynamic Downsampling

We adaptively downsample LiDAR points based on bounding-box area A_{bbox}, targeting approximately POINTS_IN_BBOX points per detection. The downsampling ratio is computed as

$$f_{down} = \min\Big(1,\ \max\big(\epsilon,\ \frac{\text{POINTS\_IN\_BBOX}}{N_{pts}} \times \frac{0.6}{A_{bbox}}\big)\Big), \quad f_{down} \in (0, 1],$$

where $\epsilon > 0$ is a small lower bound, and points are skipped at stride $\lceil 1/f_{down} \rceil$. The 0.6 constant approximates around 60% of LiDAR points fall in the camera's FOV. Closer cars (large A_{bbox}) thus use fewer points, while distant objects remain densely sampled.

CUDA Acceleration

We implemented custom CUDA kernels for the frustum-filter test (ray-frustum intersection) to meet our 20 Hz deadline on the NVIDIA A5000 GPU.

Limitations

- **Single-mode dependency:** YOLOv8 is a single point of failure. If YOLO fails to detect a car, the downstream stack is redundant.
- **Calibration drift:** Despite binning and expansion, severe mis-calibrations still degrade performance.
- **Time-sync issues:** Without hardware PTP across sensor modalities, we match the nearest timestamps which is another source of error.

These drawbacks motivate development of redundant, modality-agnostic pipelines that fuse information in 3D, reducing reliance on any single detection modality or precise extrinsics. This inspired the development of the second approach using late-stage fusion.

Chapter 6

Method #2: Late-Stage Fusion

6.1 Introduction

Late-stage fusion decouples each sensor modality into its own 3D detection pipeline, then merges their outputs in a common tracking framework [2]. This approach overcomes three key drawbacks of our early-stage fusion design:

- **Single-mode dependency:** Relying solely on YOLOv8 mask filtering creates a single point of failure.
- **Calibration drift:** Small errors in the LiDAR–camera extrinsics $(\mathbf{R}, \mathbf{t})$ lead to misprojected points.
- **Time-synchronization limitations:** Nearest-timestamp matching across modalities introduces temporal misalignment.

We therefore implement three independent stacks:

1. **LiDAR-only Stack** (Autoware ground segmentation and track boundary filtering)
2. **Radar-only Stack** (Decision-tree classifier on Doppler-converted parallel velocity and signal features)
3. **Camera-only Stack** (YOLOv8+monocular depth estimation to produce 3D car detections)

Each pipeline produces a set of 3D object hypotheses in the vehicle's frame, which are then fused by the Kalman filter-based tracker.

Table 6.1: How Late-Stage Fusion mitigates Early-Stage Fusion drawbacks

Early-Stage Drawback	**Consequence**	**Late-Stage Fusion Solution**
Single-mode dependency	YOLO mask failure means no 3D points	**Redundancy by modality**: LiDAR, radar, and camera each provide independent 3D tracks; loss of one modality does not halt detection.
Calibration drift	Mis-projected LiDAR points degrade detection accuracy	**Per-modality 3D estimates**: Each stack operates in its own sensor frame; extrinsic errors only affect late fusion weighting, not raw detection.
Time-sync limitations	Temporal jitter from nearest-timestamp matching	**Asynchronous track fusion**: The EKF tracker ingests time-stamped tracks from each stack and compensates for delays via prediction and interpolation, smoothing out timestamp mismatches.

6.2 LiDAR-only Stack

The LiDAR-only perception pipeline builds on Autoware's `point_cloud_preprocessor` (ground segmentation, crop-box filtering), augmented with custom modules for enhanced robustness. By segmenting the ground and cropping to the known track footprint, the remaining point clouds ideally contain only other racecars. This section details the primary stages of the LiDAR-only stack.

Ground Segmentation

We use Autoware's `ScanGroundFilter` to quickly remove the road surface. The filter operates in three steps [15]:

1. **Ray slicing:** Split the point cloud into narrow azimuthal rays.
2. **Range sorting:** Sort each ray's points by horizontal distance.

3. **Slope-based classification:** Sweep outward along each ray, comparing each point's height and local slope to the previous "ground" point; points exceeding configurable height or slope thresholds are labeled non-ground.

This lightweight, ray-wise approach reliably strips away flat surfaces in real time, yielding a point cloud of only obstacles and vehicles.

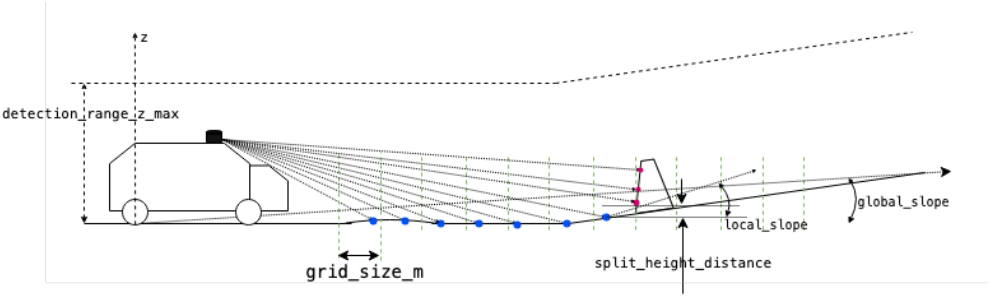


Figure 6.1: Autoware ground filtering

Addressing the Radial Blindspot

Because our LiDARs only begin returning ground echoes beyond ∼8m, the inner 8m radius forms a "ground blindspot" (Figure 6.2). Within this zone, the filter would misclassify car points as ground, splitting the ray incorrectly. This is because the algorithm is expecting `split_height_distance` z difference between the radially closer ground (blue) and further non-ground (red) points in Figure 6.1).

We therefore override the segmentation: *all* points within 8m are treated as non-ground, preserving close-range car detections essential for overtaking and tight maneuvers.

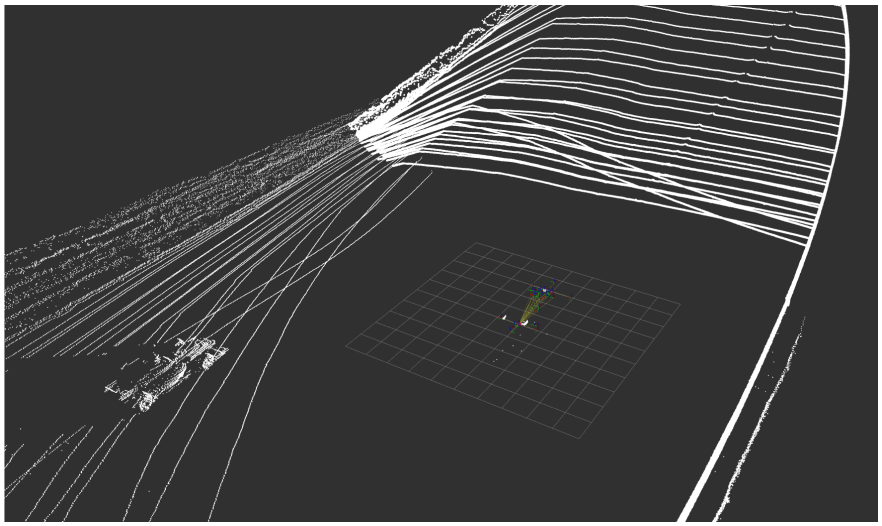


Figure 6.2: LiDAR ground blindspot ∼8m radially around the car. The ego car is at the center of the grid, and LiDAR points (white) only appear on the ground ∼8m away.

Parameter Tuning

Ground filtering parameters are highly track-dependent. For example, oval tracks like Kentucky Speedway has banking of up to 17°, while flat courses like Las Vegas Road Course remains near zero. This affects parameters such as `global_slope_max_angle_deg` and `non_ground_height_threshold`.

Additionally, because these racecars present very low upper bodywork slopes, the filter often confuses banked track sections with car surfaces, and means the algorithm is very sensitive to tuning. However, once tuned to a track, we have found it to be quite reliable.

Track Boundary Filtering

To enforce on-track detections, we polygon-clip the segmented point cloud against the known track edges:

1. Load GPS-defined track boundary vertices.
2. Transform boundary polygon from global (ENU) into the LiDAR frame via the ego-pose.
3. Discard any non-ground point outside the polygon using a point-in-polygon test.

Dependence on Global Localization

Accurate heading and position are critical: even a few degrees of heading error at low speeds can shift the polygon so that track walls (non-ground) are retained as in-track points, causing false positives (Figure 6.3).

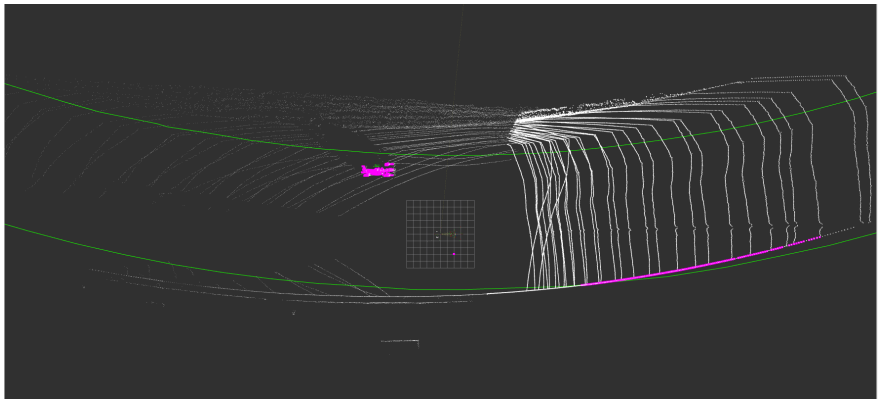


Figure 6.3: Effect of inaccurate localization on boundary filtering. Poor heading has rotated the boundary (green), causing a part of the track wall to be classified as a car (pink).

Pitlane Perception

During pit entry and exit, we dynamically swap the boundary polygon to the pitlane curb edges, enabling the same stack to detect other vehicles in the pit lane without code changes.

Reflectance and Existence Filtering

The newer Luminar Iris LiDARs report per-return reflectance and an `existence_prob` metric. To suppress trailing "backwash" from dust and hot exhaust, we discard points below a reflectance threshold or with low `existence_prob`. This post-filter cleans spurious, low-confidence returns and focuses detection on solid vehicle surfaces.

6.3 Radar-only Stack

Our radar-only pipeline was developed and evaluated on Continental ARS-548 radars (followed by trials on newer ZF units). Each radar return provides not only range and Doppler velocity, but a rich set of per-detection attributes:

- **range**: radial distance ρ
- **radial_velocity**: Doppler speed v_r

- **radar_cross_section** (RCS)
- **ambgt_id**: ambiguity group ID
- **signal_noise_ratio** (SNR)
- **existence_prob**: detection confidence
- **multi_target_prob**: probability of unresolved multi-path returns
- **received_signal_strength**: raw signal amplitude
- **az_ang_std_dev**, **elev_ang_std_dev**: angular measurement noise
- **azi_qual**, **elev_qual**, **range_qual**, **rad_velo_qual**: quality metrics

Velocity Transformation

Raw Doppler radial velocity v_r is measured along the ray from the radar to the target. To identify fast-moving vehicles, we convert v_r into an *absolute parallel velocity* $v_\parallel$ along the vehicle's heading:

$$v_\parallel = \frac{v_r \; - \; \mathbf{v}_{\text{ego}} \cdot \hat{\mathbf{r}}}{\hat{\mathbf{r}} \cdot \hat{\mathbf{h}}} \quad \text{where} \quad \hat{\mathbf{r}} = \frac{\mathbf{X}}{\|\mathbf{X}\|}, \quad \hat{\mathbf{h}} = \frac{\mathbf{v}_{\text{ego}}}{\|\mathbf{v}_{\text{ego}}\|}.$$

Here $\mathbf{X}$ is the 3D range vector to the detection, $\hat{\mathbf{r}}$ its unit direction, and $\mathbf{v}_{\text{ego}}$ the ego-vehicle velocity vector. The numerator subtracts the ego-component of the radial Doppler, yielding the target's Doppler relative to ground; dividing by $\cos\theta = \hat{\mathbf{r}} \cdot \hat{\mathbf{h}}$ rotates that along the track axis. A $|v_\parallel| > 20$m/s (≈45mph) is a strong car indicator, since few other objects move so fast.

Decision-Tree Classifier

We train a transparent decision tree on per-point features [19]:

$$\{\, \rho,\ \phi,\ v_r,\ v_\parallel,\ \text{SNR},\ \text{RCS},\ p_{\text{exist}\ \ldots} \},$$

where ρ is range, ϕ is azimuth, and p_{exist} the existence probability. To handle the extreme class imbalance ($> 95\%$ background), we apply class-rebalancing weights during training rather than downsampling positives. The resulting tree yields explicit rules (e.g. "if $v_\parallel > 25$ m/s and SNR > 10 dB then car") that are human-auditable and can be pruned or adjusted online.

To prevent *cheating* (overfitting to scene-specific cues, like in Figure 6.4), we omitted purely spatial fields (e.g. X, Y, Z) from training and diversified our dataset across tracks, speeds, and sensor orientations.

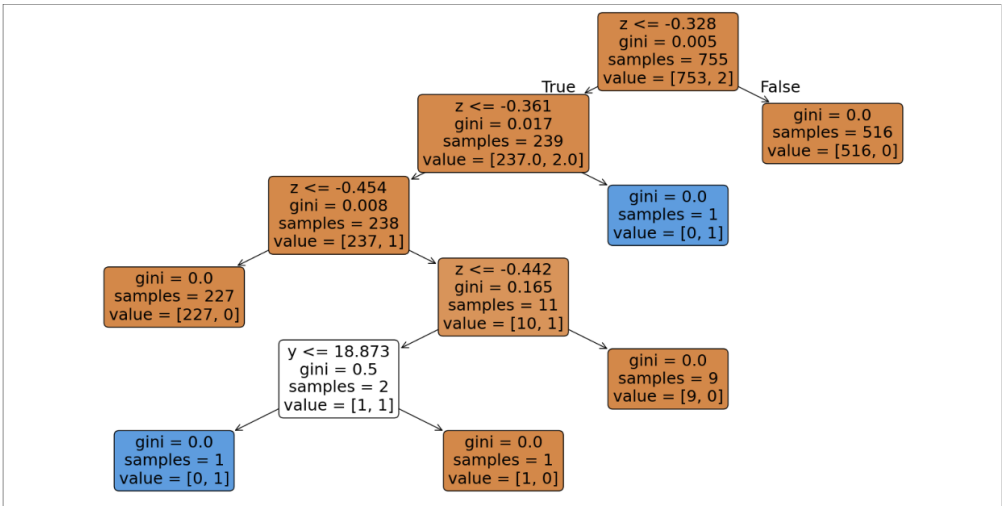


Figure 6.4: Example of the decision tree cheating. It is using rules based on (X, Y, Z).

Specular Reflection Mitigation

Specific surfaces (guardrails, walls) produce *specular reflections* that mimic a real car's Doppler and SNR, leading to a ring of ghost detections at the same range. To suppress these:

1. After decision-tree filtering, group detections by their rounded range bin.
2. In each bin, retain only the largest cluster of points (true car) and discard smaller "ghost" clusters.

We also observed direct reflections off the ego vehicle itself, which appear as high-speed returns; these are similarly removed by range-bin clustering. Additionally, radar's coarse angular resolution (several degrees) further blurs true object shape, so cluster-based rules are essential.

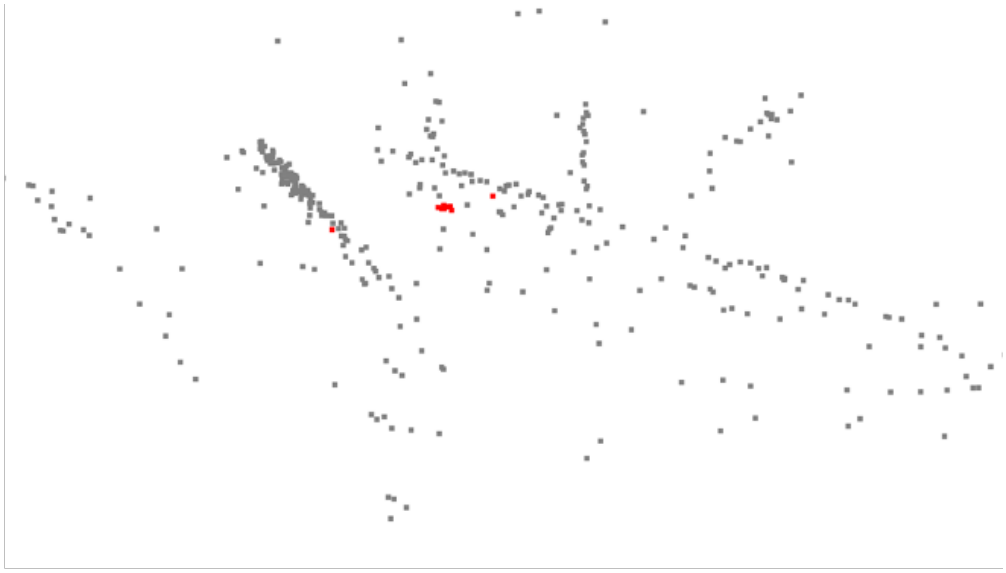


Figure 6.5: Output pointcloud of the decision tree. Inliers are shown in red, and reflections from the walls ('streaks') are also clearly visible.

Training and Labeling

Because manual labeling of radar returns is infeasible, we generate ground truth by pairing each radar detection with the nearest LiDAR-clustered car surface (from the LiDAR-only stack). Detections within a 0.5 m radius of a LiDAR car cluster center are labeled *positive*, and all others *negative*. This automatic labeling yields thousands of examples per session.

Limitations and Future Work

- **Sensor noise:** Our prototype used Continental radars with high measurement noise. The new ZF radars promise much lower noise and finer Doppler resolution, enabling more generalizable trees.
- **Specular ambiguity:** Specular returns still confound the tree. We plan to integrate multi-frame association and geometric track-continuity checks.
- **Dependence on LiDAR labels:** The current training pipeline requires a LiDAR-based stack for supervision. We aim to explore weakly supervised and self-supervised radar labeling to reduce that dependency.

Despite these challenges, the radar-only stack provides a valuable redundant perception channel, especially under sensor occlusion and high range.

6.4 Camera-only Stack

To complement our LiDAR- and radar-based pipelines, we implement a purely camera-based stack that fuses instance segmentation (from YOLOv8) with monocular depth estimation to recover 3D opponent poses even in LiDAR/radar blindspots. Unlike depth-from-stereo or LiDAR-fusion approaches, this pipeline relies solely on a single RGB feed and a fast, transformer-based depth estimator, enabling full 360° coverage with minimal additional hardware. The output of the depth estimator, and hence the camera-only pipeline as a whole, is also a pointcloud that is passed into the tracker.

Monocular Depth Estimation Model

Architecture (DPT) We adopt the Dense Prediction Transformer (DPT) architecture [33] for dense depth regression, as it combines global attention with multi-scale feature fusion to produce high-quality depth maps. Input images are tokenized via a ViT-style patch embedder (or hybrid ResNet backbone), augmented by positional embeddings, and passed through successive transformer stages. At each stage, features are reassembled into spatial maps and fused in a U-Net–like decoder to yield pixel-wise depth predictions (see Figure 6.6). The global receptive field of the transformer encoder resolves long-range ambiguities critical for racing, while the decoder's fusion modules preserve fine-grained details required for accurate opponent localization.

Figure 6.6: Output of depth estimation. Darker pixels are nearer.

Training Regime We first pretrain the DPT depth backbone on KITTI [12] for broad driving scenarios, then finetune on our racing dataset using LiDAR-derived depths as supervision. During finetuning, we emphasize consistency across scales and focus the loss within YOLOv8 car regions to maximize opponent localization accuracy. To close the sim-to-real gap, we apply domain randomization and style-transfer augmentations on synthetic sequences.

Metric Calibration Raw DPT outputs are relative depths; we calibrate them to metric values via a distance-balanced linear regression, ensuring equal weighting of near/far ranges and preserves metric accuracy up to 100m.

Backup Depth Estimation via Bounding-Box Geometry

As a lightweight fallback, we exploit the fact that all opponent racecars share a known real-world height H. From projective geometry, the height of a detected car in image pixels h_{pix} relates inversely to its distance Z:

$$h_{\text{pix}} \propto \frac{H}{Z} \implies Z \approx \frac{f\,H}{h_{\text{pix}}},$$

where f is the camera focal length in pixel units. Thus, measuring the YOLOv8 bounding-box height alone yields an approximate depth estimate.

Advantages and Limitations

- *Extremely efficient*: only requires reading box height and a single multiplication.
- *Head-on reliability*: works well on rear-facing cameras when opponents appear roughly frontal.
- *Failure modes*: side or diagonal views distort apparent height; foreshortening leads to over- or underestimation.

By using this geometric fallback on the rear camera, we achieve a cheap, yet sufficiently accurate, depth cue whenever the learning-based DPT model is unavailable or uncertain.

Chapter 7

Results

The development of our perception stack unfolded under tight on-track deadlines, leaving little room for extensive offline experimentation. Much of the early tuning relied on qualitative "looks good" assessments. In this chapter, we formalize a set of quantitative evaluation metrics and present preliminary results comparing our two fusion approaches.

7.1 Evaluation Metrics

Existing Metrics

Current self-driving benchmarks typically report only:

- **Positional accuracy:** Mean or root-mean-square error (RMSE) of estimated 3D positions against ground truth.
- **CPU utilization:** Percentage of CPU cycles consumed by the perception pipeline.

While useful, these metrics do not capture the full performance profile needed for high-speed racing.

Proposed Metrics

To more deeply characterize our stack, we define additional measures:

- **Orientation error:** Angular deviation of the estimated heading from ground truth.
- **Velocity error:** Difference between estimated and true longitudinal speed.
- **Trajectory smoothness:** Variance of successive position/velocity estimates, penalizing jitter.
- **Detection precision/recall:** Per-frame IOU of 3D bounding boxes against transponder or hand-labeled reference.

- **GPU load:** Fraction of GPU cycles used by deep models (YOLOv8, depth estimator).
- **Module-level diagnostics:** Intermediate errors (e.g. calibration reprojection error, segmentation mask accuracy) to isolate failure modes.

7.2 Positional Accuracy Comparison

Using recently instrumented transponder data on the opponent car as ground truth, we compare early-stage and late-stage fusion on two representative maneuvers: following and being overtaken.

Maneuver #1: Following

This section compares the perception stack output against the transponder positions over time when the ego car trails the opponent at a varying distance.

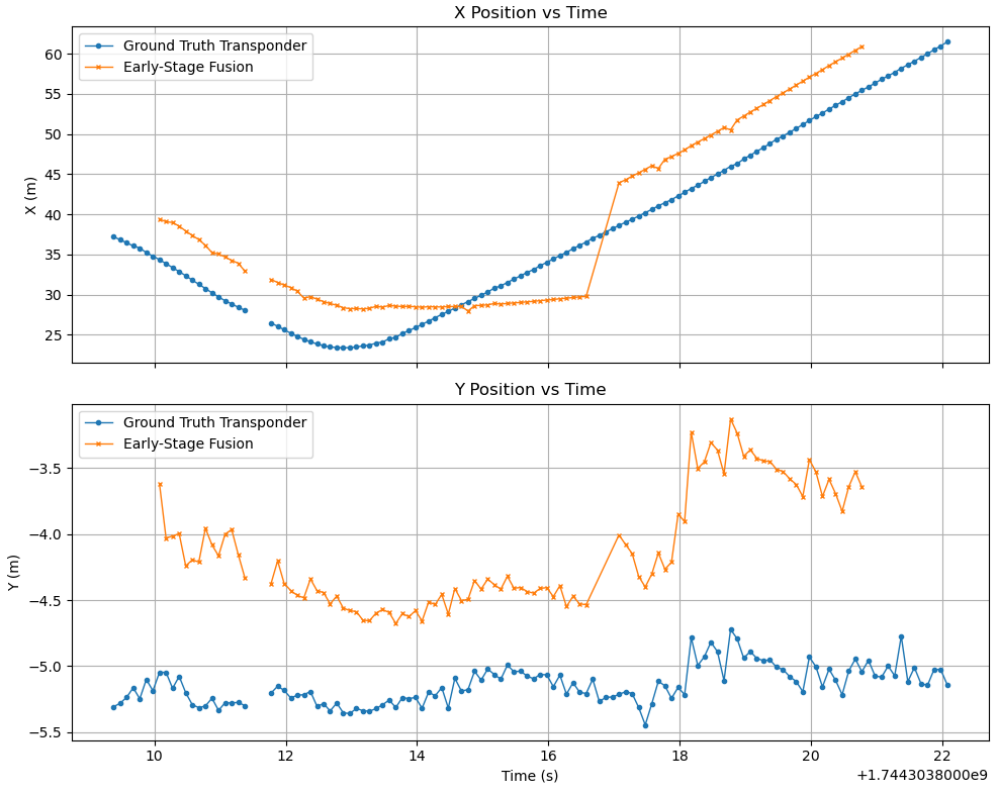


Figure 7.1: Early-stage output vs. transponder position during a following maneuver.

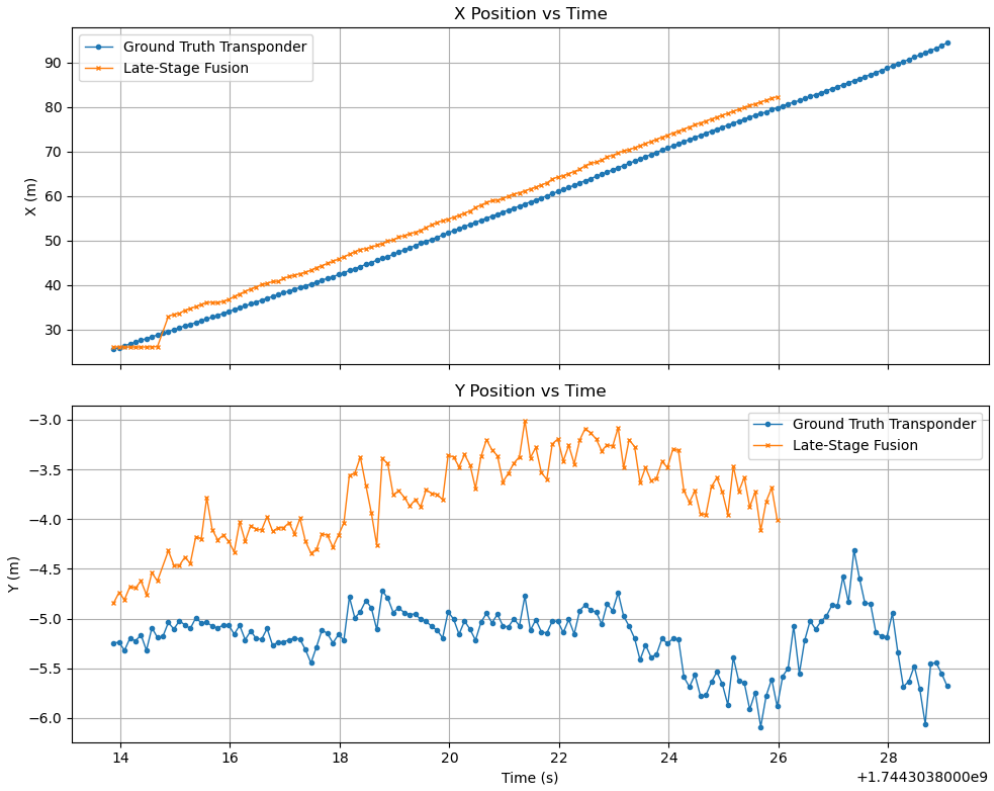


Figure 7.2: Late-stage output vs. transponder position during a following maneuver.

In the early-stage case (Figure 7.1), YOLOv8 fails to detect the trailing car at around t=13s, which triggers the EKF lane-keeping hallucination. The hallucinated trajectory underestimates longitudinal velocity, causing a growing lag of up to 15m before the next true detection "snaps" the estimate back. This highlights how YOLOv8 is a single point of failure in the early-stage method. False negatives from YOLOv8 are attempted to be "filled in" downstream but the errors generated because of this are very significant.

By contrast, late-stage fusion (Figure 7.2) maintains tracking from 25m to beyond the 80m range, with only minor jitter. This clearly outperforms the early-stage method.

Additionally, in both cases, there seems to be a constant offset in the longitudinal and latitudinal directions of roughly 3m. This might be due to fixed transform inaccuracies from the transponder data being transmitted from the other car.

Maneuver #2: Overtaking

This section evaluates a scenario where the opponent passes the ego car from behind to the right.

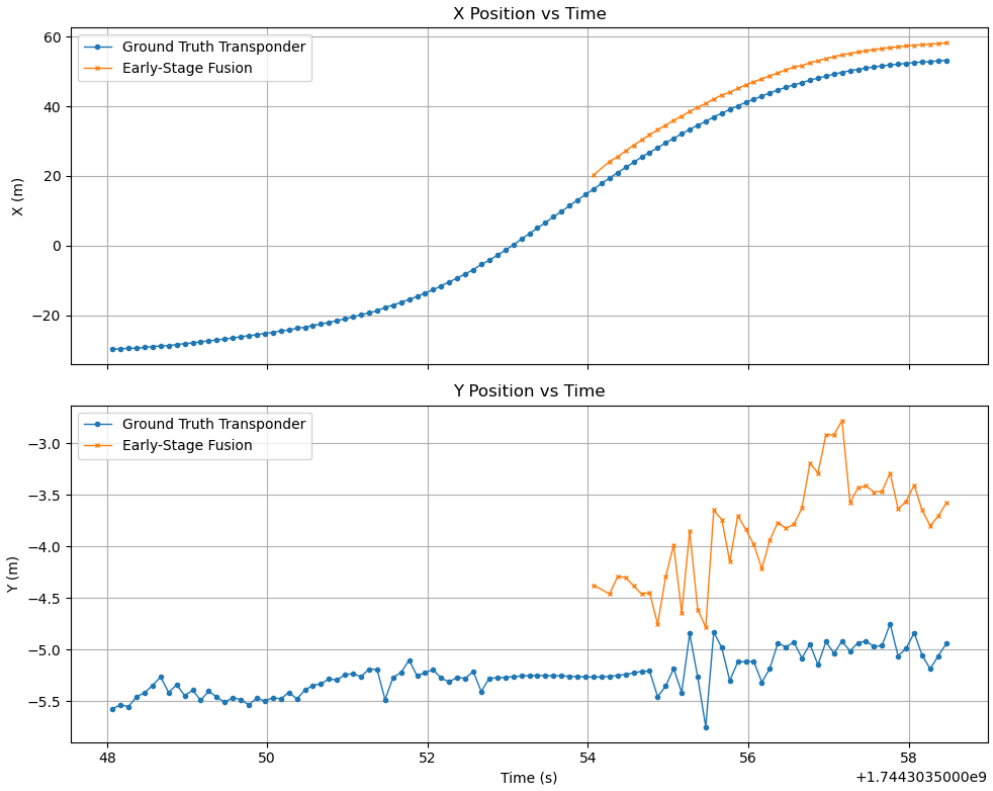


Figure 7.3: Early-stage output vs. transponder position during an overtaking maneuver.

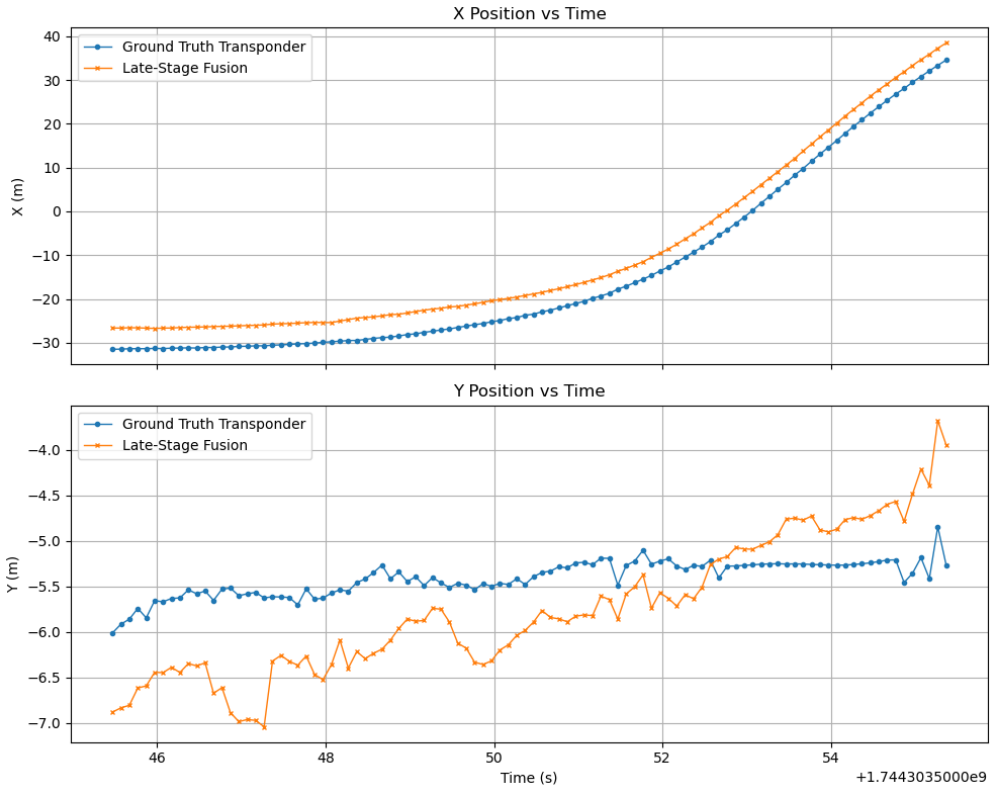


Figure 7.4: Late-stage output vs. transponder position during a overtaking maneuver.

The early-stage method (Figure 7.3) only detects the overtaking opponent after it has fully passed by 20m. This is largely in part due to the lack of LiDAR-camera calibration between the right side LiDAR and rear camera. Even though YOLOv8 detected the car behind the ego car, the lack of calibration for the sensor pair meant that projection could not occur. Hence, no points enter the EKF until post-overtake.

On the other hand, the late-stage method (Figure 7.4) doesn't depend on any such external calibration. It is therefore able to smoothly track the overtaking car from 30m behind to 40m ahead, after which the experiment was terminated. The results show that the car would have most likely continued to be tracked until 80m, similar to the following experiment in Figure 7.2.

Again, constant errors in longitudinal and latitudinal positions between the estimated and ground-truth values are likely due to a fixed transform issue.

7.3 CPU Utilization

We measured end-to-end CPU load, filtering for perception processes only, on the AV-24 car (24-core dSPACE Autera Autobox with Nvidia A5000 GPU) and on a reference laptop (32-core Intel i9 with Nvidia 4090 GPU):

Method	**Car CPU (%)**	**Laptop CPU (%)**
Early-Stage Fusion	30.21	10.34
Late-Stage Fusion	34.26	11.59

Table 7.1: Average CPU utilization of the perception stack.

Comparing the CPU loads between the two methods reveals a very small difference. The slight advantage of early-stage fusion likely stems from its use of CUDA kernels for the LiDAR–camera projection (GPU time isn't reflected in these CPU numbers), whereas the late-stage radar and LiDAR pipelines remain fully CPU-bound. Given the small gap, a detailed per-process profile including the sensor driver overhead would be required to pinpoint the true sources of CPU usage.

Note that these CPU figures exclude the sensor driver processes, which in practice represent a substantial fraction of total CPU usage. In addition, these values are approximate and will vary according to driver configuration, data rates, and system load.

7.4 Takeaways

- **Robustness:** Late-stage fusion markedly reduces positional error in both following and overtaking, especially when single modalities fail.
- **Efficiency:** CPU profiling is comparable between the two methods - no definitive conclusion can be made yet.
- **Modularity:** Isolating sensor failures into separate stacks enables graceful degradation - e.g. radar can sustain tracking when cameras or LiDAR temporarily drop detections.
- **Next steps:** Future work will quantify orientation and velocity accuracy, evaluate precision/recall, and benchmark GPU usage to complete a holistic performance profile.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, we have addressed the challenge of robust opponent detection and state estimation for high-speed autonomous racing under stringent real-time, accuracy, and reliability requirements. We began by formulating the problem - fixed track, known localization, and exclusive racecar occupancy - and surveying both classical modular perception and end-to-end learning paradigms. Building on these foundations, we developed and compared two fusion architectures:

Early-Stage Fusion a tightly-coupled LiDAR-camera projection pipeline leveraging precise extrinsic calibration, hybrid frustum filtering, segmentation-mask refinement, and post-processing (range-bin SVD and dynamic downsampling). This approach yields dense 3D point-cloud "cuts" of each opponent surface, but proved highly sensitive to calibration drift, time-sync errors, and single-mode failure.

Late-Stage Fusion three independent modality stacks - Autoware LiDAR filtering, radar decision-tree classification, and YOLOv8 with monocular depth estimation - whose 3D detections are merged in an EKF tracker with lane-keeping hallucination and confidence-based pruning. This design delivers redundancy, calibration resilience, and asynchronous fusion, achieving superior robustness under sensor degradation while maintaining acceptable latency on constrained hardware.

Initial experiments against transponder ground truth demonstrated that late-stage fusion consistently outperforms early-stage fusion in terms of positional accuracy without sacrificing real-time performance. The modular tracker innovations - Frenet-frame hallucination and dynamic confidence scoring - proved critical to sustaining object tracks through occlusions and noisy observations.

8.2 Future Work

Despite these advances, several avenues remain to further enhance both performance and generality:

- **LiDAR-Only Stack Optimizations:**
  - *Dynamic Track Boundary Detection:* Replace the existing boundary-transformer (which relies on a priori map edges and upstream localization) with an on-the-fly, scan-based method. For example, apply a Hough-transform or RANSAC-based line/curve fitting directly to the LiDAR's 2D planar projection [11] to extract left/right boundary hypotheses. This eliminates the dependency on global pose and pre-mapped edges, allowing the stack to remain track-agnostic and to adapt to subtle variations in boundary geometry (e.g., kerbs, rumble strips) without manual configuration.
  - *Polar-First Processing:* Retain and process LiDAR returns in their native range-azimuth-elevation representation, avoiding the costly back-and-forth transformations into Cartesian (x, y, z) space. By operating in polar coordinates from the outset, we can:
    - Implement ground segmentation via range and vertical angle thresholds directly on the polar array.
    - Perform clustering and object separation using contiguous azimuth bins, reducing neighbor-search overhead.
    - Leverage lookup tables for angular to metric conversion only when required for fusion, cutting down per-point trigonometric computations.
  - *Deep Learning–Based Object Extraction:* Transition from heuristic filters and Euclidean clustering to a lightweight, point-based neural network (e.g. RandLA-Net [16] or PointTransformerV3 [42]) fine-tuned for the high-speed racing domain. This will require building a scalable labeling framework to generate per-point annotations, as shown by initial experiments with an off-the-shelf RandLA-Net (see Figure 8.1) which highlights the need for domain-specific retraining.

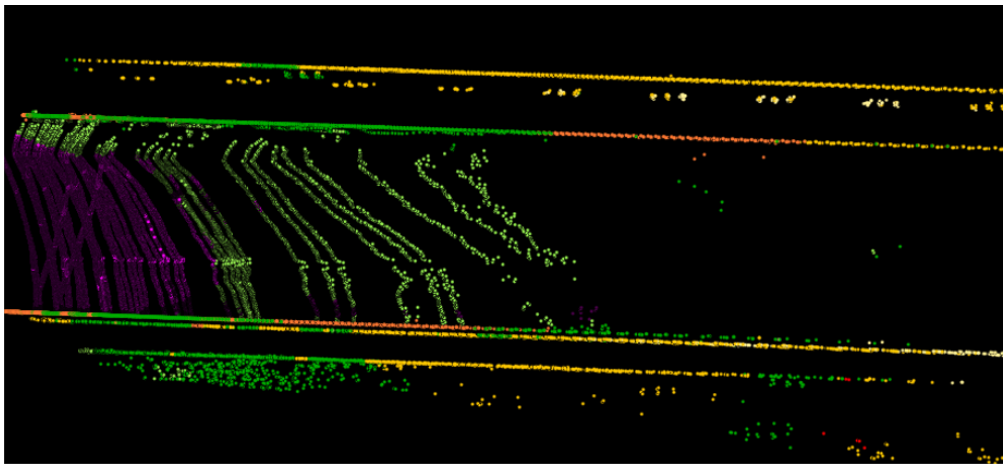


Figure 8.1: Unfinetuned RandLA-Net on racing LiDAR data: the network fails to segment opponent vehicles and track walls accurately.

- **Radar-Only Stack Enhancements:**
  - *Learned Classifiers:* supplement or replace the decision tree with a small, temporal neural network (e.g. one-dimensional CNN or LSTM) that fuses multi-frame Doppler signatures to suppress specular reflections and ego-vehicle ghosts.
  - *Self-Supervised Labeling:* exploit cycle-consistency between LiDAR and radar tracks to bootstrap radar training online, reducing dependency on LiDAR supervision.
- **Camera-Only Depth Improvements:**
  - *Hybrid Geometric–Learning Fusion:* combine the bounding-box height heuristic with the transformer-based depth estimator, dynamically selecting the most confident estimate per object.
- **Advanced Tracking and Prediction:**
  - *Long-Term Occlusion Management:* Extend the existing Frenet-based hallucination, effective for short dropouts (< 5s), with an image-based re-identification module that leverages vehicle livery patterns. This enables persistent track association through extended occlusions (30s+), allowing the system to maintain per-opponent state variables (e.g., push-to-pass budget) and feed higher-level strategic planners.
  - *Trajectory Forecasting:* Incorporate learned motion models (e.g., LSTM-based predictors or Gaussian process regression) alongside the EKF to generate multi-second forecasts of each opponent's trajectory. By providing anticipatory state

estimates, this enhancement supports more aggressive overtaking, robust defensive maneuvers, and safer collision avoidance at high speeds.

- **Continuous Sensor Extrinsic Calibration:**
  - *Online Self-Calibration:* Deploy scene-based calibration algorithms, such as maximizing mutual information between camera imagery and LiDAR intensity returns, to refine extrinsic transforms in real time. Exploiting overlapping FOVs and static track features, this approach automatically compensates for extrinsic drift without the need for calibration targets and can run continuously or during pre-race warm-up [24].

Pursuing these directions will enable an even more resilient, accurate, and flexible perception stack - one capable of supporting advanced autonomy strategies and pushing state-of-the-art performance in high-speed racing.

Bibliography

[1] Autoware Foundation. *Minimum-update Successive Shortest Path (muSSP) Data Association*. https://autowarefoundation.github.io/autoware_universe/main/perception/autoware_multi_object_tracker/. Autoware Universe documentation. 2024.

[2] Yaakov Bar-Shalom and Xiaoming Li. "Multitarget-Multisensor Tracking: Principles and Techniques". In: *YBS Publishing* (1995). Chapters on track-level fusion and EKF-based hypothesis merging.

[3] Johannes Betz and Hongrui Zheng. "Autonomous Vehicles on the Edge: A Survey on Autonomous Vehicle Racing". In: *arXiv* (2022). eprint: 2202.07008.

[4] Johannes Betz et al. "TUM Autonomous Motorsport: An Autonomous Racing Software for the Indy Autonomous Challenge". In: *Journal of Field Robotics* 40.4 (2023), pp. 783–809. DOI: 10.1002/rob.22153. URL: https://doi.org/10.1002/rob.22153.

[5] Mariusz Bojarski et al. "End to End Learning for Self-Driving Cars". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2016, pp. 1–9. DOI: 10.48550/arXiv.1604.07316.

[6] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. "The DARPA Urban Challenge: Autonomous Vehicles in City Traffic". In: *Springer Tracts in Advanced Robotics*. Ed. by Martin Buehler, Karl Iagnemma, and Sanjiv Singh. Vol. 56. Berlin, Heidelberg: Springer, 2009. ISBN: 978-3-642-00236-1.

[7] Holger Caesar et al. "nuScenes: A Multimodal Dataset for Autonomous Driving". In: *arXiv preprint arXiv:1903.11027* (2020). URL: https://www.nuscenes.org/.

[8] Xiaozhi Chen et al. "Multi-View 3D Object Detection Network for Autonomous Driving". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 1907–1915.

[9] Ashwat Chidambaram. "Leveraging Zero-Shot Sim2Real Learning to Improve Autonomous Vehicle Perception". Master's thesis, EECS Department. MA thesis. University of California, Berkeley, 2024.

[10] Edward R. Dougherty and Roberto A. Lotufo. *Hands-On Morphological Image Processing*. SPIE Press, 2003.

[11] Bertrand Douillard et al. "On the Segmentation of 3D LIDAR Point Clouds". In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 2798–2805. DOI: 10.1109/ICRA.2011.5980408.

[12] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: *Proceedings of the International Journal of Robotics Research (IJRR) Workshop on Performance Evaluation of Tracking and Surveillance*. 2013. URL: http://www.cvlibs.net/datasets/kitti/.

[13] Ross Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 580–587.

[14] David L. Hall and James Llinas. *Handbook of Multisensor Data Fusion*. Second. CRC Press, 2001.

[15] Michael Himmelsbach, Felix V. Hundelshausen, and Hans-Joachim Wuensche. "Fast segmentation of 3D point clouds for ground vehicles". In: *2010 IEEE Intelligent Vehicles Symposium*. 2010, pp. 560–565. DOI: 10.1109/IVS.2010.5548059.

[16] Qingyong Hu et al. "RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 11108–11117. DOI: 10.1109/CVPR42600.2020.01112.

[17] Wei Huang et al. "Baidu Apollo: An Open, Reliable and Efficient Platform for Autonomous Driving". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018, pp. 1372–1377.

[18] Sunghoon Hwang et al. "Multispectral Pedestrian Detection: Benchmark Dataset and Baseline". In: *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1037–1045.

[19] Eugin Hyun and YoungSeok Jin. "Doppler-Spectrum Feature-Based Human–Vehicle Classification Scheme Using Machine Learning for an FMCW Radar Sensor". In: *Sensors* 20.7 (2020), p. 2001. DOI: 10.3390/s20072001.

[20] Junaid Janai et al. "Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art". In: *Foundations and Trends in Computer Graphics and Vision* 12.1–3 (2020), pp. 1–308.

[21] Rudolf E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME—Journal of Basic Engineering* 82 (1960), pp. 35–45.

[22] Sunwoo Kim, Soohyun Kim, and Seungryong Kim. "Deep Translation Prior: Test-Time Training for Photorealistic Style Transfer". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2022, pp. 10484–10492.

[23] Yangwoo Kim et al. "Assessment of an Autonomous Racing Controller: A Case Study From the Indy Autonomous Challenge's Simulation Race". In: *The Journal of Technology, Management, and Applied Engineering* 39.2 (2023), pp. 1–19. DOI: 10.31274/jtmae.15670. URL: https://doi.org/10.31274/jtmae.15670.

[24] Jesse Levinson and Sebastian Thrun. "Automatic Online Calibration of Cameras and Lasers". In: *Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*. 2013.

[25] Jesse Levinson and Sebastian Thrun. "Efficient 3-D Spatial Occupancy Grid Generation for Large-Scale Mapping". In: *Robot. Sci. Syst. (RSS)*. 2007.

[26] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016, pp. 21–37.

[27] Brian Paden et al. "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles". In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), pp. 33–55. DOI: 10.1109/TIV.2016.2578706.

[28] Gaurav Pandey, Mario H. Ang, and Trevor Bailey. "Automatic Extrinsic Calibration of a Camera and a 3D Laser Scanner". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012, pp. 1990–1997. DOI: 10.48550/arXiv.1904.12433.

[29] Jade Philion and Sanja Fidler. "Lift, Splat, Shoot: Encoding Images from Pixels to Points for 3D Object Detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 467–476.

[30] Dean A. Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network". In: *Proceedings of the 1989 Conference on Neural Information Processing Systems (NIPS)*. Denver, CO, 1989, pp. 305–313.

[31] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*. http://www.ros.org. Kobe, Japan, 2009.

[32] Ayoub Raji et al. "Motion Planning and Control for Multi Vehicle Autonomous Racing at High Speeds". In: *arXiv preprint arXiv:2207.11136* (2022). URL: https://arxiv.org/abs/2207.11136.

[33] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. "Vision Transformers for Dense Prediction". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 12179–12188.

[34] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788.

[35] Nicolas Scheiner et al. "Object Detection for Automotive Radar Point Clouds—A Comparison". In: *AI Perspectives* 3 (2021). DOI: 10.1186/s42467-021-00012-z.

[36] Pei Sun et al. "Scalability in Perception: Lessons from the Waymo Open Dataset". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 2446–2454.

[37] Sebastian Thrun et al. "Stanley: The Robot that Won the DARPA Grand Challenge". In: *Journal of Field Robotics* 23.9 (2006), pp. 661–692.

[38] Ultralytics. *YOLOv8: Real-Time Instance Segmentation and Object Detection*. https://docs.ultralytics.com/. Accessed May 2025. 2023.

[39] Sean Vora et al. "PointPainting: Sequential Fusion for 3D Object Detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 4604–4612.

[40] Moritz Werling et al. "Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenet Frame". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2010, pp. 987–993.

[41] Alexander Wischnewski et al. "A Tube-MPC Approach to Autonomous Multi-Vehicle Racing on High-Speed Ovals". In: *IEEE Transactions on Intelligent Vehicles* 8.1 (2022), pp. 368–378. DOI: 10.1109/TIV.2022.3169986. URL: https://doi.org/10.1109/TIV.2022.3169986.

[42] Xiaoyang Wu et al. "Point Transformer V3: Simpler, Faster, Stronger". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024.

[43] Yan Yan et al. "BEVFusion: Multi-Camera Bird's-Eye-View Fusion for 3D Object Detection". In: *arXiv preprint arXiv:2112.11790* (2021).

[44] Erdem Yurtsever et al. "A Survey of Autonomous Driving: Common Practices and Emerging Technologies". In: *IEEE Access* 8 (2020), pp. 58443–58469. DOI: 10.1109/ACCESS.2020.2983149.

[45] Fan Zhang, Chen Deng, and Changjiu Wang. "Obstacle Detection and Ranging Using Automotive Ultrasonic Sensors: A Review". In: *Sensors* 18.1 (2018), p. 281.

[46] Zhengyou Zhang. "A Flexible New Technique for Camera Calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 22. 11. 2000, pp. 1330–1334. DOI: 10.1109/34.888718.

[47] Xingyi Zhou, Vladlen Koltun Wang, and Philipp Krähenbühl. "Objects as Points". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 7794–7803.