Autonomous Articulated Vehicle Control via Deep Reinforcement Learning and High-Dimensional Perception

by

Benjamin David Schnapp

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2026

© Benjamin David Schnapp 2026

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The learned controller, the simulation environment, and the RL bridge presented in this thesis are the author's own work. The following contributions of others are acknowledged.

- The path-following study that this work extends on was a joint graduate course project with R. Tymkow [1].
- S. Huang developed the onboard LiDAR hitch-angle estimator of chapter B and contributed the Autoware planning components of the conventional Electrans autonomy stack, which runs as an alternative to the RL bridge presented here.
- A. Beaudette, J. Graansma, and M. Duthie contributed the mechanical and electrical work on the AgileX platform and the Electrans shunt truck.
- The infrastructure sensor nodes and their perception stack were developed by F. Yang and M. Ning. The author contributed only the interface that delivers their fused obstacle list to the vehicle's ROS2 stack.

Abstract

Autonomous control of articulated tractor-trailer vehicles is difficult because the passive hitch adds a revolute degree of freedom whose reverse dynamics are open-loop unstable, driving the trailer toward the jackknife regime. This thesis develops and evaluates a learning-based controller for this setting, targeting the confined low-speed maneuvers typical of distribution-yard and loading-dock operations, and carries it from simulation onto physical hardware.

A Twin Delayed Deep Deterministic Policy Gradient (TD3) agent is trained in a custom Gymnasium simulation of a shunt-truck tractor-trailer. Controlled ablations isolate the learning algorithm, reward formulation, and observation modality. TD3 under a multiplicative reward that jointly penalizes tracking error and hitch articulation produces the highest and most consistent reverse completion while also reducing tracking and articulation errors. A compact engineered state representation is more reliable than higher-dimensional LiDAR or bird's-eye-view (BEV) image observations under an otherwise identical training recipe. Training a policy directly on the raw image fails, but controlled experiments show that the image contains sufficient task-relevant information and identify the off-policy actor-critic training process as the primary source of failure. Geometry distillation addresses this failure by training a supervised encoder to recover the vehicle's path geometry from the BEV image and freezing that encoder during policy optimization. The resulting policy restores reliable from-scratch learning to near engineered-state performance in both forward and reverse lane following. The same approach extends to obstacle navigation, where the image-derived representation provides both an avoidance path and a measure of layout difficulty, allowing the learned controller to follow feasible detours and stop when a layout should not be attempted.

The geometry-encoded BEV policies are deployed on a physical articulated platform, an AgileX Hunter SE with a scale trailer. Deployment uses a stripped-down Autoware stack whose planning and control layers are replaced by an RL bridge that reproduces the observation and action interfaces used during training. Four formal lane-following trials, two forward and two reverse, were conducted and all reached the goal under policy control. Forward tracking transferred with low error, while reverse tracking exhibited a sustained, steering-rate-limited articulation oscillation that was absent from simulation. The hardware results therefore demonstrate physical transfer of the learned controller while identifying reverse articulation dynamics as the dominant remaining sim-to-real limitation.

Acknowledgements

I would first like to thank my supervisors, Professor Amir Khajepour and Professor Arash Arami, for their guidance and technical expertise on this work and throughout my graduate studies at the University of Waterloo.

I would also like to thank the members of the Mechatronic Vehicle Systems Lab that I learned from and worked with, specifically Shucheng (Bruce) Huang, Ryan Tymkow, and Alexandre Beaudette. I would also like to thank the MVSL technicians Jeff Graansma and Mike Duthie for their support in the lab and with the hardware platforms.

Thank you to my family and friends for their continuous support. Thank you to my dog, Cassie, for reminding me to take breaks and go on walks, where I was able to think with a clear head. And finally, to my wife, Lianna, thank you for your patience and encouragement over these two years, I couldn't have done this without you.

Generative Artificial Intelligence Acknowledgement

Generative artificial intelligence tools were used during the development of this thesis as review and software-development aids only. These tools were primarily used to identify problems, inconsistencies, or ambiguities. They were not used to originate the research questions, methodology, experimental design, analysis, conclusions, or other intellectual contributions presented in this thesis.

All thesis text was written by me. Generative artificial intelligence tools were used to review material, provide feedback on clarity, organization, and technical consistency, and identify possible errors or discrepancies.

Generative artificial intelligence tools were used more extensively during software development, primarily to assist with debugging and to accelerate the implementation of my ideas and algorithms. The underlying concepts, system designs, experimental decisions, and research contributions are my own.

I reviewed, tested, and validated all artificial-intelligence-assisted work and accept full responsibility for the accuracy, integrity, and content of this thesis and its associated software.

Contents

Author's Declaration ii

Statement of Contributions iii

Abstract iv

Acknowledgements v

Generative Artificial Intelligence Acknowledgement vi

List of Figures x

List of Tables xi

List of Abbreviations xiii

List of Symbols xiv

1 Introduction 1

1.1 Motivation . . . 1
1.2 Thesis Contributions . . . 2
1.3 Outline . . . 3

2 Literature Review 4

2.1 The Articulated-Vehicle Control Problem . . . 4
2.2 Classical Model-Based Control . . . 5
2.3 Learning-Based Control . . . 7
2.4 Perception and Representation Learning . . . 9
2.5 Sim-to-Real Transfer . . . 10
2.6 Safety and Standards . . . 11
2.7 Summary . . . 11

3 Modeling, Simulation, and Problem Formulation **12**
3.1 Tractor Dynamics . . . 12
3.2 Trailer Articulation and Reverse Instability . . . 14
3.3 Vehicle Parameters . . . 16
3.4 Simulation Environment . . . 18
3.4.1 Control Policy Interface . . . 20
3.5 Environment Tasks . . . 20
3.5.1 Lane Following . . . 21
3.5.2 Obstacle Avoidance . . . 21
3.6 Scalable Simulation: GPU-Parallel Batched Environments . . . 21
3.6.1 Batched Environment Design . . . 22
3.6.2 Numerical Parity . . . 23
3.6.3 Throughput and Scaling . . . 23
3.7 Summary . . . 25

4 Learning Framework and Experimental Results **26**
4.1 Learning Framework Overview . . . 26
4.2 Learning Algorithm and Training Configuration . . . 28
4.2.1 Twin Delayed Deep Deterministic Policy Gradient . . . 28
4.2.2 Training Configuration . . . 28
4.3 Evaluation Protocol . . . 30
4.4 Reward Formulation and Selection . . . 31
4.4.1 Reward Variants . . . 31
4.4.2 Selecting the Algorithm and the Reward . . . 34
4.5 Observation and Perception Design . . . 37
4.5.1 Image Encoding and Representation Learning . . . 38
4.5.2 Observation Space Summary . . . 39
4.6 Observation-Space Ablation . . . 40
4.7 Training High-Dimensional-Observation Policies . . . 42
4.7.1 An Off-Policy Instability Hypothesis . . . 43
4.7.2 Representation Pretraining and Warm-Starting the Policy . . . 43
4.7.3 Geometry Distillation . . . 44
4.7.4 Isolating the Failure Mode . . . 45
4.7.5 What the Frozen Encoder Recovers . . . 47
4.8 Obstacle Navigation . . . 49
4.8.1 Obstacle Reward and Curriculum Design . . . 50

4.8.2 Direct Obstacle Encoding . . . 51
4.8.3 Observing the Avoidance Path . . . 51
4.8.4 The Stop Decision and Forward Results . . . 52
4.8.5 State-Dependent Difficulty in Reverse . . . 52
4.8.6 Recovering the Difficulty from the Image . . . 55
4.8.7 Deterministic Safety Gate . . . 56
4.9 Comparison with Classical Controllers . . . 58
4.9.1 Information Parity between Learned and Classical Controllers . . . 60
4.10 Failure Modes and Deployment Relevance . . . 61
4.11 Summary . . . 61

5 Sim-to-Real Deployment **63**
5.1 Overview . . . 63
5.2 Hardware Platform . . . 64
5.3 Software Architecture . . . 65
5.3.1 Middleware and Framework . . . 65
5.3.2 Relationship to the Conventional Articulated Pipeline . . . 66
5.3.3 RL Bridge Node Architecture . . . 66
5.3.4 Trailer State Estimation . . . 68
5.4 Sim-to-Real Gap Analysis . . . 69
5.5 Infrastructure Sensor Nodes . . . 71
5.6 Deployment Results . . . 73
5.6.1 Trial Protocol . . . 73
5.6.2 Tracking Performance . . . 74
5.6.3 Articulation Limit Cycle . . . 75
5.7 Summary . . . 77

6 Conclusion **78**
6.1 Summary of Findings . . . 78
6.2 Limitations . . . 79
6.3 Future Work . . . 80

Bibliography **81**

APPENDICES **90**

A Classical Controller Implementation **91**
A.1 Common framework . . . 91
A.1.1 Observation layout and shared action mapping . . . 91
A.1.2 Forward and reverse tracking . . . 92
A.1.3 Vehicle parameters . . . 92
A.2 Pure Pursuit . . . 93
A.3 PID . . . 93
A.4 LQR . . . 94
A.5 Kinematic LTV MPC . . . 95
A.5.1 Prediction models . . . 95
A.5.2 Condensed quadratic program . . . 96
A.6 Gain-tuning methodology . . . 97
A.7 Evaluation harness . . . 98

B Onboard Hitch Angle Estimation **99**

C High-Fidelity Geographic Simulation **101**
C.1 Articulated Vehicle Model . . . 103

List of Figures

3.1 Tractor-trailer geometry and articulation angle 14
3.2 The laboratory's full-size electric shunt truck 17
3.3 The simulated scene behind the BEV observation 19

4.1 Architecture of the learning system 27
4.2 Reverse completion by reward formulation 36
4.3 Observation-space ablation learning curves 42
4.4 Geometry-distillation training 45
4.5 Geometry-encoder policy completion, forward lane-following 49
4.6 Reverse obstacle outcomes against dynamic difficulty 54

5.1 The physical articulated platform 64
5.2 Deployment software stack 66
5.3 RL bridge dataflow 68
5.4 The deployment running in RViz 68
5.5 Tractor tracks for the four deployment trials 75
5.6 Deployment trials against distance traveled 76

C.1 Blender reconstruction of the target site 102
C.2 The `electrans` tractor spawned at a loading dock within the world. 103
C.3 The `electrans` tractor-trailer model 104

List of Tables

3.1 Vehicle parameters used in simulation 17
3.2 Environment-stepping throughput, batched GPU against scalar 24
3.3 NumPy against CuPy throughput by batch size 24

4.1 Training configuration for the algorithm and reward studies 29
4.2 Reward variants compared 34
4.3 Forward lane-following, algorithm and reward comparison 35
4.4 Reverse lane-following, algorithm and reward comparison 35
4.5 Reverse lane-following, TD3 reward detail 37
4.6 Observation space components 40
4.7 Observation-space ablation on lane-following 41
4.8 The vision-training ladder on forward lane-following 46
4.9 State recoverability from the BEV image 48
4.10 Forward obstacle outcomes by layout difficulty 52
4.11 Collision geometry by direction 53
4.12 Reverse obstacle outcomes by dynamic difficulty 54
4.13 Reverse dynamic-difficulty model across three seeds 55
4.14 Forward crash rate, static against dynamic difficulty 55
4.15 Recoverability of the avoidance-path state 56
4.16 Closed-loop vision pipeline 56
4.17 Reverse obstacle maneuvering with and without the safety gate 58
4.18 Forward lane-following controller benchmark 59
4.19 Reverse lane-following controller benchmark 59
4.20 Reference-path information available to each method 60

5.1 Lane following on the physical platform 74

A.1 Shunt-truck parameters used by the classical controllers. 93

List of Abbreviations

BEV	Bird's-Eye View
CAN	Controller Area Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CTE	Cross-Track Error
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
GNSS	Global Navigation Satellite System
GPU	Graphics Processing Unit
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
LQR	Linear-Quadratic Regulator
MASc	Master of Applied Science
MDP	Markov Decision Process
MPC	Model Predictive Control
MVSL	Mechatronics Vehicle Systems Lab
NDT	Normal Distributions Transform
NMPC	Nonlinear Model Predictive Control
PID	Proportional-Integral-Derivative
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
ROS	Robot Operating System
RTK	Real-Time Kinematic
SAC	Soft Actor-Critic
SDF	Signed-Distance Field
TD3	Twin Delayed Deep Deterministic Policy Gradient

List of Symbols

δ	Tractor steering angle
$\dot{\delta}$	Steering rate (control action)
γ	Hitch articulation angle
ψ	Tractor heading
ψ_t	Trailer heading
$\dot{\psi}$	Tractor yaw rate
$e_y,\ e_\psi$	Tractor cross-track and heading error
$e_{y,t},\ e_{\psi,t}$	Trailer cross-track and heading error
$\kappa_1,\ \kappa_2$	Near and far path-curvature samples
L_h	Signed hitch offset from the tractor rear axle
L_t	Trailer wheelbase (hitch to trailer axle)
$l_f,\ l_r$	Tractor front and rear axle distance from the center of gravity
m	Tractor mass
I_z	Tractor yaw moment of inertia
$C_f,\ C_r$	Front and rear cornering stiffness
v	Longitudinal speed
σ	Stop signal (control action)
d	Geometric obstacle-difficulty rating
γ_d	Reinforcement-learning discount factor
α_k	Guided-reward annealing coefficient
Δt	Simulation timestep

Chapter 1

Introduction

1.1 Motivation

The growth of autonomous logistics and large-scale goods transportation has made articulated tractor-trailer vehicles a critical target for autonomous control research. Moving from rigid-body navigation to an articulated system is a significant escalation in mechanical complexity, perception requirements, and control instability, particularly during the low-speed reverse maneuvering that dominates the confined distribution-yard and loading-dock operations this thesis targets.

What distinguishes a tractor-trailer from a single-body vehicle is the revolute degree of freedom at the hitch. In reverse motion the articulation angle γ between the tractor and trailer is open-loop unstable, so without precise closed-loop control any perturbation grows unbounded until the vehicle reaches the jackknife condition, from which no steering correction can recover the trailer's alignment. Classical approaches such as nonlinear model predictive control manage this instability by optimizing over a prediction horizon [2], at a computational cost and a dependence on accurate system models that limit their applicability in unstructured or sensor-rich environments.

Reinforcement learning offers a different approach to articulated-vehicle control by learning a policy through repeated interaction with the system rather than prescribing the control law directly. This shifts much of the controller optimization to training and allows the resulting policy to map complex observations directly to actuation at fixed computational cost. These properties make reinforcement learning a promising candidate for combining articulated-vehicle control with increasingly rich sensor observations.

The challenge of applying reinforcement learning in this setting is twofold. First, the kinematic and dynamic complexity of tractor-trailer systems makes it difficult to formulate learning objectives that simultaneously promote path tracking and hitch stability. Second,

real-world deployment requires perception of lane boundaries and obstacles directly from high-dimensional sensor data rather than pre-computed symbolic state representations. This thesis addresses both challenges in simulation and then evaluates whether the resulting policies transfer to a physical articulated vehicle.

1.2 Thesis Contributions

This thesis builds on two pieces of the author's prior graduate coursework, which established its modeling and learning foundations. The first is a Deep Deterministic Policy Gradient (DDPG) [3] path-following study for a single-body vehicle in a custom Gymnasium [4] simulator, carried out jointly with R. Tymkow [1]. The second is a single-author linear-quadratic-regulator design for the linearized tractor-trailer model [5]. The present work combines and substantially extends these along three axes, from a single rigid body to an articulated tractor-trailer, from a hand-engineered state vector to high-dimensional perception, and adding the physical hardware deployment that neither prior report attempted.

Building on that foundation, this thesis makes the following contributions:

- a combined dynamic tractor and kinematic trailer model, parameterized to a shunt truck, from which the reverse articulation instability is derived, together with the custom Gymnasium simulation of that vehicle and a GPU-batched reimplementation validated against it for numerical parity (chapter 3);
- controlled ablations that isolate the learning algorithm, the reward formulation, and the observation modality, establishing Twin Delayed Deep Deterministic Policy Gradient (TD3) [6] under a multiplicative tracking-and-articulation reward as the combination that achieves reliable completion of the unstable reverse task (chapter 4);
- the identification of raw-image policy training as an optimization pathology of off-policy actor-critic learning rather than a shortage of perceptual information, and its resolution by distilling the bird's-eye-view (BEV) image into the vehicle's path geometry with a frozen supervised encoder (chapter 4);
- an obstacle-navigation policy that follows an avoidance path inferred from the BEV image and decides when a layout is impassable, benchmarked alongside the lane-following policies against tuned pure-pursuit [7], PID, and MPC [8] baselines on matched articulated tasks (chapter 4);
- deployment of the trained forward and reverse policies directly on a physical articulated platform (an AgileX Hunter SE) through a stripped-down Autoware stack, the "RL

bridge," which reproduces on the vehicle the observation and action interfaces the policies were trained against (chapter 5).

1.3 Outline

Chapter 2 reviews the classical and learning-based approaches to articulated control, together with the perception, sim-to-real, and safety work that bounds any physical deployment, and identifies the gap this thesis fills. Chapter 3 derives the tractor-trailer model and the reverse instability it produces, parameterizes it to a shunt truck, and describes the simulation environment and the batched implementation that makes the experimental campaign feasible. Chapter 4 develops the learned controller alongside the evidence for each design choice it makes, covering the selection of the algorithm and the reward, the observation ablation, the training methodology that makes high-dimensional observations usable, obstacle navigation, and the comparison against tuned classical baselines. Chapter 5 carries the trained policies onto the physical platform, describing the deployment stack, the sim-to-real gaps encountered, and the lane-following results measured on hardware. Chapter 6 draws the findings together and sets out the work that remains.

Chapter 2

Literature Review

Controlling an autonomous tractor-trailer system can be done through two distinct methods, classical model or optimization based or learning-based planning and control. This chapter reviews both, with the addition of sim-to-real transfer and safety considerations that bound any physical deployment. This review begins with the control problem that makes articulated vehicles distinct, surveys the classical methods that have traditionally addressed it and the learning-based methods that now challenge them, examines how perception, the simulation-to-reality gap, and the governing safety requirements shape a control policy, and closes by identifying the specific gap that this thesis fills.

2.1 The Articulated-Vehicle Control Problem

Autonomous driving is conventionally decomposed into a pipeline of perception, planning, and control, with each stage feeding the next [9]. Within that pipeline the tractor-trailer is a demanding case for the control stage, because of the passive hitch that couples its two bodies.

Prior analyses have established why. The hitch adds an internal configuration variable, the articulation angle, whose reverse dynamics are open-loop unstable. In forward motion a small hitch disturbance decays as the tractor pulls the trailer back into alignment, but in reverse the sign of the longitudinal velocity flips this behavior. A small articulation error grows rather than decays and, without corrective steering, runs away to the jackknife condition, a folded configuration from which no steering input can recover the trailer [10]. Reverse driving is additionally non-minimum-phase, in that to move the trailer to one side the tractor must first steer to the other [11]. These two properties, open-loop instability and non-minimum-phase response, are what separate articulated control from ordinary rigid-body lane keeping. The kinematic and dynamic model that produces them is derived in

chapter 3, and the confined low-speed setting that fixes the capabilities a controller must deliver is set out in chapter 1.

2.2 Classical Model-Based Control

The longest-established response to articulated instability is model-based control, deriving a dynamic model, then designing a control law that provably stabilizes it. This section reviews the two layers of that approach, the feedback controllers that stabilize and track, and the motion planners that supply the reference for them to track. Together they form the conventional pipeline against which the learned controller of this thesis is benchmarked.

Feedback Control and Stabilization

At the simplest level, a proportional-integral-derivative (PID) controller is trivial to implement and computationally negligible, but it acts only on the present error and cannot anticipate the jackknife condition. Its gains must also be re-tuned for different payloads and path distributions. A linear state-feedback law derived from the linearized tractor-trailer model addresses the multi-variable coupling more directly. Choosing that feedback as a linear-quadratic regulator (LQR) trades tracking accuracy against control effort through a quadratic cost and is optimal for the linearized plant [12]. The author's prior work applied exactly this design to the linearized tractor model [5].

Model predictive control (MPC) generalizes linear feedback by re-optimizing the steering sequence over a receding horizon subject to explicit state and input constraints [8]. This constraint handling is valuable for articulated vehicles because a hard bound can be placed directly on the articulation angle. Linear MPC suffices near an operating point, whereas nonlinear MPC (NMPC) retains the full nonlinear kinematics, including the reverse instability, at a higher computational cost [2]. Control tailored specifically to the articulated case has taken two further forms. Active trailer steering adds an actuated degree of freedom at the trailer axle or hitch to damp articulation directly, and has been shown to improve both the high-speed stability and the low-speed maneuverability of tractor-semitrailers [13, 14]. For a conventional trailer with no such actuation, all authority rests on the tractor's steering angle, and recent work has used an intrinsically stable MPC formulation to suppress jackknifing while tracking a reference, exploiting stable inversion of the non-minimum-phase dynamics [15].

Geometric and Nonlinear Control

A parallel line of work exploits the geometric structure of the non-holonomic system directly. Trailer systems are a classic example of a differentially flat system, meaning that the full state and the control inputs can be written algebraically in terms of a flat output and its derivatives, which reduces trajectory generation to the problem of shaping that output [16]. Converting the kinematics to chained form enables time-varying feedback laws with provable path-following and point-stabilization guarantees for wheeled vehicles [17]. For the specific and difficult case of reverse motion, feedback schemes have been derived that stabilize a truck and trailer backing along a path in spite of the open-loop instability [11]. These methods achieve strong guarantees by committing fully to an analytical model of the vehicle, which is precisely the commitment that the learned controller of this thesis seeks to avoid.

Motion Planning for Articulated Vehicles

The feedback controllers above track a reference path, which must first be planned. For articulated vehicles this planning problem is itself difficult, because a feasible path must respect the vehicle's non-holonomic constraints and avoid configurations from which the trailer cannot recover. Sampling-based planners such as the rapidly-exploring random tree and its asymptotically optimal variant RRT* explore the configuration space without an explicit discretization [18, 19], whereas search-based freespace planners such as hybrid A* discretize the search yet retain kinematic feasibility [20]. Tailored to the articulated case, lattice-based planners assemble precomputed motion primitives that connect circular-equilibrium configurations, enabling complex forward and reverse maneuvers [21], and optimization-based formulations compute reference paths that minimize the swept area of the vehicle bodies subject to road and obstacle constraints [22]. Once a path exists, geometric trackers such as pure pursuit follow it by steering toward a look-ahead point on the path [7], and refined trackers such as the Stanley controller add an explicit cross-track error term with demonstrated performance on full-scale autonomous vehicles [23].

Strengths and Limitations

The appeal of this pipeline is that it comes with guarantees. Given an accurate model, the stability of the controller [2, 17] and the feasibility of the planned path [18, 19] can be analyzed and, in principle, guaranteed. Its cost is precisely that dependence. The model must be identified and kept accurate across payloads and operating conditions, the controllers and planners must be tuned, and the system must be assembled from separately designed planning and control stages. These costs motivate the alternative pursued in this thesis, a

controller learned directly from interaction, which maps observations to actions without an explicit model inversion or a separately planned reference trajectory.

2.3 Learning-Based Control

Where a classical controller encodes the vehicle model explicitly and executes a fixed control law, a reinforcement learning (RL) agent infers a control policy directly from interaction with its environment, optimizing the policy end-to-end against a scalar reward [24]. This trades the analytical guarantees of the model-based approach for the ability to learn behaviors that are difficult to hand-design and to fold perception and control into a single learned mapping. This section traces the RL methods relevant to the thesis from their general foundations to their specific application to articulated vehicles.

Deep Reinforcement Learning Foundations

Modern RL for control rests on the deep-network function approximators that made high-dimensional value estimation practical. The Deep Q-Network first showed that a single architecture could learn to act directly from high-dimensional visual input across a range of tasks, using experience replay and a target network to stabilize learning [25]. Value-based methods of this kind are naturally suited to discrete action sets. The continuous steering and speed commands of a vehicle instead call for policy-based methods.

Actor-Critic and Policy-Gradient Methods

The deterministic policy gradient provides a route to continuous control by learning a deterministic actor alongside a critic that estimates action value [26]. Deep Deterministic Policy Gradient (DDPG) is the deep-network realization of this idea and a canonical off-policy actor-critic algorithm for continuous action spaces [3], but it is prone to critic overestimation bias and unstable updates. Two lines of work address this. Proximal Policy Optimization (PPO) is an on-policy method that improves stability through a clipped objective, at the cost of lower sample efficiency, which matters when simulation rollouts are expensive [27]. Soft Actor-Critic (SAC) instead augments the reward with an entropy term to encourage exploration and improve robustness in continuous control [28]. This thesis adopts Twin Delayed DDPG (TD3), a direct successor to DDPG that targets its two weaknesses through twin critics, delayed actor updates, and target-policy smoothing [6], and whose role is detailed in chapter 4. The author's prior coursework used DDPG for single-body path following [1]. The move to TD3 here is a direct response to the training instability observed in that work.

Reinforcement Learning for Autonomous Driving

RL has been applied across the driving stack, and recent surveys catalog its use for lane keeping, behavioral decision making, and end-to-end control [29]. Two threads are relevant here. The first is end-to-end learning, in which a single network maps sensor input to control, whether trained by imitation of a human driver [30] or by reinforcement. A notable demonstration of this trained a lane-following policy on a full-scale car with on-board RL in a single day [31]. The second is the explicit framing of the driving task as a Markov decision process with a shaped reward, as set out in early RL-for-driving frameworks [32]. Both threads inform the observation and reward design developed in this thesis.

Reinforcement Learning for Articulated Vehicles

Applying RL specifically to articulated vehicles is more recent but growing, and the earliest instance long predates deep RL. The neural truck-backer-upper of Nguyen and Widrow learned to reverse a trailer toward a target, and depended on training that exposed the network to the unstable configurations it had to correct [33]. Modern work has revisited the problem with deep methods. A DDPG backing controller paired with a fuzzy-logic supervisor learned to reverse while avoiding the jackknife state [34], and policy-gradient agents have been trained to track reference paths with a tractor-trailer using rewards that penalize both tracking deviation and articulation growth [35]. A recurring theme in this small body of work, and one this thesis examines directly through its reward and observation ablations, is the importance of the hitch-stability term. The reward must discourage articulation growth for reverse driving to be learned reliably. Reward shaping of this kind rests on a firm theoretical footing, as potential-based shaping can accelerate learning while provably preserving the optimal policy [36].

Sequence Models and Curriculum Learning

Beyond actor-critic control, Decision Transformers recast RL as sequence modeling, predicting an action from a context window of past states, actions, and returns-to-go [37]. Such models can represent longer temporal dependencies than a shallow actor-critic policy, but their performance depends heavily on the diversity of the offline dataset, and a dataset dominated by nominal tracking behavior tends to give poor recovery from the high-error states it underrepresents, an instance of the distribution-shift problem that motivates deliberately collecting such states. This need for coverage of hard configurations reappears as a training-data problem regardless of the policy class, and it motivates curriculum learning: presenting tasks in a deliberately ordered sequence, or automatically emphasizing the situations the

agent handles worst [38, 39]. This thesis retains an actor-critic formulation but adopts a failure-replay curriculum (chapter 4) that revisits failed scenarios, an idea related in spirit to prioritized experience replay [40].

2.4 Perception and Representation Learning

A learned controller is only as good as what it perceives. A central question of this thesis is whether a compact, hand-engineered state vector or a learned representation of a bird's-eye-view (BEV) image yields better and more transferable control. This section reviews the perception building blocks that make the latter possible.

Convolutional Networks for Driving

Convolutional neural networks (CNNs) are the standard front-end for visual perception, learning spatial features directly from images rather than from hand-specified descriptors [41, 42]. Network depth was later shown to be trainable at scale through residual connections, which underpin most modern vision backbones [43]. In driving, CNNs have been trained end-to-end to map camera images directly to steering commands [44]. The bird's-eye view is a particularly convenient representation for ground-vehicle control because it places the vehicle and its surroundings in a top-down frame. Recent perception systems learn to lift camera images into exactly such a representation [45], and the classical occupancy grid is its long-standing hand-built analogue [46]. In this thesis a CNN processes a rendered BEV image to extract lane geometry and obstacle layout.

Representation Learning and Autoencoders

Training a control policy directly on raw images is sample-inefficient, because the policy must learn to see and to act at the same time. Autoencoders decouple these by compressing images into a compact latent space in an unsupervised pre-training stage [47], an idea applied to RL to shrink the policy's input and accelerate learning [48]. Variational autoencoders impose a probabilistic structure on the latent space that can improve its regularity [49], and spatial autoencoders tailored to control learn latent features corresponding to task-relevant image locations [50]. The U-Net architecture, originally developed for image segmentation, preserves multi-scale spatial detail through skip connections and provides a stronger structural prior than a plain convolutional encoder [51]. This thesis treats these learned encoders as one end of a representation spectrum, with the engineered state vector at the other, and compares them directly in chapter 4.

Range Sensing and Point-Cloud Perception

Bird's-eye imagery is not the only high-dimensional modality available to a ground vehicle. Planar LiDAR provides a direct geometric measurement of the surrounding free space, which the classical occupancy grid represents as a discretized map of drivable space [46]. At the other extreme of complexity, deep networks such as PointNet learn features directly from raw, unordered three-dimensional point clouds [52]. This thesis deliberately adopts a lighter-weight LiDAR representation that sits between these poles, a fixed-length vector of range returns appended to the state vector, rather than a learned point-cloud encoder. This keeps the observation compact and the sensing model simple, and it allows the LiDAR and image modalities to be compared on an equal footing in chapter 4.

2.5 Sim-to-Real Transfer

Because the controllers in this thesis are trained entirely in simulation, transferring a simulation-trained policy to physical hardware is a first-class concern rather than an afterthought. The central obstacle is the reality gap, the systematic mismatch between simulation and hardware that degrades a naively transferred policy [53]. It is useful to separate this gap into a visual component, a difference in what the sensors report, and a dynamics component, a difference in how the vehicle responds to a command, because the two are attacked by different tools.

The dominant learning-based remedy is domain randomization, which randomizes parameters during training so that the real system appears to the policy as one more variation of a distribution it has already learned to handle. It has been applied to the visual gap by randomizing rendering [54] and to the dynamics gap by randomizing physical parameters such as mass and friction [55]. Randomization can also be closed into a loop. System-identification methods adapt the simulator's parameter distribution from a small amount of real experience, so that simulated and real behavior are matched rather than merely broadened [56], while domain-adaptation methods instead learn a mapping that reconciles simulated and real observations [57]. A complementary strategy is to raise simulator fidelity directly. The open urban driving simulator CARLA is a widely used example [58], and chapter C describes a geographically faithful simulator built for this work. Actuation delay is a particularly damaging instance of the dynamics gap for articulated reversing, and is addressed both by the delay-aware learning methods discussed above and by the Smith-predictor compensator. That chapter characterizes the specific gaps encountered on the physical platform, namely system latency and actuation delay, hitch-angle sensing noise, and path-distribution mismatch, and addresses them through targeted engineering compensation while noting domain

randomization as a route to further robustness.

2.6 Safety and Standards

A controller intended for a physical vehicle must ultimately answer to safety requirements, and the learning-based setting raises safety questions of its own. Safe reinforcement learning studies how to learn while respecting constraints or avoiding catastrophic states [59], and control barrier functions offer a complementary, model-based mechanism to overlay hard safety constraints on a learned policy, a direction identified here as future work [60]. For an articulated vehicle the binding safety concern is the jackknife condition, which the reward must discourage and the operational limits of chapter 5 enforce. A fielded deployment would in addition have to sit within the established automotive-safety framework, most directly the Safety of the Intended Functionality standard for hazards that arise under nominal operation [61], alongside the broader functional-safety, safety-case, and automation-level standards [62–64]. These requirements bound the present simulation study rather than being evaluated within it, and they frame the deployment reported in chapter 5.

2.7 Summary

This chapter reviewed the two traditions that bear on articulated control, together with the perception, sim-to-real, and safety considerations that bound any physical deployment. Classical model-based control offers guarantees but demands an accurate model, careful tuning, and a separate planning stage. Learning-based control removes those demands and must instead contend with reward design, the reverse instability, high-dimensional perception, and the reality gap.

The gap this thesis fills lies between them. Prior learned controllers for articulated vehicles have been demonstrated almost exclusively in simulation and on a single, hand-specified observation modality. The backing controllers of Nguyen and Widrow [33] and Bejar and Morán [34], and the tracking agent of Kang et al. [35], each learn from a fixed state representation and report results in simulation, without a physical deployment or a comparison across perception modalities. This thesis develops a learned articulated-vehicle controller with high-dimensional perception, compares observation modalities and reward formulations through controlled ablations, benchmarks the result against tuned classical baselines, and deploys the trained policies on physical hardware with explicit attention to hitch stability and sim-to-real transfer. The vehicle model and simulation environment that support this program are developed in chapter 3.

Chapter 3

Modeling, Simulation, and Problem Formulation

3.1 Tractor Dynamics

The tractor is represented by a planar single-track bicycle model [65], in which the front axle is lumped into one steerable wheel and the rear axle into one fixed wheel. Its global position (x, y) follows from the body-frame longitudinal and lateral velocities (v_x, v_y) and the heading ψ:

$$\dot{x} = v_x \cos\psi - v_y \sin\psi \tag{3.1}$$

$$\dot{y} = v_x \sin\psi + v_y \cos\psi \tag{3.2}$$

Lateral force is generated at each tire as a function of its slip angle. Under the small-angle assumptions appropriate to the operating envelope, the front and rear slip angles are

$$\alpha_f = \delta - \frac{v_y + l_f\dot{\psi}}{v_x}, \qquad \alpha_r = -\frac{v_y - l_r\dot{\psi}}{v_x}, \tag{3.3}$$

where δ is the steering angle. Rather than evaluating the full Pacejka magic-formula tire curve, the simulator operates in its linear region. For the modest slip angles encountered here the curve is well approximated by a constant cornering stiffness, giving the lateral tire forces

$$F_{yf} = C_f\,\alpha_f, \qquad\qquad F_{yr} = C_r\,\alpha_r. \tag{3.4}$$

This linearized-Pacejka (constant cornering-stiffness) model is the one implemented in the simulator. It preserves the lateral behavior relevant to path tracking while avoiding the hard-to-identify parameters of the full nonlinear tire curve. Collecting the longitudinal, lateral, and yaw-moment balances, with a quadratic aerodynamic drag $F_{\text{drag}} = \frac{1}{2}C_d\rho A v_x^2$ and a net drive force F_x, yields the tractor dynamics. Although the drag force is negligible at low speeds, it was added as a damping force so that the simulation eventually came to rest under no input. Active braking is achieved through applying a negative drive force.

$$m\left(\dot{v}_x - v_y\dot{\psi}\right) = F_x - F_{\text{drag}} - F_{yf}\sin\delta, \tag{3.5}$$

$$m\left(\dot{v}_y + v_x\dot{\psi}\right) = F_{yf}\cos\delta + F_{yr}, \tag{3.6}$$

$$I_z\,\ddot{\psi} = l_f F_{yf}\cos\delta - l_r F_{yr}. \tag{3.7}$$

For control design and for the time-stepping used in simulation, these equations are linearized about straight-line motion ($\delta \approx 0$) at a constant reference speed $v_x = v$. With state $\boldsymbol{\xi} = [\,v_x,\ v_y,\ \dot{\psi}\,]^\top$ and input $\mathbf{u} = [\,F_x,\ \delta\,]^\top$, the linearized dynamics take the time-invariant form $\dot{\boldsymbol{\xi}} = A\boldsymbol{\xi} + B\mathbf{u}$ with

$$A = \begin{bmatrix} -\dfrac{C_d\rho A v}{m} & 0 & 0 \\ 0 & -\dfrac{C_f + C_r}{mv} & \dfrac{l_r C_r - l_f C_f}{mv} \\ 0 & \dfrac{l_r C_r - l_f C_f}{I_z v} & -\dfrac{l_f^2 C_f + l_r^2 C_r}{I_z v} \end{bmatrix}, \qquad B = \begin{bmatrix} \dfrac{1}{m} & 0 \\ 0 & \dfrac{C_f}{m} \\ 0 & \dfrac{l_f C_f}{I_z} \end{bmatrix}. \tag{3.8}$$

The continuous system is converted to the discrete update $\boldsymbol{\xi}_{k+1} = A_d\boldsymbol{\xi}_k + B_d\mathbf{u}_k$ by a zero-order-hold discretization at the simulation step $\Delta t = 0.1\,\text{s}$, and it is this discrete map that is advanced at each timestep.

This level of fidelity is sufficient for the control tasks studied in this thesis, where the quantities of interest are path-relative position error, heading error, steering angle, and hitch articulation. The aim is a physically meaningful environment for training and evaluating classical and trained policies, not a high-fidelity vehicle simulator. An overly simplified kinematic model of the tractor would make the simulation task unrealistically easy and could reduce transferability to physical systems by neglecting actuator response, steering-

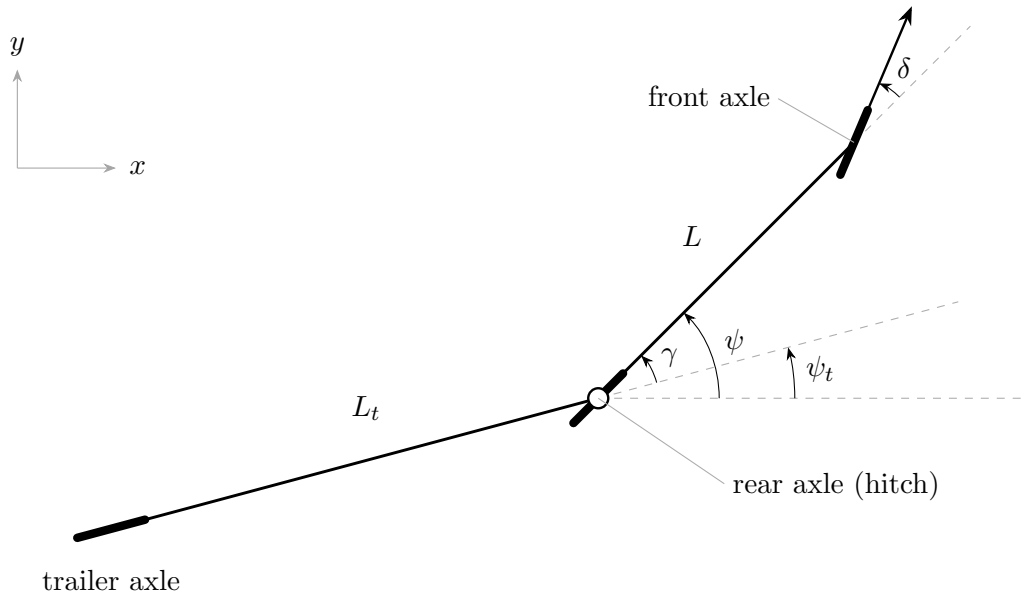


Figure 3.1: Tractor-trailer geometry. The tractor rear axle carries the hitch ($L_h \approx 0$), from which the trailer of wheelbase L_t trails at articulation angle $\gamma = \psi - \psi_t$. The front wheel is steered by δ relative to the tractor heading ψ, and L is the tractor wheelbase.

rate limits, and tire-force effects on the vehicle's motion. In contrast, a kinematic model was chosen for the trailer because, at the low speeds considered in this thesis, the trailer's dominant behavior is governed by its rolling constraint rather than by lateral tire-slip or inertial dynamics. The full-length trailer therefore primarily contributes geometric articulation through the hitch. Its axle constrains the trailer to roll along its longitudinal direction, and this nonholonomic constraint determines the evolution of the hitch angle during forward and reverse maneuvers. This reduced-order representation preserves the main source of reverse instability while avoiding the additional tire, load-transfer, and yaw-inertia parameters required by a full dynamic trailer model.

3.2 Trailer Articulation and Reverse Instability

Figure 3.1 illustrates the tractor-trailer geometry, the tractor and trailer headings ψ and ψ_t, and the articulation angle $\gamma = \psi - \psi_t$ used throughout this section. The hitch is drawn at the rear axle because the reference tractor places it just 0.25 m ahead of that axle, under a tenth of the 2.95 m tractor wheelbase, and the model retains the offset L_h as a free parameter rather than assuming it away.

The articulated system adds a trailer body coupled to the tractor through a revolute hitch. The relevant geometry is:

- L_h: signed distance from the tractor rear axle to the hitch point (in both the training model and the hardware platform the hitch sits at the rear axle, so $L_h \approx 0$);
- L_t: distance from the hitch to the trailer axle, i.e. the trailer wheelbase;
- $\gamma = \psi - \psi_t$: the articulation angle between the tractor heading ψ and the trailer heading ψ_t (the same convention used by the deployment in chapter 5).

The hitch point is carried rigidly by the tractor,

$$x_h = x - L_h \cos\psi, \qquad y_h = y - L_h \sin\psi, \tag{3.9}$$

and translates with the tractor's longitudinal speed v. Imposing a pure-rolling (no lateral slip) constraint on the trailer axle fixes the trailer yaw rate from the lateral component of the hitch velocity:

$$\dot{\psi}_t = \frac{v}{L_t}\sin(\psi - \psi_t) = \frac{v}{L_t}\sin\gamma. \tag{3.10}$$

The articulation angle therefore evolves as

$$\dot{\gamma} = \dot{\psi} - \frac{v}{L_t}\sin\gamma, \tag{3.11}$$

where the tractor yaw rate $\dot{\psi}$ is supplied by the tractor model of eq. (3.8) (equivalently $\dot{\psi} = (v/L)\tan\delta$ for a purely kinematic tractor of wheelbase $L = l_f + l_r$). The trailer axle pose then follows from the hitch point and the trailer heading,

$$x_t = x_h - L_t \cos\psi_t, \qquad y_t = y_h - L_t \sin\psi_t. \tag{3.12}$$

The sign of v in eq. (3.11) is what makes reverse motion qualitatively harder. Linearizing about the aligned configuration ($\gamma \approx 0$, so $\sin\gamma \approx \gamma$) and holding the steering-driven tractor yaw fixed, the unforced articulation mode obeys

$$\dot{\gamma} \approx -\frac{v}{L_t}\gamma, \qquad \lambda = -\frac{v}{L_t}. \tag{3.13}$$

In forward motion ($v > 0$) the eigenvalue λ is negative. Articulation errors decay and the trailer self-aligns behind the tractor. In reverse ($v < 0$) the eigenvalue becomes positive and the same errors grow exponentially toward the jackknife condition. This is an open-loop instability rather than a mere tuning difficulty, and it is the defining feature of reverse articulated maneuvering, the reason the reverse task demands active hitch stabiliza-

tion, trailer-referenced error metrics, and the dedicated reward and curriculum treatments developed in chapter 4.

The jackknife toward which this instability drives is a folded configuration from which no steering input can recover the trailer's alignment, so successful reverse maneuvering demands anticipatory control that corrects a developing articulation error before it grows large enough to become unrecoverable. In practice the controller must regulate three coupled objectives at once (trailer path tracking, tractor placement, and articulation stability), and a controller that attends only to the tractor's cross-track error can look reasonable in the short term while driving the hitch toward instability.

3.3 Vehicle Parameters

The vehicle model is parameterized to represent a shunt truck (terminal tractor, or yard hostler), the class of vehicle that performs the confined low-speed trailer maneuvering in distribution centers that this thesis targets. The laboratory operates a full-size electric shunt truck of this class, built for the Electrans project and shown in fig. 3.2. It is a one-off prototype whose mass distribution, yaw inertia, and tire properties are not characterized to the standard of a production specification sheet, so the model is instead parameterized from a documented vehicle of the same class. The specific reference vehicle is the Kalmar Ottawa T2 4x2 terminal tractor. Its layout maps directly onto the single-track bicycle model of eq. (3.8), and its manufacturer specification sheet provides genuine primary-source data [66].



Figure 3.2: The laboratory's full-size electric shunt truck, the class of vehicle the simulation model represents, shown bobtail as in table 3.1. The parameter values are taken from a production terminal tractor of the same class rather than from this prototype.

The tractor has a wheelbase of 2.95 m, a maximum steering angle of 50° (0.87 rad), and is governed to a top speed near 20 m/s, more than enough to operate in the low-speed regime studied here. Table 3.1 lists the parameter set used in simulation, corresponding to the bobtail (no-trailer) diesel configuration.

Table 3.1: Vehicle parameters used in simulation, for the bobtail Kalmar Ottawa T2 diesel terminal tractor.

Parameter	Value	Parameter	Value
m	6577 kg	l_f	1.33 m
I_z	16 000 kg m^2	l_r	1.62 m
C_f	190 000 N/rad	C_d	0.8
C_r	200 000 N/rad	A	7.0 m^2
Δt	0.1 s	ρ	1.225 kg/m^3

The mass, wheelbase, and overall dimensions are taken directly from the manufacturer specification sheet. The yaw moment of inertia I_z is estimated from the mass and body

envelope using a standard moment-of-inertia estimator for road vehicles [67]. The cornering stiffnesses C_f and C_r are derived from published heavy-truck tire data [68], scaled by the number of tires per axle (a single pair at the steered front axle against dual tires at the drive axle) and by the axle loads. The front and rear stiffnesses therefore differ, unlike the equal-stiffness simplification commonly used for passenger-car examples. The longitudinal center-of-gravity split is estimated from an approximate static front/rear axle load distribution, and the aerodynamic terms C_d and A are engineering estimates for a terminal-tractor cab. Neither the aerodynamic terms nor the center-of-gravity split is backed by primary data, but their influence is minor in the operating regime of interest. Aerodynamic forces are negligible at governed yard speeds, and the center-of-gravity split mainly shapes high-speed transient response rather than the low-speed trajectories studied here. The timestep $\Delta t = 0.1\,\mathrm{s}$ corresponds to the 10 Hz control loop used in the hardware deployment (chapter 5), so the simulation and the deployed controller operate at a matched rate.

The values in table 3.1 describe the bobtail tractor, which defines the dynamic tractor subsystem used in simulation. The trailer is represented kinematically through the hitch model of eq. (3.11), for the reasons given in section 3.1. Its inertia is deliberately omitted rather than approximated. Because the tasks prescribe longitudinal speed, trailer mass never enters the control loop through the longitudinal balance, and both the rolling constraint of eq. (3.10) and the reverse-instability eigenvalue of eq. (3.13) are independent of mass and moment of inertia. Trailer inertia is therefore not a control-critical state in the nominal environment.

3.4 Simulation Environment

The training and evaluation environment is a custom 2D simulation built on the Farama Foundation Gymnasium library [4]. Gymnasium provides a standardized API for environments to integrate directly with reinforcement learning algorithm libraries, including Stable Baselines3 [69].

The simulation is implemented in Python, with Pygame [70] providing collision masks and rendering. It was built around a rendering pipeline that draws the tractor-trailer and lane geometry as a top-down frame. This representation can be viewed live for monitoring, or passed directly to the image encoder as a bird's-eye-view (BEV) observation, making high-dimensional perception available to the agent without a separate image-synthesis step. At each timestep, the discrete-time tractor-trailer state-space equations are re-evaluated from the previous state and control inputs, updating the vehicle position, heading, hitch angle, and associated error signals, and Pygame redraws the scene as a 2D top-down view from which

the BEV image observation is cropped. For large-scale training this rendered observation is later produced without the Pygame renderer by the batched GPU simulator (section 3.6), but the rendering pipeline remains the conceptual origin of the BEV observation and the reference for what it encodes.

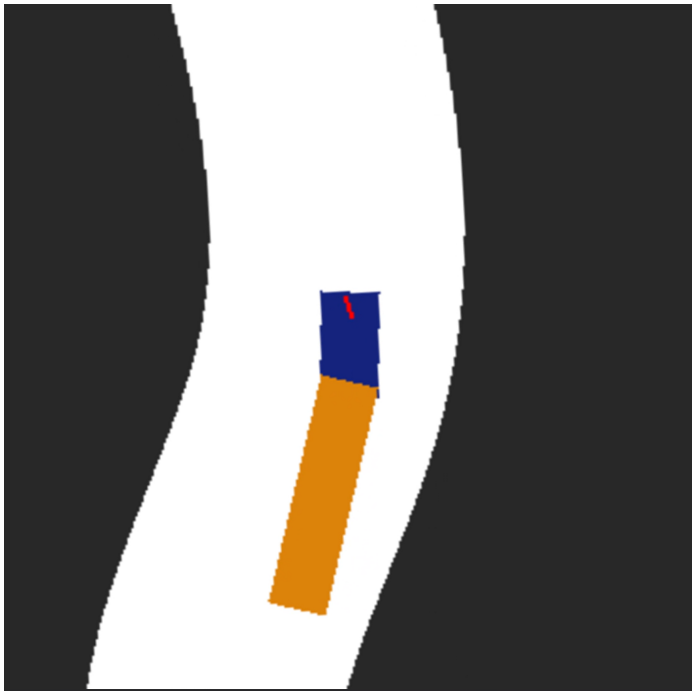


Figure 3.3: The simulated scene from which the bird's-eye-view observation is cropped. The drivable lane corridor is white and everything outside it is blocked; the tractor is blue and the trailer orange.

The simulation environment includes:

1. **Vehicle Model**: A combined dynamic tractor (bicycle model with a linearized Pacejka tire model [71]) and kinematic trailer with hitch constraints, using the shunt-truck parameters of section 3.3.

2. **Lane Generation**: Randomly generated lane paths using cubic spline interpolation, producing varying curvature paths for training and evaluation.

3. **Obstacle Placement and Local Re-planning**: Randomly placed static obstacles within the lane and a local path planner that perturbs the centerline around those obstacles for avoidance experiments.

4. **Collision Detection**: Pygame sprite mask-based collision detection, providing pixel-precise contact sensing between the vehicle body and obstacles or lane boundaries.

5. **Observations**: the rendered top-down BEV image described above, alongside engineered state and range-sensor (LiDAR) observations. The full set of observation modalities available to the policy is defined in chapter 4.

3.4.1 Control Policy Interface

A traditional or learning based policy interacts with the simulation through a deliberately narrow interface. In the reinforcement-learning sense its action is two-dimensional, a continuous steering rate $\dot{\delta}$ and a stop signal σ:

$$\mathbf{a} = [\dot{\delta},\, \sigma] \in [-25^\circ/\mathrm{s},\, 25^\circ/\mathrm{s}] \times [-1,\, 1]. \tag{3.14}$$

Longitudinal speed is not itself a control degree of freedom. The environment is driven by a target speed, but the policy does not set it directly. Instead, a thin interface layer maps the stop signal onto that speed, commanding the fixed operating value of the current task (5 m/s forward, -5 m/s in reverse) while the vehicle drives and zero speed the instant a stop is issued. Because speed is prescribed in this way, every method solves the same lateral problem and the forward and reverse tasks share a common control interface. Commanding the steering rate rather than the steering angle reflects the fact that the steering angle cannot change instantaneously on a physical vehicle, so regulating its rate of change is more physically realistic. The stop signal is a go/stop decision. When σ exceeds a fixed threshold the vehicle halts and the episode terminates, which lets the policy decline to attempt a layout it judges impassable. This stop-capable action is used across all learning-based tasks, so that the interface does not change between the no-obstacle and obstacle settings.

Episodes terminate on collision, jackknife ($|\gamma| > 90^\circ$), intentional stop, or successful task completion.

3.5 Environment Tasks

The environment is structured around reusable task and observation components. A base lane-following task is combined with different observation heads, while obstacle-aware variants reuse the same task logic through an environment that adds obstacle generation, local path planning, and obstacle-aware reward evaluation. This avoids maintaining separate forward, reverse, LiDAR, and BEV implementations with duplicated task logic.

3.5.1 Lane Following

The lane-following environment extends the base tractor-trailer environment to present a lane-tracking task. A reference lane is generated at the start of each episode using cubic spline interpolation through randomly placed waypoints. The reward function encourages the agent to keep both the tractor and trailer within the lane boundaries while maintaining progress along the path.

Reverse lane-following environments use the same path generation and rendering pipeline, but reverse the longitudinal command and compute reward and tracking terms with the trailer as the leading unit, so that the same task can be studied in backward maneuvers without changing the task definition.

3.5.2 Obstacle Avoidance

The obstacle avoidance environment adds static obstacles within the lane. The agent must navigate the tractor-trailer through the lane while avoiding obstacles, which requires both path-following behavior and collision avoidance. Instead of following the global lane centerline directly, the environment constructs a local path around the placed obstacles and evaluates tracking error relative to this locally planned path. Obstacles are placed as a single cluster of one or two circular obstacles per episode, drawn from a small set of layout categories (clear, shift, squeeze, and blocked), so that the difficulty distribution is controlled and can be stratified during evaluation. The locally planned avoidance path is used only to shape the reward and is never exposed to the agent, so the detour must be inferred from the LiDAR or BEV observation. In addition to terminal collision penalties, the base environment applies a per-step proximity penalty near blocked occupancy-grid cells, so both lane boundaries and inserted obstacles discourage the agent before contact occurs. Forward and reverse obstacle environments are both available. The scalar difficulty rating that indexes these layouts, and the reward that keys on it, are developed in section 4.8.1.

3.6 Scalable Simulation: GPU-Parallel Batched Environments

The observation, reward, and curriculum studies of chapter 4, together with the controller benchmark of section 4.9, require training and evaluating a large matrix of policies: several observation modalities, multiple reward formulations, forward and reverse variants, and repeated random seeds for statistical validity. The scalar Pygame/Gymnasium environment

of section 3.4 is the physically faithful reference model, but it advances a single environment at a time in Python and rebuilds an occupancy grid on every reset, which makes a study of this size prohibitively slow. To make the empirical campaign tractable, the environment was re-implemented as a batched simulator that advances N independent environments simultaneously on a graphics processing unit (GPU). The original scalar environment is retained unchanged as the parity oracle against which the batched implementation is validated, so that the physical fidelity established in chapter 3 continues to hold for every result reported in this thesis.

3.6.1 Batched Environment Design

The batched environment adds a leading batch axis of size N to every state array, so that a single vectorized operation advances all N environments at once. It is written against a thin array-backend abstraction and runs identically on NumPy for the CPU [72] or CuPy for the GPU [73], selected at run time by a single environment variable. Because CuPy exposes a NumPy-compatible interface, the same vectorized code executes on either device without modification. Achieving this required replacing five constructs in the scalar environment that are inherently sequential or defined one environment at a time:

1. the per-step zero-order-hold discretization, computed in the scalar model with `scipy.signal.cont2discrete`, is replaced by a batched matrix exponential using the scaling-and-squaring method [74], which discretizes all N state-space models in one operation;

2. the per-reset occupancy grid and k-d tree nearest-neighbor lookup are replaced by an analytic distance-to-centerline computation together with an on-device pool of pre-generated paths, which removes the reset bottleneck entirely;

3. the sequential, per-beam LiDAR ray-march is replaced by a vectorized fixed-length march with a first-hit reduction;

4. the Python-loop collision and proximity tests are replaced by a batched gather of vehicle body points against the corridor;

5. the scalar termination, stop-signal, and spawn branches are replaced by boolean masks with internal automatic reset, so that finished environments reset, drawing a new path and setting a new vehicle pose, and restart without host-side control flow.

The batched environment also produces the bird's-eye-view image observation without invoking the Pygame renderer. Because a top-down BEV image is a membership query at

a grid of sample points (whether each point lies inside the lane corridor, on an obstacle, or outside the world), it is generated using the same analytic corridor primitives as the LiDAR march: for each environment an $S \times S$ grid is laid out in the vehicle frame, transformed to world coordinates by the anchor pose, and evaluated at all $N \cdot S \cdot S$ points in a single vectorized gather. The resulting observation is an occupancy image (drivable versus blocked) rather than a rendered grayscale scene, and its cost per environment is comparable to a single LiDAR frame. All three observation modalities (state, LiDAR, and BEV image) therefore run on the GPU, so the entire ablation matrix can be executed on the batched environment.

3.6.2 Numerical Parity

Retaining the scalar environment as an oracle allows the batched implementation to be validated for numerical equivalence rather than merely for plausible behavior. Across a suite of parity tests, the batched dynamics, geometry, observation, and reward pipeline reproduces the scalar model to within floating-point tolerance: the matrix-exponential discretization matches `scipy` to roughly 10^{-8} absolute error, the vehicle dynamics agree to within $\sim 10^{-14}$ over a 200-step rollout, the state and LiDAR observation vector is bit-identical, and the reward matches to $\sim 10^{-14}$ wherever the proximity term is inactive. The NumPy and CuPy backends produce identical results, confirming that the GPU port introduces no numerical divergence of its own.

Two terms are deliberately not bit-identical, because they replace a rasterized grid with analytic geometry: the near-boundary proximity penalty differs by 0.1–0.5 only within the narrow lane-edge region, and the analytic LiDAR march agrees with the grid raycast to within approximately one grid cell, with collision decisions matching the grid on 97–98.5% of randomly sampled states (the remainder lying within one cell of the boundary). These differences are confined to the immediate vicinity of boundaries and do not affect the terminal outcomes that drive reward and evaluation, so the batched environment is a faithful stand-in for the scalar reference in every experiment reported here.

3.6.3 Throughput and Scaling

The batched environment was benchmarked on an NVIDIA RTX 4080 SUPER GPU. Table 3.2 reports raw environment-stepping throughput: at large batch sizes the GPU environment sustains roughly 1.7 million transitions per second for the state observation, against about 2,100 for the single-core scalar environment, and roughly 64,000 transitions per second for the compute-heavier 24-beam LiDAR observation, against about 470. These figures are governed by the batch size and observation type rather than by the vehicle parameters, so

they are unaffected by the specific plant used in a given experiment.

Table 3.2: Environment-stepping throughput (transitions per second) for the batched GPU environment versus the single-core scalar reference, measured on an RTX 4080 SUPER.

Observation	Batched GPU (best N)	Scalar (single core)
State	$\sim$1,700,000	$\sim$2,100
LiDAR (24 beams)	$\sim$64,000	$\sim$470

Because the identical code runs on either backend, the GPU's contribution can be isolated by swapping only the array library. Table 3.3 shows this crossover: at small N the GPU's per-kernel launch overhead loses to CPU NumPy, but as N grows the GPU pulls ahead, reaching $6.6\times$ at $N = 4096$ for the state observation. For the compute-bound LiDAR observation the GPU wins at every batch size, up to $62\times$ at $N = 4096$, because each step performs far more work per environment. This crossover is the expected profile of a batched accelerator and motivates the use of large batch sizes ($N \geq 1024$) together with the LiDAR observation to exploit the device.

Table 3.3: Same batched code, NumPy (CPU) versus CuPy (GPU), by batch size N and observation type. Throughput in transitions per second. Speedup is CuPy relative to NumPy.

Observation	N	NumPy (CPU)	CuPy (GPU)	Speedup
State	256	87,060	30,821	0.35×
State	1024	81,668	122,464	1.5×
State	4096	72,200	479,141	6.6×
LiDAR-24	256	2,676	24,815	9.3×
LiDAR-24	1024	2,334	71,394	30.6×
LiDAR-24	4096	1,005	62,601	62.3×

These gains apply to environment stepping, not to the entire training loop. For off-policy algorithms such as TD3, the gradient-update loop remains the bottleneck: it is a sequence of comparatively small-batch optimizer steps that leaves the GPU underutilized (around 34% utilization during off-policy reverse training), so a single off-policy run is not accelerated in proportion to the environment speedup. The batched environment's value for off-policy learning is therefore twofold: it allows many configuration and seed jobs to share one GPU concurrently, and it makes on-policy algorithms, which consume large environment batches directly and so exploit the fast environment, practical to include in the algorithm

comparison of section 4.4.2. To keep that comparison fair, the batched environment is exposed to Stable Baselines3 [69] through a vectorized-environment adapter, so that the vetted implementations of TD3, PPO, and DQN run on it unchanged. This infrastructure is what makes the multi-seed ablation campaign of chapter 4 feasible within a practical time budget.

3.7 Summary

This chapter fixed the plant, the environment, and the control problem that the rest of the thesis works against. The vehicle is a dynamic single-track tractor coupled to a kinematic trailer through the hitch, parameterized to the Kalmar Ottawa T2 terminal tractor, a level of fidelity that retains steering-rate limits and tire-force effects while leaving out the trailer inertia terms that do not enter the low-speed control loop. From that model comes the property the controller has to answer to, the open-loop instability of the articulation angle in reverse and the jackknife configuration it runs toward. The training environment is a custom Gymnasium simulation that renders the scene top-down, so the bird's-eye-view observation is available without a separate image-synthesis step, and the policy interacts with it through a narrow two-dimensional action of steering rate and stop signal, with longitudinal speed prescribed by the task so that every method solves the same lateral problem. Lane following and obstacle avoidance are built as variants of one task definition in both directions of travel. Running the ablation campaign at the required scale needed a batched GPU reimplementation of that environment, validated against the scalar reference for numerical parity, with the differences confined to the immediate vicinity of lane boundaries. Chapter 4 develops the learned controller on this foundation.

Chapter 4

Learning Framework and Experimental Results

4.1 Learning Framework Overview

In this chapter, the learning-based controller is developed, and the experimental evidence supporting its design choices is presented. The algorithm and the reward are settled first on a compact engineered state, where learning is fast and the outcome is unambiguous. The observation is then opened up, first to planar LiDAR and then to the raw bird's-eye-view image, and the study turns to what richer sensing buys and what it costs. Because inserting raw perception into an otherwise successful policy destabilizes learning, a dedicated section develops the training methodology that makes high-dimensional-observation policies trainable at all, arriving at a geometry-distillation encoder that recovers from-scratch learning entirely. The BEV image contains sufficient task-relevant information, but training the policy to use the image effectively is difficult. Isolating and resolving this training failure is the central result of the chapter. The obstacle-navigation task, which is the one setting where the engineered state is genuinely insufficient, and the comparison against tuned classical controllers close it. Throughout, the aim is to separate two questions that are easy to conflate, whether the articulated control task benefits from richer sensing, and whether any benefit comes from the sensing itself or from the machinery used to turn it into features.

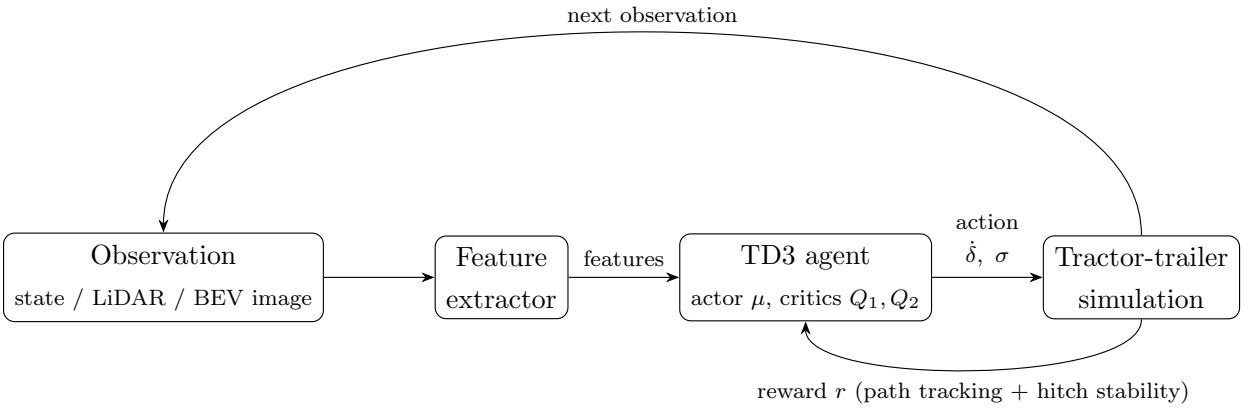


Figure 4.1: Architecture of the learning system: an observation (engineered state, LiDAR, or BEV image) is encoded into a feature vector and passed to the TD3 actor and critics, which output a steering rate and a stop signal. The reward combines path-tracking and hitch-stability terms.

The agent perceives its surroundings through one of three observation families, forming a deliberate progression from a fully engineered representation to raw imagery:

- **State**: an 8 input vector $[\delta, \gamma, e_y, e_\psi, e_{y,t}, e_{\psi,t}, \kappa_1, \kappa_2]$ containing the steering angle, the hitch articulation angle, the tractor and trailer cross-track and heading errors, and two local path-curvature samples.
- **Minimal State + LiDAR**: a 2 input vector $[\delta, \gamma]$ containing the steering angle and the hitch angle augmented with N normalized range readings, where the beam count $N \in \{8, 24, 64\}$ is varied in the LiDAR ablation of section 4.6.
- **Minimal State + BEV image**: The same minimal state vector and a 32×32 grayscale bird's-eye-view image.

The LiDAR and image modalities are taken up in section 4.5. This progression lets the marginal value of richer perception be measured against its added representation-learning and inference cost rather than assumed.

The LiDAR- and BEV-based observation spaces are augmented with the minimal intrinsic state vector because steering and hitch angles are available through onboard sensing and state estimation and therefore do not need to be inferred by the exteroceptive perception system.

4.2 Learning Algorithm and Training Configuration

4.2.1 Twin Delayed Deep Deterministic Policy Gradient

TD3 is the learning algorithm used throughout this thesis. It is an off-policy actor-critic method for continuous action spaces, and it extends deterministic policy-gradient methods with three improvements: twin critic networks (to reduce overestimation bias), delayed policy updates (to improve stability), and target policy smoothing (to reduce variance in critic updates). Three properties make it the natural candidate for articulated-vehicle control: the action space is continuous (a continuous steering rate and a stop signal that is triggered above a threshold), a deterministic policy avoids the action-distribution spread that can aggravate the non-minimum-phase reverse dynamics, and the off-policy replay buffer amortizes the comparatively high cost of image-based rollouts by reusing every transition many times. That reasoning is an expectation rather than a result. It is tested directly in section 4.4.2, where TD3 is compared against on-policy PPO, a discrete-action variant of PPO, and DQN across four reward formulations, and proves to be the only algorithm that solves the reverse task.

The TD3 actor network $\mu(s \mid \theta^{\mu})$ maps states to deterministic actions, while the critic networks $Q_1, Q_2(s, a \mid \theta^{Q_1}, \theta^{Q_2})$ evaluate state-action pair quality. The target value is computed using the minimum of the two critic estimates:

$$y = r_t + \gamma_d \min_{i=1,2} Q_i(s_{t+1},\, \mu(s_{t+1}) + \epsilon) \tag{4.1}$$

Where $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c)$ is clipped Gaussian noise added to the target action for smoothing. Parameters are updated by minimizing the temporal difference error in the critics and maximizing the expected return through the actor using the deterministic policy gradient theorem [26].

The implementation is Stable Baselines3 [69] on the Gymnasium environment API [4].

4.2.2 Training Configuration

The training configuration is held fixed across the observation and reward studies, so that an ablation changes only the variable under test. It is deliberately a conservative configuration rather than a tuned one: the library defaults for the off-policy algorithms, together with a modest learning rate and explicit exploration noise. An earlier throughput-oriented configuration, which raised the learning rate to 2×10^{-3} alongside a much larger minibatch and many gradient steps per environment step, diverged once the reward magnitude grew

and the stop action was added to the action space. Table 4.1 lists the settings actually used, and they are the regime in which every result reported here was obtained.

Table 4.1: Training configuration used for the algorithm and reward studies. Settings not listed are the Stable Baselines3 defaults for the algorithm in question, including the TD3 soft-update rate, policy delay, and target-noise parameters. A dash marks a setting that does not apply to that algorithm.

Setting	TD3	PPO	DQN
Learning rate	10^{-3}	10^{-3}	10^{-3}
Steps before learning	10,000	—	10,000
Minibatch size	256	4096	4096
Replay buffer	10^6	—	3×10^5
Update schedule	1 per episode, all steps	32 rollout steps, 5 epochs	1 per step, 16 gradient steps
Policy network	$[400, 300]$	$[256, 256]$	$[256, 256]$
Discount γ_d	0.99	0.99	0.99
Exploration	Gaussian action noise, reduced on stop channel	stochastic policy, clip range 0.2	ϵ: $1.0 \rightarrow 0.05$ over 20% of training

The same backbone and settings are shared across the state, LiDAR, and BEV observation variants wherever possible, so that the observation ablation isolates the effect of the observation and its representation rather than of the training configuration. This sharing cannot extend to the algorithm comparison of section 4.4.2, where TD3, PPO, and DQN each require the settings appropriate to their own update rule. There, only the environment, reward, and observation are held fixed, and each algorithm is given its own sensible configuration, so that the comparison reflects the algorithms rather than a single tuning that happens to favor one of them.

What was deliberately searched is the task design rather than the optimizer. The shape of the stop reward and its difficulty gate, the dynamic-difficulty threshold, and the warm-up schedule of the obstacle curriculum were each swept over repeated training runs, and those sweeps are reported with the obstacle results in section 4.8, because their outcome is a design finding rather than a nuisance parameter.

4.3 Evaluation Protocol

The observation, reward, and training designs of this chapter are assessed by a single evaluation protocol, described once here and applied to every number that follows.

Every policy is evaluated on a fixed pool of pre-generated scenarios rather than on freshly sampled episodes, so that different controllers face identical layouts. Outcomes are recorded categorically as completion, deliberate stop, or collision. Because the obstacle task's difficulty distribution is bimodal, results are reported stratified by the geometric difficulty of eq. (4.20) rather than pooled, so that easy and hard layouts are not averaged into a single misleading number. Each reported rate is a proportion, so it is given with a Wilson 95% confidence interval, and each configuration is trained over three random seeds and reported as a mean with its across-seed interval, so that a result's stability, and not only its central value, is visible. This matters throughout, because articulated control training is high-variance, and on the harder tasks the spread across seeds carries as much of the story as the mean does. Lane-following results are reported at the best checkpoint.

Generalization is treated as something to measure rather than assume. The evaluation pool is held out from training, and where two methods are close in mean performance the across-seed interval, rather than the point estimate alone, decides whether the difference is real. A gap of a few centimeters in mean cross-track error routinely sits inside the run-to-run scatter on the reverse task, and reporting a single best number would read that scatter as an effect.

A raw collision rate is misleading for a policy that can decline to attempt a layout, because stopping deflates it: a policy that stops on everything never crashes. The honest avoidance metric is therefore the collision rate conditional on attempting, crash/(complete + crash), reported alongside the raw collision rate and the stop rate. The minimum clearance to any obstacle over an episode is also reported, as a measure of how close the vehicle came to contact. The obstacle results, where declining to attempt is itself part of the behavior, are read through these attempt-conditional metrics rather than through completion alone.

To interpret the stop-gate behavior, the geometric difficulty is calibrated against actual drivability using a classical stack as an oracle. The traditional pipeline (a global path, an artificial-potential-field local planner, and a pure-pursuit tracker, the last of which is one of the classical baselines detailed in chapter A) is run with the stop action disabled so that it always attempts, and its per-difficulty completion rate is read as an empirical drivability $P_{\text{clear}}(d)$. This turns the static geometric proxy of eq. (4.20) into a measured, direction-specific quantity, and it fixes the difficulty at which stopping becomes rational. With the terminal rewards used in this comparison (success $+100$, collision -500), the expected value

of attempting a layout is

$$\mathbb{E}[\text{attempt}](d) = 100\, P_{\text{clear}}(d) - 500\,\big(1 - P_{\text{clear}}(d)\big), \tag{4.2}$$

which is zero at $P_{\text{clear}} = 5/6 \approx 83.3\%$. The difficulty at which measured drivability crosses this break-even is the true, per-direction drivability threshold $d^\star$ against which the stop-gate policy should be judged, rather than the raw geometric scale. The forward and reverse values of $d^\star$, and what they reveal about the reverse task in particular, are reported in section 4.8.

So that completion measures tracking rather than stop-timing, the stop action is disabled during the lane-following and drivability evaluations and enabled only when the stop-gate behavior itself is under test.

4.4 Reward Formulation and Selection

4.4.1 Reward Variants

The thesis evaluates several reward variants rather than a single scalar shaping rule. Let e_y and e_ψ denote the tractor cross-track and heading errors, $e_{y,t}$ and $e_{\psi,t}$ the trailer cross-track and heading errors, γ the hitch articulation angle, $\dot{x}$ the longitudinal speed, and δ the steering angle. Longitudinal speed is not a control degree of freedom: it is held at the fixed operating value from chapter 3 ($\dot{x} = \pm 5\,\text{m/s}$ forward or reverse), so $\dot{x}$ appears in the reward only as a near-constant progress term that a stop forfeits, and the second action dimension is a stop signal rather than a commanded speed (chapter 3). The pure-pursuit guide steering is denoted δ_{pp}, with steering deviation $\Delta_\delta = (\delta - \delta_{\text{pp}})/(2\delta_{\max})$. The lane-following base environment also defines a proximity penalty P_{prox} that is subtracted from non-terminal step rewards. This penalty is zero when the vehicle is farther than $d_{\text{prox}} = 2\,\text{m}$ from blocked occupancy-grid cells and increases quadratically to $P_{\max} = 5$ as clearance approaches zero:

$$P_{\text{prox}} = \begin{cases} 0, & c \geq d_{\text{prox}}, \\ P_{\max}\left(1 - \frac{c}{d_{\text{prox}}}\right)^2, & c < d_{\text{prox}}, \end{cases} \tag{4.3}$$

where c is the minimum clearance between the rendered vehicle footprint and the lane-boundary/obstacle occupancy grid. In the no-obstacle lane-following environment this discourages driving near lane boundaries; obstacle environments inherit the same term for both lane boundaries and inserted obstacle cells.

Forward lane-following rewards

For forward driving, the terminal reward is

$$R_{\text{term}}^{\text{fwd}} = \begin{cases} 100, & \text{success}, \\ -100, & \text{failure}, \end{cases} \tag{4.4}$$

and the common shaping terms are

$$r_p^{\text{fwd}} = 0.5\,\dot{x}, \tag{4.5}$$

$$P_{\text{tractor}} = e_y^2 + e_\psi^2, \tag{4.6}$$

$$P_{\text{trailer}} = (0.5\,e_{y,t})^2 + (0.5\,e_{\psi,t})^2. \tag{4.7}$$

The four forward reward variants are then

$$R_{\text{dense}}^{\text{fwd}} = r_p^{\text{fwd}} - P_{\text{tractor}} - P_{\text{trailer}} - P_{\text{prox}}, \tag{4.8}$$

$$R_{\text{tractor}}^{\text{fwd}} = r_p^{\text{fwd}} - P_{\text{tractor}} - P_{\text{prox}}, \tag{4.9}$$

$$\begin{aligned} R_{\text{mult}}^{\text{fwd}} &= 4\exp(-|e_y| - 0.5|e_\psi| - 0.75|e_{y,t}| - 0.5|e_{\psi,t}|)\exp(-1.5|\gamma|) \\ &\quad + 0.5\,\text{clip}(\dot{x}, 0, 1) - P_{\text{prox}}, \end{aligned} \tag{4.10}$$

$$R_{\text{guide}}^{\text{fwd}}(k) = \alpha_k \left(-5\Delta_\delta^2 - P_{\text{prox}}\right) + (1 - \alpha_k)\, R_{\text{mult}}^{\text{fwd}}. \tag{4.11}$$

The guided reward interpolates between a pure-pursuit imitation term $-5\Delta_\delta^2$, which rewards matching the pure-pursuit steering, and the multiplicative reward $R_{\text{mult}}^{\text{fwd}}$ defined above. The schedule $\alpha_k \in [0, 1]$ governs the transition from imitation toward the task reward and is described in section 4.7, where the same mechanism is reused to fine-tune an imitated policy. At $\alpha_k = 1$ the guided reward is a pure-pursuit clone.

Reverse lane-following rewards

For reverse driving, the terminal reward is

$$R_{\text{term}}^{\text{rev}} = \begin{cases} 200, & \text{success}, \\ -500, & \text{failure}, \end{cases} \tag{4.12}$$

The terminal rewards for the reverse scenario are significantly larger magnitude than the forward variant to encourage the policy to continue driving and learn to stabilize the hitch angle while tracking the path. Using the same terminal rewards as the forward scenario resulted in reverse policies that intentionally ended each episode early with failure to avoid

the penalty from poor path tracking that accumulated over each step of the episode.

The shared shaping terms are

$$r_p^{\text{rev}} = 3\,\text{clip}(-\dot{x}, 0, 1), \tag{4.13}$$

$$P_{\text{path}} = e_y^2 + 0.5\,e_\psi^2 + 0.5\,e_{y,t}^2 + 0.25\,e_{\psi,t}^2, \tag{4.14}$$

$$P_{\text{jack}} = 10\max(0, |\gamma| - 0.4)^2. \tag{4.15}$$

The reverse variants are

$$R_{\text{dense}}^{\text{rev}} = r_p^{\text{rev}} - P_{\text{path}} - P_{\text{jack}} - P_{\text{prox}}, \tag{4.16}$$

$$R_{\text{no-hitch}}^{\text{rev}} = r_p^{\text{rev}} - P_{\text{path}} - P_{\text{prox}}, \tag{4.17}$$

$$\begin{aligned} R_{\text{mult}}^{\text{rev}} &= 5\exp(-|e_y| - 0.5|e_\psi| - 0.75|e_{y,t}| - 0.5|e_{\psi,t}|)\exp(-2|\gamma|) \\ &\quad + 0.5\,\text{clip}(-\dot{x}, 0, 1) - P_{\text{prox}}, \end{aligned} \tag{4.18}$$

$$R_{\text{guide}}^{\text{rev}}(k) = \alpha_k\left(-5\Delta_\delta^2 - P_{\text{prox}}\right) + (1 - \alpha_k)\,R_{\text{mult}}^{\text{rev}}. \tag{4.19}$$

Obstacle difficulty

The obstacle-aware terms below are keyed to a scalar difficulty $d \in [0, 1]$, a static geometric clearance ratio computed as the tightest cross-section in the observable path window:

$$d = \text{clip}\left(1 - \frac{g_{\max} - w_v}{w_L - w_v},\ 0,\ 1\right), \tag{4.20}$$

where $g_{\max}$ is the widest contiguous free lateral gap, w_v the vehicle width, and w_L the lane width, so $d = 0$ is a fully open lane and $d = 1$ a gap at or below the vehicle width. Its rationale and per-direction calibration are developed in section 4.8.1 and chapter 4.

Obstacle-aware extensions

Obstacle environments inherit the underlying forward or reverse reward, including the proximity penalty above. A failed obstacle episode receives the standard terminal penalty (chapter 3). A deliberate stop instead receives a difficulty-scaled terminal reward,

$$s(d) = \frac{e^{k_d d} - 1}{e^{k_d} - 1}, \qquad k_d = 3, \tag{4.21}$$

$$R_{\text{stop}}^{\text{obs}} = s(d)\,B_{\text{hard}} - \big(1 - s(d)\big)B_{\text{easy}} + p\,B_{\text{prog}}, \tag{4.22}$$

where $p \in [0, 1]$ is the progress made before stopping and $B_{\text{hard}} = B_{\text{easy}} = 60$, $B_{\text{prog}} = 20$. The soft difficulty gate $s(d)$ makes stopping pay on an impassable layout, where it must beat

a crash, and lose to driving through on an easy one.

Branching hidden-path reward

Rather than tracking the single hidden avoidance path of chapter 3, the obstacle reward tracks two hidden corridors, one passing each side of the obstacle, and keeps the more favorable of the two evaluations:

$$R^{\text{branch}} = \max_{c \in \{\text{L},\,\text{R}\}} R\Big(e_y^{(c)},\, e_\psi^{(c)},\, e_{y,t}^{(c)},\, e_{\psi,t}^{(c)}\Big)\,, \tag{4.23}$$

where $e_\bullet^{(c)}$ are the tracking errors relative to corridor c and $R(\cdot)$ is the underlying forward or reverse reward above. Its rationale is developed in section 4.8.1.

The variants above differ chiefly in how the tracking and articulation terms combine, additively or as a product that forces the errors down jointly rather than letting the agent collect partial credit from a single term, and are compared empirically in section 4.4.2. Table 4.2 summarizes them. The obstacle-specific terms defined above, the difficulty rating, the stop gate, and the branching hidden-path reward, are motivated in section 4.8.1 where the obstacle task itself is taken up.

Table 4.2: Reward variants compared in chapter 4. Every variant also includes the progress term r_p and subtracts the proximity penalty P_{prox}, with terminal rewards as defined above. The variants differ in how the tracking and articulation terms combine.

Variant	Tracking penalty	Articulation term	Role
Forward			
Dense	Additive, tractor + trailer	None	Baseline
Tractor-focus	Additive, tractor only	None	Drops trailer term
Multiplicative	Multiplicative product	$\exp(-1.5\lvert\gamma\rvert)$ factor	Chosen recipe
Guided (PP)	Multiplicative (via $R^{\text{fwd}}_{\text{mult}}$)	Via $R^{\text{fwd}}_{\text{mult}}$	Anneals from pure-pursuit imitation
Reverse			
Dense	Additive (P_{path})	P_{jack}	Baseline
No-hitch	Additive (P_{path})	None	Drops jackknife term
Multiplicative	Multiplicative product	$\exp(-2\lvert\gamma\rvert)$ factor	Chosen recipe
Guided (PP)	Multiplicative (via $R^{\text{rev}}_{\text{mult}}$)	Via $R^{\text{rev}}_{\text{mult}}$	Anneals from pure-pursuit imitation

4.4.2 Selecting the Algorithm and the Reward

The first question is which learning algorithm and which reward give stable articulated tracking, and the cleanest place to answer it is the engineered state, where perception adds

nothing to confound the comparison. Four algorithms, TD3, PPO, a discrete-action PPO, and DQN, were each trained under four reward formulations, in both the forward and the reverse direction, over three seeds. The environment, the observation, and the evaluation were held fixed, and each algorithm was given the settings appropriate to its own update rule as described in section 4.2.2. Tables 4.3 and 4.4 report best-checkpoint completion for every cell.

Table 4.3: Forward lane-following, best-checkpoint completion rate (%) for every algorithm and reward combination on the engineered state observation. Cells are the mean over three seeds with a 95% confidence interval. The chosen recipe is highlighted.

Algorithm	Dense	Tractor-focus	Multiplicative	Guided (PP)
TD3	100.0±0.0	100.0±0.1	100.0±0.0	100.0±0.0
PPO	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
DQN	100.0±0.0	100.0±0.0	85.0±15.2	60.6±59.1
PPO (discrete)	83.3±32.6	90.7±18.2	100.0±0.0	49.0±1.5

Table 4.4: Reverse lane-following, best-checkpoint completion rate (%) for every algorithm and reward combination on the engineered state observation. Cells are the mean over three seeds with a 95% confidence interval. The chosen recipe is highlighted.

Algorithm	Dense	No-hitch	Multiplicative	Guided (PP)
TD3	79.2±4.3	80.5±12.8	**95.9±0.6**	86.1±6.5
PPO	16.1±31.4	16.4±32.1	48.2±0.8	82.0±24.6
DQN	16.9±32.8	1.4±2.8	0.0±0.0	48.7±55.1
PPO (discrete)	47.5±1.2	32.5±31.9	15.8±31.0	57.3±8.9

Forward lane-following is saturated. Both continuous-control algorithms complete every forward layout under every reward, which matches the classical baselines, where pure pursuit, PID, LQR, and MPC all reach full completion forward. Only the combination of a discrete action space and a poorly matched reward breaks the forward task at all. Forward tracking is therefore not where the algorithm or the reward is decided, and the forward reward choice is left to the tracking-quality detail rather than to completion.

Reverse lane-following is where the choice is made, and it confirms the expectation set out in section 4.2.1. The open-loop reverse dynamics are unstable, as established in chapter 3, so early policies fail often and the algorithm has to recover a stabilizing controller rather than merely refine a tracker. TD3 is the only algorithm that solves the reverse task, with

every one of its reward variants completing at least 79% of the pool. The best non-TD3 reverse cell, PPO under the guided reward at 82.0%, carries a confidence interval of ±24.6, so its apparent success rests on a single lucky seed rather than on reliable learning. The best reward is also algorithm-conditional. TD3 pairs best with the multiplicative reward, which forces the tracking and articulation errors down jointly, whereas the on-policy PPO only works under the guided reward that clones the pure-pursuit teacher, and the multiplicative reward is close to catastrophic for DQN. The guided reward fails both discrete algorithms in both directions, which is the expected behavior of a reward that clones a continuous teacher into an action space that cannot represent it, and serves as a clean negative control.

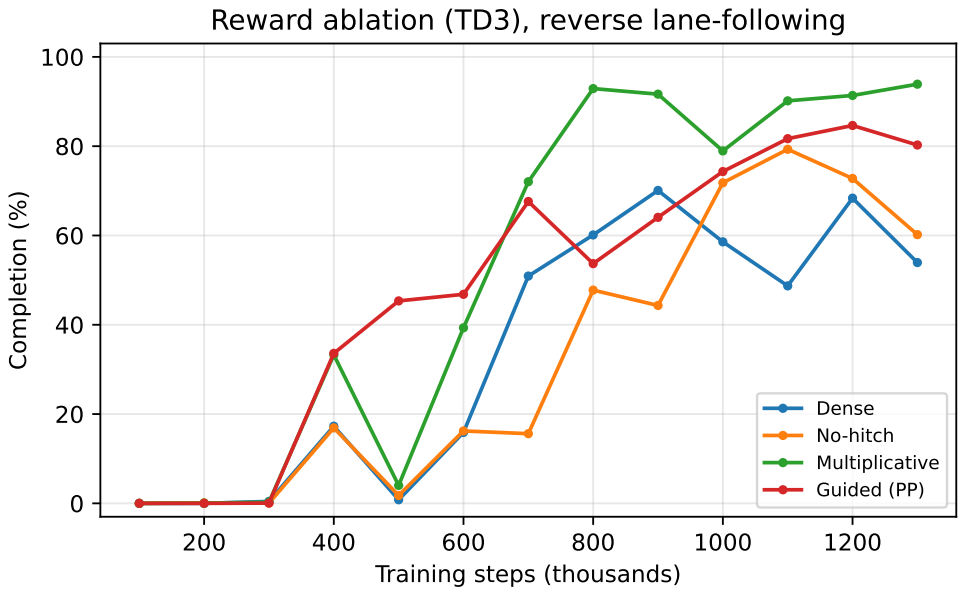


Figure 4.2: Reverse lane-following completion against training steps for TD3 under each reward formulation. The multiplicative reward reaches the highest and steadiest completion.

The chosen recipe is therefore TD3 with the multiplicative reward, and the reverse task is where that choice is justified. Table 4.5 looks past completion to the tracking detail for TD3, and the multiplicative reward is best on every axis at once, the highest completion at 95.9%, the lowest trailer cross-track error, and the tightest hitch behavior. This single configuration, TD3 with the multiplicative reward on the engineered state, reaches 0.959 ± 0.006 reverse completion with no collapsed seeds, and is carried into the rest of the chapter. Holding it fixed is what lets every later ablation change one design variable at a time.

Table 4.5: Reverse lane-following, TD3 reward detail on the engineered state observation. Beyond completion the multiplicative reward also gives the lowest trailer cross-track error and the tightest hitch behavior, which is why it is carried forward. Mean over three seeds, 95% confidence interval on completion.

Reward	Completion (%)	Trailer CTE (m)	Max $\lvert\gamma\rvert$ (°)	Jackknife (%)
Dense	79.2±4.3	1.14	17.5	1.2
No-hitch	80.5±12.8	1.01	14.7	0.5
Multiplicative	**95.9±0.6**	0.94	12.3	0.3
Guided (PP)	86.1±6.5	1.20	11.5	2.0

4.5 Observation and Perception Design

The core learning system above operates on the engineered state vector. This section introduces the two higher-dimensional observation modalities that this thesis compares against that baseline, a planar LiDAR scan and a bird's-eye-view image, and the representation learning required to turn raw imagery into features the policy can use.

LiDAR Observation

The obstacle environments expose a simulated planar LiDAR observation by appending normalized obstacle distances to the state vector. In forward driving the LiDAR pose is attached to the front tractor, while in reverse driving it is attached to the trailer and rotated by π so that the sensing direction follows the leading unit and covers the area that the system is driving towards. This keeps the sensing model consistent with the physical task, in which the front of the articulated system changes when the direction of travel reverses. The LiDAR is a qualitatively different modality from the image, a set of range scans rather than a two-dimensional picture, and the beam count is treated as a resolution knob in the ablation of section 4.6.

Bird's-Eye-View Observation

The bird's-eye-view (BEV) observation is a 32×32 single-channel top-down image of a local region (approximately a 20 m square) centered on the vehicle and oriented so that the direction of travel points toward the top of the image. It encodes the task-relevant spatial structure that the state vector does not represent: lane geometry and obstacle layout. Forward and reverse variants share the same size and format, differing only in the crop

direction and in the interpretation of the accompanying state vector, which uses trailer-referenced tracking errors in reverse because the trailer leads the maneuver.

As introduced in section 3.4, the BEV image originates from the Pygame renderer, which draws the top-down scene as a grayscale frame that is cropped to the local window. For large-scale training the same observation is produced without the renderer by the batched GPU simulator (section 3.6), which rasterizes it analytically as an occupancy image, where each pixel records whether the corresponding ground point is drivable (inside the lane corridor and clear of obstacles) or blocked. The two share resolution and role, though the analytic form omits photometric detail and does not draw the vehicle body, which sits at the image center by construction.

4.5.1 Image Encoding and Representation Learning

The BEV image must be turned into a feature vector before it can drive the policy. Two routes are studied: training an image encoder end-to-end with the agent, and pretraining an encoder so that a usable visual representation is available from the first control step.

A Convolutional Neural Network (CNN) [41] serves as the end-to-end image front-end, implemented as a custom feature extractor within Stable Baselines3 [69]. Matching the scale of the 32×32 observation, the image branch is deliberately small, a two-layer convolutional encoder with $1 \to 5 \to 5$ channels maps the image to an 80-dimensional feature vector, followed by a linear projection to 64 image features. A separate two-layer multilayer perceptron embeds the 8 input state vector to 64 dimensions. The two branches are concatenated into a 128-dimensional feature vector for the TD3 actor and critic heads. Two choices in this extractor are made specifically for stability with a high-dimensional input: layer normalization is applied to the image features so their scale cannot swamp the intrinsic state embedding, and the actor and critic feature extractors are kept separate, following the TD3 default, so that critic representation updates do not overwrite the actor's features.

Training this image branch end-to-end is not free of pitfalls. Inserting a randomly initialized CNN branch into an otherwise successful state-based policy can harm early TD3 learning. The agent converged to a short-horizon wall-collision behavior even though the same vector state was sufficient for rapid learning. Two diagnostic encoders isolate this effect. The `state_only` BEV encoder keeps the dictionary observation and multi-input policy path but ignores the image branch, confirming that the BEV environment and multi-input wrapper can still learn from the vector state alone. The `scaled_cnn` encoder instead introduces the image branch gradually, scaling the image features by a coefficient that ramps from zero to one over training, with an optional second schedule that fades out the extrinsic

vector components once the image branch carries full weight.

These diagnostics motivate decoupling perception from control, and this thesis pursues a progression of ways to do so, developed in section 4.7. The first and most direct decouples at the level of the representation. If a random image branch can derail learning that the vector state alone proved learnable, then supplying a usable visual representation from the first training step should remove that failure mode. An autoencoder is used to pretrain an image representation before policy optimization [48]: an encoder maps the image observation to a compact feature vector, and a decoder reconstructs the image from it. The autoencoder is pretrained on BEV images collected from a mixture of simple pure-pursuit rollouts, already-trained compatible policies, and random actions, to broaden state coverage. After reconstruction training, the encoder weights are transferred into the RL feature extractor and can be frozen or fine-tuned while the policy is trained.

A UNet-style [51] autoencoder provides a second encoder pre-training method with a stronger structural prior. Its encoder uses repeated double-convolution blocks with max pooling, followed by global average pooling to produce a compact 128-dimensional image descriptor. For reinforcement learning only the encoder path is kept, concatenated with the vector embedding. It tests whether preserving multi-scale spatial detail during representation learning improves downstream control, particularly in scenes where obstacle shape and lane boundaries must both be interpreted from the image.

Together, the end-to-end CNN, the frozen and fine-tuned autoencoder encoders, and the UNet encoder form a spectrum of image representations, from learned-during-control to learned-before-control, that is compared against the engineered state and LiDAR in section 4.6. Pretraining the encoder is only the first of the decoupling strategies, and on its own it addresses only part of the problem. The stronger routes decouple at the level of the policy and of perception itself, and they rest on a specific view of why raw vision destabilizes learning. That view, and the training methodology built on it, are the subject of section 4.7.

4.5.2 Observation Space Summary

The full observation interface used across experiments is summarized in table 4.6.

Table 4.6: Observation space components

Component	Description	Role
`vector`	8 input state: $[\delta, \gamma, e_y, e_\psi, e_{y,t}, e_{\psi,t}, \kappa_1, \kappa_2]$	Provides precise articulation and path-tracking state information
`lidar`	N normalized obstacle ranges, with $N \in \{8, 24, 64\}$	Encodes local free space around the leading unit without image processing
`image`	32×32 BEV crop, binary occupancy or signed-distance field, with optional coordinate channels	Encodes lane layout, local path shape, obstacle positions, and vehicle geometry

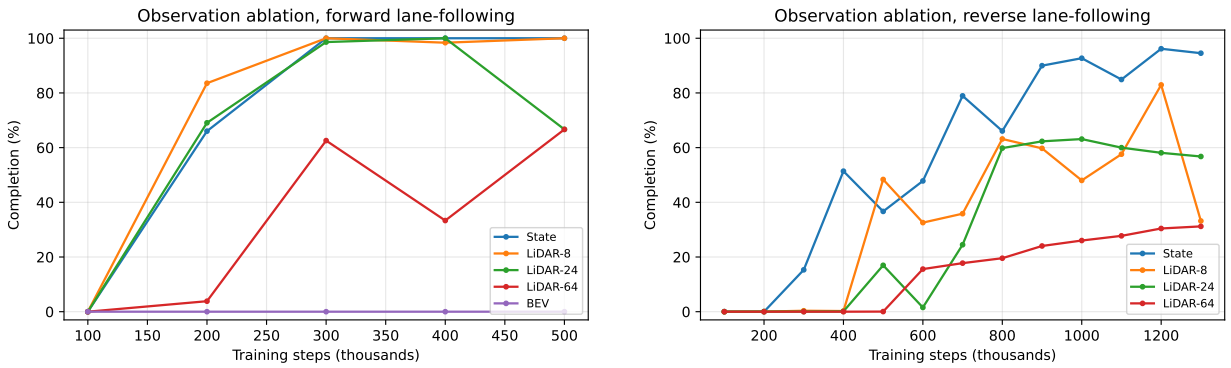
4.6 Observation-Space Ablation

With the algorithm and reward fixed, the observation can be opened up. The engineered state is a compact, hand-designed summary of exactly the quantities a tracking controller needs, while LiDAR and the bird's-eye-view image carry more of the raw scene but leave the policy to extract those quantities for itself. The question this section answers is what that trade buys. The comparison uses the identical TD3 recipe for every observation, with no curriculum, no failure replay, and no representation warm-up, so that any difference is attributable to the observation and its dimensionality rather than to the training treatment. Table 4.7 reports completion and trailer cross-track error for the engineered state, three LiDAR resolutions, and the raw image, in both directions.

Table 4.7: Observation-space ablation on no-obstacle lane-following, identical TD3 recipe (multiplicative reward, fixed speed) for every observation. Best-checkpoint completion and trailer cross-track error, mean over three seeds with a 95% confidence interval on completion. Reliability falls as the observation grows, and the effect is sharper in reverse. The reverse BEV run is deferred to the vision-rescue study.

Observation (dim)	Forward completion (%)	Forward CTE (m)	Reverse completion (%)	Reverse CTE (m)
State (8)	$100.0_{\pm 0.0}$	0.34	$97.0_{\pm 0.4}$	1.01
LiDAR-8 (10)	$100.0_{\pm 0.0}$	0.41	$90.0_{\pm 12.5}$	1.31
LiDAR-24 (26)	$100.0_{\pm 0.0}$	0.46	$63.1_{\pm 61.9}$	1.21
LiDAR-64 (66)	$66.7_{\pm 65.3}$	0.95	$31.2_{\pm 61.1}$	1.00
BEV (32×32)	$0.0_{\pm 0.0}$	1.17	—	—

The result contradicts the expectation that more sensing is better. Reliability falls as the observation grows, and the effect sharpens on the harder reverse task. The engineered state is not merely sufficient but the most robust option in both directions. The LiDAR variants show the same pattern internally, which reframes the beam count as a resolution-against-reliability knob rather than a more-is-better axis. The 8-beam scan is the most robust of the LiDAR options, matching the state forward and reaching 90% reverse, while raising the count to 24 and then 64 beams widens the across-seed spread and then brings on outright seed collapses. The wide confidence intervals on the reverse LiDAR-24 and LiDAR-64 rows are not noise to be averaged away, they are the signature of that collapse, with individual seeds falling to zero completion while others still succeed. The point retained here is that added input dimensions cost reliability on this task rather than earning it.



(a) Forward

(b) Reverse

Figure 4.3: Observation-space ablation learning curves, completion against training steps for the identical TD3 recipe under each observation. The raw image never leaves zero, and the higher-dimensional LiDAR scans learn less reliably than the compact state.

The raw image is the extreme case. Trained from scratch with the same recipe that solves the state task, the bird's-eye-view policy never completes a single forward layout, and the reverse image run was set aside for the dedicated study of the next section. This is a striking outcome, because the image plainly contains the geometry the state encodes, and more besides. Whatever is stopping the image policy from learning, it is not a shortage of information in the observation. The next section examines why.

4.7 Training High-Dimensional-Observation Policies

An image observation that contains the exteroceptive geometry the engineered state summarizes, yet cannot learn the task the state solves easily, is a contradiction that must be resolved, because the obstacle task of section 4.8 requires the image. Introducing a raw image into the observation does more than add parameters. It changes what it takes to train the policy at all. This section first sets out a hypothesis for why that happens, then works through three progressively cleaner ways to decouple perception from control: pre-training the representation, warm-starting the policy by imitation, and distilling the state into a frozen perception module. Each route addresses more of the hypothesized failure than the last, and the third recovers ordinary from-scratch learning entirely. The evidence that isolates the cause and separates the three routes follows in sections 4.7.4 and 4.7.5.

4.7.1 An Off-Policy Instability Hypothesis

The failure is hypothesized to be an optimization pathology of off-policy actor-critic learning rather than a limitation of perception. In TD3 the actor is trained to maximize the critic's value estimate, while the critic is trained by bootstrapping from its own estimates of the actor's next actions. Early in training the critic is essentially random, and it over-estimates the value of actions far from the data it has seen. This is extrapolation error [75], a manifestation of the deadly triad that arises when bootstrapping, function approximation, and off-policy updates are combined [24]. With a low-dimensional state the actor recovers quickly. With a high-dimensional image, the actor and its convolutional encoder have far more capacity to chase the critic into these over-valued, off-distribution actions before the critic corrects, so the policy can settle into a degenerate short-horizon behavior. Two ideas from the literature target the same mechanism and inform the designs below: constraining the actor to stay near the behavior data, as in the TD3+BC (Behavior Cloning) offline method [76], and decoupling the encoder from the reward-driven critic gradient, as in the stop-gradient image encoder of SAC-AE [77]. The stronger claim, that this mechanism and not the image content, the representation, or the encoder is the true cause, is tested directly by the causal ablation of section 4.7.4.

4.7.2 Representation Pretraining and Warm-Starting the Policy

The first and most direct response, introduced in section 4.5, is to pretrain the image encoder by reconstruction (the autoencoder and UNet) so that a usable representation exists from the first control step. This attacks the noisy co-trained encoder head-on, but it addresses only one of the two failure sources: it does nothing about the actor being dragged off-distribution by a cold critic, and reconstruction spends encoder capacity on task-irrelevant pixel detail rather than on the geometry the controller needs. It is a partial fix, and it motivates the two stronger routes below.

If a cold critic dragging the actor off-distribution is the problem, then starting from a policy that is already good, and keeping it there, should avoid it. This second decoupling strategy acts at the level of the policy rather than the representation. A competent low-dimensional `state` observation agent is trained first and then used as an expert. It is rolled out inside the image environment, its observations and actions are recorded, and the image policy (convolutional encoder and actor head together) is trained by supervised regression to reproduce the expert's actions. This behavior-cloning warm-start [44] supplies a non-degenerate policy and encoder from the first reinforcement-learning step, so the actor never has to survive a cold critic from scratch. One caveat, taken up in section 4.7.4, is that the

observation still contains the full state vector and the expert's actions derive from it, so the cloning loss can be satisfied largely through the state branch.

A single annealing schedule governs the transition from imitation to improvement: $\alpha_k = \max(0,\, 1 - k/T)$, which decays from one to zero over T steps. It appeared already in the guided reward of section 4.4.2, where it blends the pure-pursuit imitation term into the multiplicative task reward, so that the agent first clones the classical controller and is then progressively rewarded for surpassing it. Holding α_k near one keeps the agent close to its imitation while the critic warms up, and decaying it hands control to the task reward once the values are trustworthy. Fixing $\alpha_k = 1$ throughout instead recovers pure imitation, which is how the guided reward is used when a stable clone, rather than improvement, is the goal.

4.7.3 Geometry Distillation

The cleanest route removes both failure sources at once, by refusing to let reinforcement learning shape the encoder at all. A convolutional encoder is trained supervised to predict the vehicle and path geometry directly from the bird's-eye-view image, using the simulator's ground-truth state as a privileged teacher label. This is the privileged-distillation, or learning-by-cheating, paradigm [78]. The trained encoder is then frozen and used as a fixed perception module, and the policy is trained from scratch with ordinary TD3 over a low-dimensional vector, exactly as in the engineered-state baseline. Figure 4.4 shows the two-stage scheme.

The encoder takes the SDF-BEV with its two coordinate channels (section 4.5) through a small three-layer convolutional tower into a 128-dimensional trunk with layer normalization, from which a linear head regresses the standardized geometry vector. The observation the policy receives is then split along the physics of the problem. The internal states, the steering angle δ and the hitch articulation angle γ, are supplied as true proprioception, because they are available on the vehicle as measurements or estimates from on-board sensing and, being internal, are less reliably recoverable from an ego-centric top-down image, as noted in section 4.5. The exteroceptive geometry, the tractor and trailer tracking errors and the path-curvature look-aheads, is taken from the frozen encoder's prediction, because that is precisely what an external top-down view encodes.

This design removes both failure sources identified above. Because the encoder is trained only by supervised regression and then frozen, the critic is never fed the non-stationary, high-variance features of a co-trained encoder, so the concern behind Route A does not arise. Because the policy is trained from scratch on the frozen features, its actor and critic co-evolve from a cold start, so there is no pretrained-actor-versus-fresh-critic mismatch of

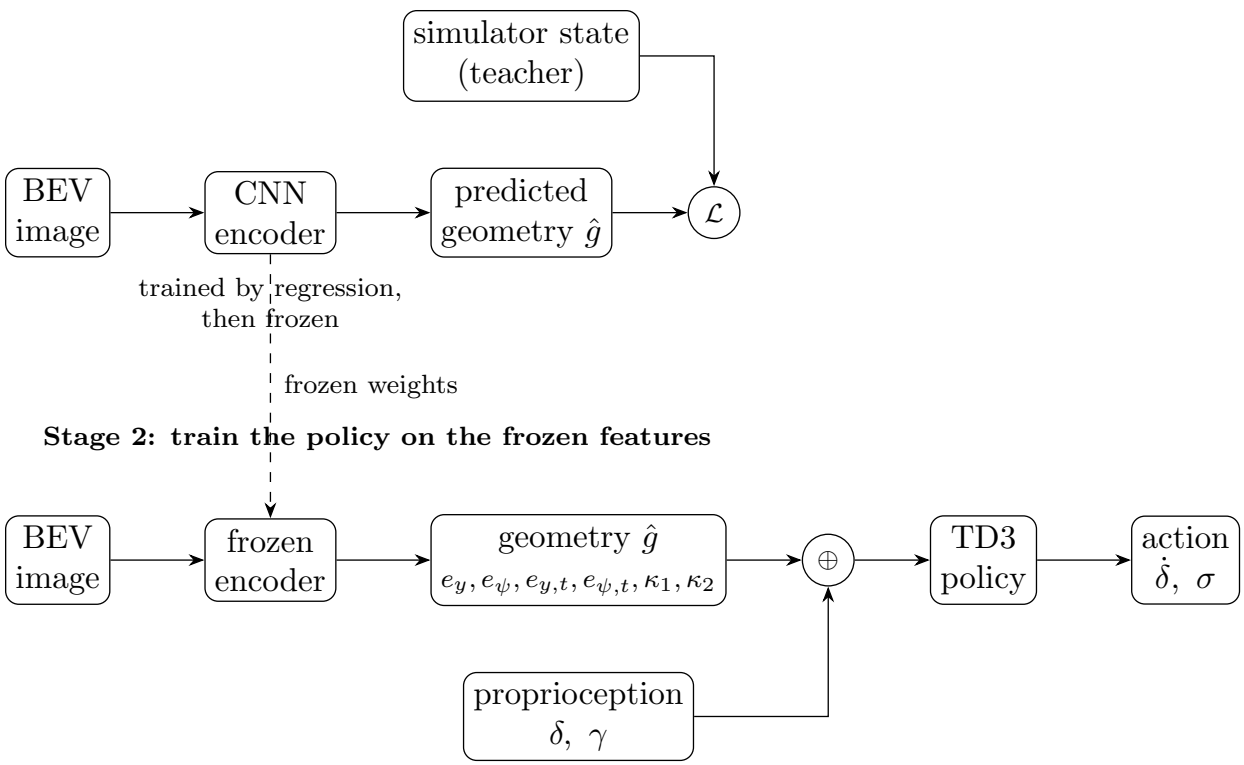


Figure 4.4: Geometry-distillation training. In Stage 1 a convolutional encoder is trained by supervised regression to predict the vehicle's path geometry from the bird's-eye-view image, using the simulator's true state as a privileged teacher, and is then frozen. In Stage 2 the policy is trained from scratch with TD3 on the frozen geometry prediction concatenated with the steering and hitch angles supplied as proprioception.

the kind Route B must manage. Distillation also settles a question the imitation route could not: predicting the state from the image alone cannot be satisfied through a state branch, so success at the prediction task is direct evidence that the image is a sufficient statistic for the geometry it encodes. The per-component recoverability of the geometry and the from-scratch learning curve are reported in section 4.7.5.

4.7.4 Isolating the Failure Mode

The evidence takes the form of a ladder in which each rung changes exactly one thing relative to the rung above. Table 4.8 lays it out on the forward task.

Table 4.8: The vision-training ladder on forward lane-following. Each rung changes one thing relative to the rung above, so the first configuration that reaches full completion identifies the cause. Representation changes alone leave the policy at zero, imitation restores it, unconstrained fine-tuning destroys it again, and only freezing the actor holds it, which points to the actor rather than the encoder as the failure. Completions are from the development runs. The actor-frozen configuration was repeated over three seeds (1.00 / 1.00 / 0.92).

Configuration	Completion	What it isolates
BEV occupancy, from scratch	0.00	Ordinary RL on the raw image fails outright
+ signed-distance encoding	0.00	A denser, task-aligned image is not the fix
+ coordinate channels	0.00	Spatial addressing is not the fix
+ stability extractor	0.00	Layer normalization and separate actor and critic encoders are not enough
Imitation warm-start	1.00	A cloned state expert produces a working vision policy
+ naive fine-tuning	0.00	Letting RL move the whole policy destroys the clone
+ RL, encoder frozen	0.00	Freezing perception does not save it, so the encoder is not the culprit
+ RL, actor frozen	1.00	Freezing the actor and training only the critic holds parity. Further training of the actor afterwards does not guarantee improvement, and sometimes destroys the policy

The representation-side fixes come first and none of them work. Rendering the image as a signed-distance field rather than a hard occupancy mask, adding coordinate channels, and giving the encoder the stabilizing treatment of layer normalization with separate actor and critic front-ends all leave completion at zero. If the trouble were that the image is a poor learning target, one of these would have improved completion. None does, which points away from the representation as the cause. The picture changes the moment the policy is warm-started by imitation. Cloning the competent state expert into the image policy produces a working controller with full completion, which shows that a non-degenerate policy over the image is reachable. The decisive rungs are what happens next. Letting ordinary reinforcement learning then fine-tune that cloned policy destroys it, dropping completion

back to zero, and freezing the encoder while the rest of the policy trains does not save it either. Only freezing the actor, and training the critic alone, holds the cloned policy at parity. Read down the ladder, the cause is neither the image nor the encoder but the actor being driven off-distribution by an untrained critic, exactly the mechanism hypothesized in section 4.7. The actor-frozen configuration was repeated over three seeds and held, at 1.00, 1.00, and 0.92 completion.

The last two rungs also answer a question the warm-start raises on its own, namely whether reinforcement learning can improve a cloned policy rather than merely preserve it. Freezing the actor holds parity but, by construction, can never improve the policy it never updates, and allowing the actor to move again, even under a reduced learning rate, either collapses it outright or leaves it oscillating with no gain in cross-track error. Unconstrained actor unfreezing is therefore not a usable recipe on this task.

This diagnosis also exposes why the imitation warm-start, though it works, is not a complete result. Because the image observation still carries the full state vector, the cloning objective can be met largely through the state branch, so a working cloned policy does not on its own prove that the convolutional encoder learned to read the image. Settling that requires an approach where the image alone must supply the geometry.

4.7.5 What the Frozen Encoder Recovers

Two measurements test the geometry-distillation recipe of section 4.7.3. The first asks whether the image is a sufficient statistic for the geometry, by scoring how well each state component can be predicted from the image alone. Table 4.9 reports the per-component test R^2. The exteroceptive geometry, the tractor and trailer tracking errors and the two curvature look-aheads, is recovered almost perfectly, with R^2 at or above 0.97 for seven of the eight components. Only the internal steering angle resists, at 0.66, which is exactly the component a top-down view of the surroundings cannot see, and why steering is supplied as proprioception rather than predicted. This is the direct evidence the imitation route could not provide: the image really does encode the path geometry.

Table 4.9: Per-component test R^2 of predicting each state variable from the bird's-eye-view image alone, using the frozen geometry encoder. The exteroceptive path geometry is recovered almost perfectly, while the internal steering angle resists, which is why steering and the hitch angle are supplied as true proprioception rather than predicted.

Component	Description	R^2
e_ψ	tractor heading error, exteroceptive	0.998
e_y	tractor cross-track error, exteroceptive	0.997
κ_1	near curvature sample, exteroceptive	0.997
κ_2	far curvature sample, exteroceptive	0.997
γ	hitch angle, partly visible through the path history	0.995
$e_{\psi,t}$	trailer heading error, exteroceptive	0.993
$e_{y,t}$	trailer cross-track error, exteroceptive	0.974
δ	steering angle, an internal actuator state the image cannot see	0.659

The second measurement asks whether a policy can then learn on these frozen features. Figure 4.5 shows the from-scratch learning curve. Completion climbs monotonically to full parity by roughly 300 thousand steps, with no imitation, no critic warm-up, no unfreezing schedule, and no collapse, tracing the same trajectory as the engineered-state run. The image policy that could not learn at all in section 4.6 learns cleanly once the encoder is decoupled from the reward-driven gradient. Cross-track error at parity averages 0.55 m against 0.46 m for the engineered-state policy, a gap consistent with the few-percent prediction error the encoder carries, though the per-seed spread on both runs is wide enough that the two overlap.

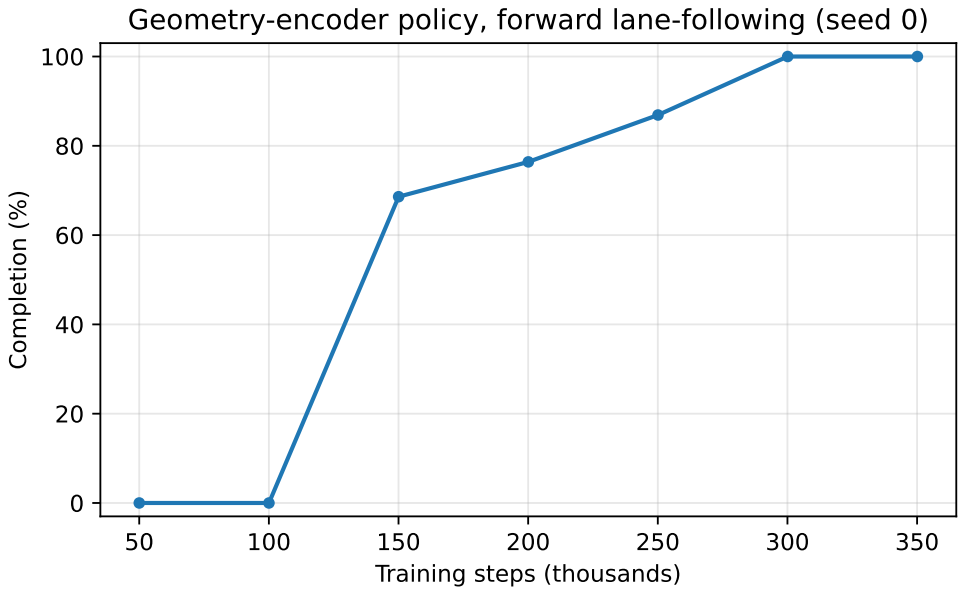


Figure 4.5: Geometry-encoder policy completion against training steps on forward lane-following, trained from scratch on the frozen encoder. Completion reaches parity with the engineered-state baseline by about 300 thousand steps with no collapse.

The single-seed curve is confirmed by the full three-seed set. Forward completion reaches 1.000 on every seed, matching the engineered-state policy trained under the same budget, and the reverse direction reaches 0.954 against that policy's 0.967. Reverse needed a longer budget to get there. At 1.3 million steps it averaged 0.831 with wide seed variance, and extending to two million steps recovered the worst seed from 0.646 to 0.910. The limiter was therefore training convergence rather than the encoder or the replay buffer, which rules out the state-only-replay rework the earlier result appeared to call for.

4.8 Obstacle Navigation

The obstacle task is where the preceding results converge. It is the one setting where the engineered state is genuinely insufficient, because a compact tracking summary is blind to obstacles that are not on the reference path, so the image has to carry information the state cannot. It is also where the value of a learned controller and perception system separates cleanly from classical tracking, because avoiding an obstacle is a joint planning-and-control problem rather than a pure tracking one. The approach taken here is not a monolithic mapping from pixels to actions, which section 4.7 showed does not train. It is a learned but interpretable geometric bottleneck, and the design and results below build it up one decision at a time.

4.8.1 Obstacle Reward and Curriculum Design

The obstacle-navigation task reuses the lane-following reward but adds three design elements, defined in section 4.4.2 and motivated here: a difficulty rating, a stop-gate, and a branching hidden-path reward. The obstacle environment itself was described in chapter 3.

Difficulty as a geometric proxy

The difficulty d of eq. (4.20) is a deliberately cheap, static geometric proxy for drivability, used because evaluating true kinematic feasibility for a non-holonomic tractor-trailer inside the training loop would be far too slow. Measuring only whether the body width fits the gap has two consequences that matter later: every gap at or below the vehicle width collapses to $d = 1$, and, because the proxy ignores the maneuvering and trailer-swing room a real trajectory needs, it labels forward and reverse layouts identically even though reverse needs far more room. Calibrating d into a per-direction drivability threshold, through the oracle of section 4.3, is a result of this section.

Stop-gate

The difficulty that scales the stop reward is computed over only the obstacles that fall inside the observable window ahead of the vehicle, and it is supplied to the policy as an observation channel. Keying the stop reward on this same windowed difficulty makes the value of a stop a deterministic function of what the agent can currently see rather than of a hidden per-episode label, which is what lets the policy learn a clean stop decision. The difficulty channel sets the prize for a correct stop, while perception sets its timing and the progress term credits driving through as much of the layout as possible first. The reverse task needs a further refinement, a state-dependent version of this windowed difficulty that rises as the trailer tracks off the avoidance path, which is developed with its results in section 4.8.5.

Branching hidden-path reward

Rewarding agreement with a single prescribed avoidance corridor would penalize the agent for choosing an equally valid other side. The branching reward of eq. (4.23) removes this by crediting the nearer of two hidden corridors, so the side of passage becomes an emergent decision from perception. This ties the obstacle task to the observation-space study. Choosing the feasible side and knowing when to stop both require seeing far enough ahead.

Curriculum and training protocol

Two further training choices apply across the obstacle experiments. First, obstacle policies are not trained from scratch but through a two-stage curriculum: Stage A trains a lane-follower in the stop-capable action space with no obstacles, and Stage B warm-starts from it with obstacles enabled, so the agent begins obstacle training already able to track a lane and operate the stop signal. The reverse Stage B is itself warm-started from a pre-validated reverse lane-follower rather than trained from scratch, because reverse learning from scratch is high-variance (section 4.4.2). This curriculum is distinct from the failure-replay curriculum, which reshapes scenario sampling within a single training run. Second, the off-policy runs use a learning rate of 10^{-3}, since raising it to 2×10^{-3} diverges once the stop action is present, together with 32 parallel environments, the regime found stable for TD3 on this task.

4.8.2 Direct Obstacle Encoding

The direct approach is to hand the policy the obstacles, as a set of ordered range-and-bearing slots appended to the state. It fails. Trained under the same curriculum, the slot-based policy climbs to a modest completion and then collapses toward zero, and it does so under every slot ordering tried, nearest-first, fixed, and randomized alike. Because a fixed ordering with no reordering collapses just as the others do, the instability is not caused by slots swapping identity between steps. It is the same off-policy training pathology seen in section 4.7, now provoked by a different high-dimensional input. This rules out explicit obstacle encoding and motivates a representation that does not ask the policy to parse a variable set of objects at all.

4.8.3 Observing the Avoidance Path

The approach that succeeds gives the policy a path rather than a scene. The environment already computes a hidden avoidance path, a potential-field offset from the centerline that bends around the obstacles, and the reward already tracks it. Instead of feeding the policy the obstacles, the policy is fed the tracking errors and curvature to this avoidance path. Obstacle avoidance then reduces to path following, the task already solved in the preceding sections, and the obstacle information arrives pre-digested as the path to follow. The image predicts this path, in the same learning-by-cheating manner as the geometry encoder of section 4.7.3, privileged in simulation and image-derived at deployment.

A ground-truth test isolates the mechanism from the perception. Taking an existing forward state policy and feeding it avoidance-path errors in place of centerline errors, with

no retraining at all, cut obstacle collisions roughly in half and lifted completion from about 0.55 to about 0.67. The controller follows whatever path it is given, so the entire obstacle problem can be moved into the choice of path, which is exactly the quantity the image can supply.

4.8.4 The Stop Decision and Forward Results

Following the avoidance path is not enough on its own, because some layouts are not passable and the right action there is to stop rather than to clip an obstacle. The policy is given a windowed difficulty input, the free-gap difficulty of eq. (4.20) computed over the obstacles inside the observable window, on the design set out in section 4.8.1.

In the forward direction this difficulty recipe works well. Table 4.10 reports the outcome stratified by layout difficulty. The policy drives through easy layouts, stops on every hard one, and does not crash on the hardest, for an overall safe rate of 0.907 at a 0.019 crash rate. The residual crash concentrates in the tight-but-passable band where the attempt-versus-stop call is ambiguous. This behavior is what the drivability calibration of section 4.3 predicts. Calibrating the geometric difficulty against the classical drivability oracle through the expected-value balance of eq. (4.2) puts the forward break-even near a difficulty of 0.8, so stopping on the hardest layouts is the rational choice rather than a shortfall.

Table 4.10: Forward obstacle maneuvering, retained static-difficulty model, outcomes stratified by layout difficulty. The policy drives through easy layouts and stops on every hard one with no crash on the hardest, for an overall safe rate of 0.907 at a 0.019 crash rate. The residual crash concentrates in the tight-but-passable band.

Difficulty	Complete (%)	Stop (%)	Crash (%)
$[0.2, 0.4)$ easy	83.1	15.5	1.4
$[0.4, 0.6)$	53.0	40.8	6.3
$[0.6, 0.8)$	0.0	100.0	0.0
$[0.8, 1.0)$ blocked	0.0	100.0	0.0

4.8.5 State-Dependent Difficulty in Reverse

Reverse is the hardest case, and the forward recipe does not transfer to it. Because the reverse dynamics are open-loop unstable, no component tuned forward can be assumed to carry over, and each correction below was rederived for the reverse direction. A static difficulty describes the layout, the free gap, but not the situation, and in reverse the two

come apart. Because reverse tracking runs a systematic off-path error of around 0.8 m, tracking well at one moment does not imply the trailer will clear the coming gap, so a policy gated on layout difficulty over-attempts in the ambiguous band and clips obstacles there, crashing on around forty to forty-five percent of the attempts it makes in that band.

The fix is to make the difficulty state-dependent. The effective vehicle width is inflated by the current trailer error to the avoidance path rather than by an average, so when the trailer is tracking off-path into an obstacle the usable gap shrinks and the difficulty rises in time to trigger a stop. A second correction concerns where the difficulty looks. In reverse the trailer leads, so a window that looked only ahead of the lead point was blind to obstacles the trailing body was still clearing, which produced a spurious cluster of crashes at apparently low difficulty. Anchoring the window to the whole vehicle body removed it. This is a reverse-specific need, as the collision geometry in table 4.11 makes plain. Reverse crashes are overwhelmingly trailer clips, the leading trailer sweeping into an obstacle, whereas forward crashes are entirely tractor clips.

Table 4.11: Collision geometry by direction. Reverse crashes are dominated by trailer clips, which the body-extent difficulty window addresses, while forward crashes are entirely tractor clips, so the window fix is a reverse-only need.

Direction	Tractor clip (fraction)	Trailer clip (fraction)
Forward	1.00	0.00
Reverse (mean of 3 seeds)	0.18	0.82

Once state-dependent dynamic difficulty was defined, the threshold for a correct stop decision was calibrated using difficulty-blind controllers: the state-based TD3 policy and the pure-pursuit controller. These controllers were evaluated on the held-out path pool without access to the difficulty measure, and the dynamic difficulty was recorded at each collision. No collisions occurred below a dynamic difficulty of approximately 0.5. This value was therefore used as the decision boundary when training the difficulty-aware policy. Attempts were rewarded for observed difficulties below 0.5, while stopping was rewarded for observed difficulties at or above 0.5.

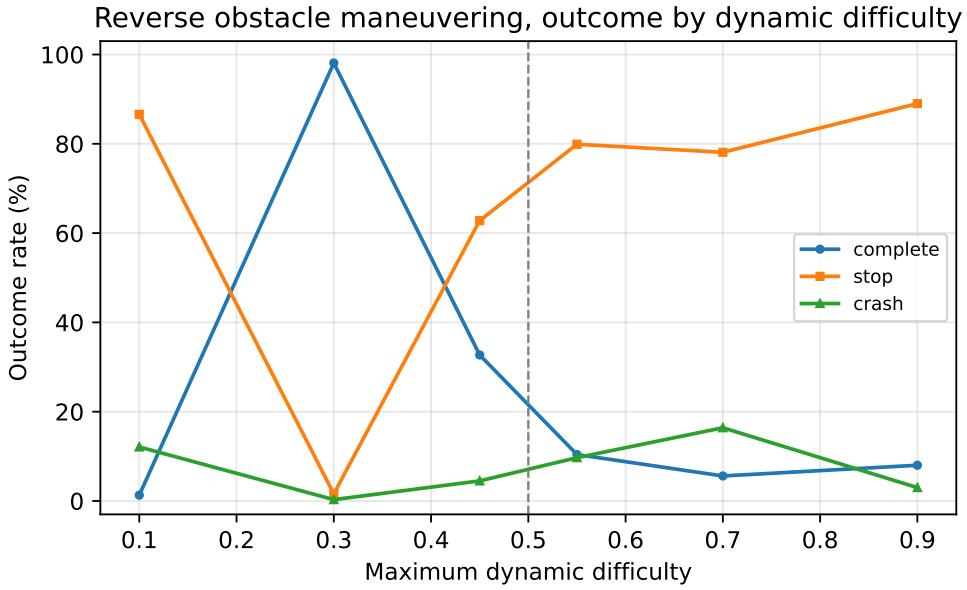


Figure 4.6: Reverse obstacle maneuvering, outcome rates against maximum dynamic difficulty.

Table 4.12: Reverse obstacle maneuvering, accepted dynamic-difficulty model, outcomes stratified by the maximum dynamic difficulty reached. Completion falls and stopping rises with difficulty, and no crash occurs below the gate at roughly 0.5. The low-difficulty stops are the accepted articulation limitation, not spurious behavior.

Dynamic difficulty	n	Complete (%)	Stop (%)	Crash (%)	Stop $\lvert\gamma\rvert$ (rad)
$[0.0, 0.2)$	762	1.3	86.6	12.1	0.204
$[0.2, 0.4)$	2239	98.1	1.6	0.3	0.105
$[0.4, 0.5)$	584	32.7	62.8	4.5	0.039
$[0.5, 0.6)$	1160	10.4	79.9	9.7	0.101
$[0.6, 0.8)$	1217	5.6	78.1	16.4	0.119
$[0.8, 1.0)$	5778	8.0	89.0	3.0	0.062

The reverse recipe is trained from scratch and repeated across three seeds in table 4.13. Two of the three reproduce the safe result at a crash rate around 0.05, while one seed converged to a less conservative policy that attempts more of the transition band and crashes more there. The spread traces to a critic shock at the boundary where the reward changes from the obstacle-free warm-up to the obstacle task, so the recipe is usable but not tightly reproducible. A critic warm-up at that boundary and a safety-aware checkpoint criterion

are the levers to tighten it. The same table records a second finding. Dynamic difficulty helps reverse but hurts forward, where the vehicle does not jackknife and its tracking error is small, so the width inflation only adds noise. Forward therefore keeps the static recipe, and a single unified stop signal for both directions is not achievable given their different dynamics.

Table 4.13: Reverse dynamic-difficulty model across three seeds trained from scratch. Two of three seeds reproduce the safe result, and seed 1 converged to a less conservative policy.

Seed	Complete (%)	Stop (%)	Crash (%)
Seed 0	26.0	68.8	5.2
Seed 1	36.8	49.7	13.5
Seed 2	32.8	62.3	5.0
Crash mean ± sd			$7.9_{\pm 4.9}$

Table 4.14: Forward crash rate, static against dynamic difficulty. Dynamic difficulty hurts forward, so forward keeps the static recipe.

Forward recipe	Crash (%)
Static difficulty	2.7
Dynamic difficulty (mean of 3 seeds)	8.7

4.8.6 Recovering the Difficulty from the Image

The obstacle recipe gates on the dynamic difficulty, which depends on the current trailer error and so is a harder image target than the static path geometry of section 4.7.5. The same distillation recovers it. Table 4.15 reports the per-component recoverability of the avoidance-path state from the obstacle image in both directions, and the dynamic difficulty channel is recovered well, at a test R^2 of 0.948 forward and 0.954 reverse. The avoidance-path errors are recovered strongly, while the avoidance-path curvature is softer than in the lane task and softer still in reverse, where the bends are sharper, though the errors and the difficulty channel carry the control.

Table 4.15: Recoverability of the avoidance-path state from the obstacle image, test R^2 per component for the forward static and reverse dynamic recipes. The dynamic difficulty channel is highly recoverable in both directions, so there is no perception roadblock. Reverse avoidance-path curvature is softer than forward.

Component	Forward R^2	Reverse R^2
e_y	0.957	0.980
e_ψ	0.932	0.943
$e_{y,t}$	0.981	0.977
$e_{\psi,t}$	0.991	0.936
κ_1	0.965	0.726
κ_2	0.903	0.639
d_{win}	0.948	0.959

Closing the loop confirms there is no perception roadblock. Table 4.16 drives each policy on its own image-predicted state, with a frozen encoder in the loop and only the steering and hitch angles taken from proprioception. The full pipeline holds in both directions, with the crash rate sitting a few points above the ground-truth-state policy from ordinary prediction noise, a larger margin in reverse where the curvature is harder to see. The image supplies both the plan, the avoidance path and its difficulty, and the control state, which is the claim the whole perception thread of this chapter was built to support.

Table 4.16: Closed-loop vision pipeline, policy driven on image-predicted avoidance-path state with a frozen encoder in the loop. The full pixels-to-control pipeline holds in both directions, a few points of crash above the ground-truth-state policy from prediction noise.

Recipe	Difficulty R^2	SAFE (%)	Crash (%)
Forward (static)	0.948	90.3	4.1
Reverse (dynamic)	0.954	74.9	8.3

4.8.7 Deterministic Safety Gate

The conservative stops seen above are not spurious behavior. On predominantly straight reverse paths the policy occasionally halts where a continued maneuver would have completed, and at those halts the trailer's hitch articulation is already degrading, with a terminal articulation angle around 0.20 rad against about 0.01 rad on completed paths, trending toward

the unrecoverable regime near 0.71 rad that produces the collisions. A reversing tractor-trailer cannot unwind a growing hitch angle by continuing in reverse, because the reverse articulation dynamics are unstable and more reverse motion only grows the hitch. The geometrically required recovery is to pull forward, re-stabilize the articulation on the stable forward dynamics, and re-approach, which needs a forward-reverse switching action space. Within a reverse-only action space, halting is therefore the correct and safe response, and pull-forward recovery is left to future work.

Rather than rely on the policy erring conservative, the same protection can be made explicit and controller-independent with a deterministic safety gate. The gate stops the vehicle before it commits to a maneuver whenever it is in a bad enough state, using two thresholds for the two ways a reverse maneuver crashes. It halts if the peak dynamic difficulty exceeds 0.375, which catches the obstacle clips where the vehicle tracks well but the gap is too tight, and it halts if the peak hitch angle exceeds 0.6 rad, which catches the articulation runaways. The hitch threshold sits above the 99th percentile of successful maneuvers, about 0.40 rad, and below the articulation-crash floor near 0.71 rad, so it catches every articulation crash while rarely stopping a maneuver that would have completed. Applied after training with no retraining, on the reference seed the gate reduces the reverse crash rate from 0.052 to 0.0004, about five episodes in 11,740.

This gate is what sets the learned and classical controllers apart, and then brings them back together. Without any deterministic gate the learned attempt-versus-abort decision separates the two controllers sharply, because pure pursuit has no way to decline an infeasible layout and crashes on 54 percent of the hard-layout-dominated pool, against about 5 percent for the learned policy. Once the deterministic gate is layered on top, the two converge, with the crash rate driven to near zero for both and comparable completion, as Table 4.17 shows. The learned stop decision is therefore what makes the policy safe when no hand-designed gate is available, and the deterministic gate is an added safety layer that a classical controller can equally use.

Table 4.17: Reverse obstacle maneuvering with and without the deterministic safety gate, learned policy against pure pursuit on the same hard-layout-dominated pool. Without a gate the learned attempt-versus-abort decision already keeps the learned policy far safer than pure pursuit, which cannot decline a layout. Adding the gate drives the crash rate to near zero for both and leaves their completion comparable. Reference seed. Commit-success is completions over completions plus crashes.

Controller	Crash, native stop no gate (%)	Crash, with gate (%)	Completion, with gate (%)	Commit success (%)
Learned (RL)	5.2	0.0	17.7	99.8
Pure pursuit	54.0	0.2	18.1	99.0

Together, these results explain why a learned controller is warranted here even though a classical planner exists. Its value is the learned attempt-versus-abort stop decision, which a geometric planner does not make, together with the reverse articulated case, where a geometric avoidance path is not by itself trackable and the stop must be tied to the articulation state. Both live in one learned module that reads its plan and its control state from the same image, in place of the separate global planner, local planner, and tracker of the classical stack. The deterministic gate does not displace that value, it adds a controller-independent safety floor on top of it.

4.9 Comparison with Classical Controllers

A learned controller has to be shown competent at plain tracking before its obstacle behavior means anything, so this section places it beside tuned classical control on the no-obstacle task. The framing matters. On pure path tracking a classical controller is complete and verifiable, and for that task it is the sensible choice, so the aim here is not to beat it but to establish that the learned controller reaches comparable tracking competence. The classical controllers were tuned by differential evolution and validated on held-out seeds, and every controller faces the same pool as the learned agent. Tables 4.18 and 4.19 report the comparison.

Table 4.18: Forward lane-following controller benchmark on the evaluation pool. Completion is saturated for every method, so it cannot separate them and tracking error does. TD3 is reported over three seeds at the extended 300M-step training budget.

Controller	Completion (%)	Trailer CTE (m)	Max $\lvert\gamma\rvert$ (°)
TD3 (state, multiplicative)	$100.0_{\pm 0.0}$	$0.176_{\pm 0.011}$	5.5
Pure pursuit	100.0	0.151	5.2
PID	100.0	0.165	5.4
LQR	100.0	0.266	6.0
MPC	100.0	0.285	6.3

Table 4.19: Reverse lane-following controller benchmark on the evaluation pool. The learned policy matches pure pursuit on completion to within the seed interval and tracks the path more tightly than every classical controller, including LQR, the strongest classical tracker here. TD3 is reported over three seeds at the extended 300M-step training budget.

Controller	Completion (%)	Trailer CTE (m)	Max $\lvert\gamma\rvert$ (°)
TD3 (state, multiplicative)	$97.5_{\pm 0.8}$	$0.499_{\pm 0.042}$	9.3
Pure pursuit	98.0	0.875	7.3
PID	87.3	0.852	15.7
LQR	95.9	0.619	10.1
MPC	97.7	0.845	8.1

Forward tracking is saturated, so completion cannot separate the methods and the comparison there is only about tracking quality. Reverse is more informative. The learned policy reaches completion comparable to pure pursuit and ahead of PID, which establishes the competence the obstacle results depend on, a policy that can actually stabilize and track the trailer on the unstable reverse task. On the unfiltered pool pure pursuit completes 98.0% of the reverse layouts and the learned policy 97.5%, a difference that sits inside the seed interval. The case for the learned controller does not rest on this tracking comparison. It rests on the integrated obstacle task of section 4.8, where pure pursuit alone is not a controller and the classical alternative is the full planner-and-tracker stack. In articulated control, success is not simply the lowest cross-track error, because hitch stability takes precedence over exact path following. A policy that holds the hitch angle inside a safe margin throughout a reverse maneuver is more successful than one that shaves cross-track error but risks jackknifing,

which is why the benchmark reports completion, trailer tracking, and articulation stability together rather than collapsing them into one score.

4.9.1 Information Parity between Learned and Classical Controllers

A controller comparison is only meaningful if the methods are given comparable information about the reference path. Table 4.20 summarizes what each method observes at a single control step on the no-obstacle task. The TD3 observation contains the instantaneous tractor and trailer tracking errors together with two path-curvature samples taken ahead of the trailer axle. This is a strict superset of the reference information used by pure pursuit and PID, which close the loop on the same instantaneous errors plus a single curvature sample at one lookahead point. The learned policy is therefore not at an information disadvantage, so the comparison against pure pursuit and PID is a like-for-like test of control quality.

Table 4.20: Reference-path information available to each method at a single control step on the no-obstacle task. The learned policy's observation is a strict superset of the information used by the pure-pursuit and PID baselines. Only MPC receives an explicit preview of the path ahead.

Method	Instantaneous errors (e_y, e_ψ)	Curvature lookahead	Full path preview
Pure pursuit	yes	1 pt ($\sim$10 m)	no
PID	yes	1 pt ($\sim$10 m)	no
TD3 (state/LiDAR/BEV)	yes	2 pts ($\sim$10, 20 m)	no
MPC	implicit	no	N=16/20 steps

MPC is the exception, and it is best read as an informed upper bound rather than a peer. It is supplied with an explicit preview of the path, a full reference trajectory over a receding horizon reconstructed from the same centerline the policy tracks, while the learned policy receives only the two scalar curvature samples. MPC is information-advantaged because it receives a full receding-horizon path preview, whereas the learned policy receives only two curvature samples. Despite this advantage, TD3 achieves lower trailer cross-track error in the reported reverse benchmark. The pure-pursuit and PID comparisons are the information-matched test of whether a learned policy can equal classical feedback control on the articulated tracking task.

For obstacle scenarios the comparison is not like-for-like in the same way, because the

learned policy solves a joint local-planning and control problem, including the stop decision, whereas the classical controllers are path trackers.

4.10 Failure Modes and Deployment Relevance

Average metrics alone are insufficient for articulated-vehicle control, because the operational significance of a failure depends on how it occurs. The seed collapses behind the wide confidence intervals of section 4.6 are the clearest case, since they are not a uniform degradation but a bimodal outcome in which a policy either learns the task or falls into a degenerate behavior, which a mean completion rate hides entirely.

This reading is what connects the simulation study to deployment. A controller that attains a low mean cross-track error but exhibits rare catastrophic jackknife episodes is less deployable than one with slightly higher average error and consistent stability margins, which is why the reverse results are read through hitch stability rather than cross-track error alone. Overall, the results position the learned controller as a credible peer to classical feedback control on the unstable reverse task, and the geometry-encoder recipe as the piece that makes high-dimensional perception trainable rather than merely available, which is what the obstacle task and eventual hardware deployment in chapter 5 require.

4.11 Summary

This chapter settled the learning recipe and then established what it can do. On the engineered state, TD3 under a multiplicative reward is the only combination that solves the unstable reverse task reliably, which fixes the algorithm and the reward for everything that follows. Opening the observation up reverses the intuition that more sensing is better: reliability falls as the observation grows, and the raw image cannot be learned from at all under the recipe that solves the state task. That failure proved to be an optimization pathology of off-policy actor-critic learning rather than a limit of perception, isolated by a ladder in which representation-side fixes change nothing and only freezing the actor rescues a cloned policy. Distilling the image into the path geometry with a supervised, frozen encoder removes the pathology at its root and restores ordinary from-scratch learning, and the recoverability measurements confirm that the image really does encode the geometry the engineered state summarizes. On the obstacle task, where the engineered state is genuinely insufficient, the same bottleneck carries both the plan and the control state: the policy follows an image-inferred avoidance path and gates its attempt-versus-stop decision on a windowed difficulty, which needs to be made state-dependent in reverse because tracking well at one moment does

not imply the trailer will clear the coming gap. Against tuned classical control the learned policy reaches comparable tracking competence, and its distinctive value is the learned decision to decline an infeasible layout, which a geometric planner does not make. Carrying that controller onto a physical platform is the subject of chapter 5.

Chapter 5

Sim-to-Real Deployment

5.1 Overview

The preceding chapters developed and evaluated an articulated tractor-trailer controller entirely within a 2D simulation environment. This chapter describes the transfer of that controller to a physical robotic platform, constituting the sim-to-real component of this thesis. Unlike a conventional model-based stack, the deployed system does not re-implement a hand-derived tracking law on the vehicle. Instead, the simulation-trained reinforcement-learning (RL) policies are executed directly on hardware through a thin bridging layer that reproduces the observation and action interfaces the policies were trained against. The objective is to demonstrate that the vehicle model, the observation definitions, and the learned control logic established in simulation can be ported to real hardware with minimal algorithmic redesign, and to characterize the engineering challenges encountered during that transfer.

The deployment is built on a deliberately stripped-down Autoware stack. The standard Autoware localization and perception pipelines are retained essentially unchanged, while the behavior, planning, and control layers are replaced by a custom RL bridge that runs the forward and reverse policies. Two pieces of supporting infrastructure carry the deployment alongside it, an onboard LiDAR-based hitch-angle estimator that recovers the articulation angle without any trailer-mounted sensor, developed by S. Huang and documented in chapter B, and a Smith predictor [79] on the state presented to the policy that compensates for command latency.



Figure 5.1: The physical articulated platform: an AgileX Hunter SE tractor coupled to the lab trailer through a revolute hitch at the tractor rear axle.

5.2 Hardware Platform

Tractor Vehicle

The physical tractor is an AgileX Hunter SE, a compact electric wheeled unmanned ground vehicle with an Ackermann steering geometry. The relevant mechanical parameters are a wheelbase of 0.65 m, a wheel radius of 0.15 m, and a maximum steering angle of 0.436 rad (25°). Vehicle actuation is commanded over a CAN bus on a 20 ms cycle. Speed commands are encoded as 16-bit signed integers in units of 1 mm/s (the commanded speed in m/s scaled by 1000) and steering commands as 16-bit signed integers scaled by approximately 1320 counts per radian (≈ 0.8 mrad per count).

Trailer

The trailer is coupled to the tractor through a revolute hitch joint located marginally behind the tractor rear axle, which was added to the tractor-trailer model used for training. The lab trailer has a wheelbase of approximately 2.8 m and a body width of roughly 0.33 m. The articulation angle γ, defined as the difference between the tractor and trailer headings ($\gamma = \psi_{\text{truck}} - \psi_{\text{trailer}}$), is estimated onboard from the tractor's own LiDAR rather than measured by

any trailer-mounted sensor. The procedure is detailed in chapter B. The estimate is published to the ROS2 network on the `/vehicle/trailer_state` topic. The control bridge consumes the most recent published value, with no staleness gating currently applied. Two distinct angular limits appear in the stack and should not be conflated. The onboard estimator clamps the published articulation estimate to $|\gamma| \leq 90°$, whereas the deployment trailer model defines a tighter operational limit of $|\gamma| \leq 80°$ (1.40 rad), itself below the 90° jackknife condition used to terminate simulation episodes.

Sensor Suite

The vehicle is instrumented with the following sensors:

- **LiDAR**: RoboSense Helios mechanical LiDAR mounted at the front of the tractor, publishing 3D point clouds on `/rslidar_points`. This single sensor is reused for three purposes: NDT-based localization [80], environment perception, and onboard hitch-angle estimation.
- **Cameras**: Six Basler Pylon industrial cameras (rear, rear-right, center, left, merging, and right) producing compressed color images. The cameras are not consumed by the LiDAR-based RL policies and are used for logging and monitoring only.
- **IMU**: WitMotion 6-DoF inertial measurement unit, with a gyroscope-bias estimator applied in the localization pipeline.
- **GNSS**: u-blox ZED-F9P receiver with RTK corrections, providing centimeter-level position estimates converted to MGRS coordinates for Autoware localization.
- **Wheel odometry**: Vehicle speed and steering-angle feedback recovered from CAN messages and fused with IMU data for dead-reckoning between GNSS and scan-matching updates.

5.3 Software Architecture

5.3.1 Middleware and Framework

The deployment stack runs on ROS2 Humble (Ubuntu 22.04) [81] and is integrated into the Autoware autonomous-driving framework [82]. Autoware provides modular localization, perception, planning, and control pipelines whose interfaces are defined by standardized ROS2 message types. As introduced in the overview, this deployment keeps the standard

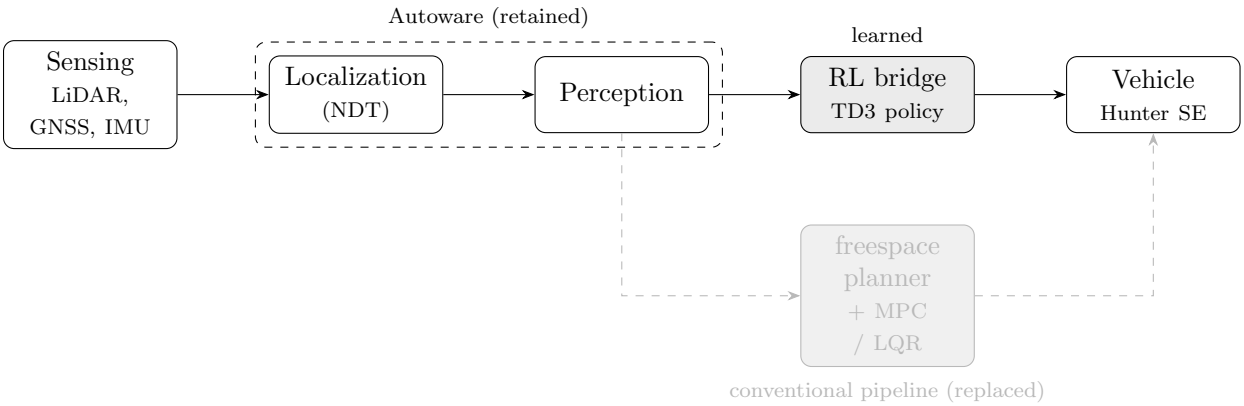


Figure 5.2: Deployment software stack. Autoware's localization and perception are retained unchanged, while the planning and control layers (the conventional freespace planner and MPC/LQR tracker) are replaced by the learned RL bridge. The two control paths are mutually exclusive and selected at launch.

sensing, localization, and perception modules and replaces only the planning and control layers with the learned RL bridge.

5.3.2 Relationship to the Conventional Articulated Pipeline

For context, Autoware's conventional approach to articulated vehicles is fully model-based. A truck-trailer mode manager routes goals to a freespace planner (a hybrid A* search [20] that respects the non-holonomic constraints of the articulated vehicle), and the resulting reference trajectory is tracked by a model-predictive or LQR lateral controller. This model-based pipeline exists within the wider project and represents the standard, well-understood baseline for trailer control. The relevant control background is reviewed in Chapter 2. The contribution of this thesis, however, is the learned controller, and the deployment described here therefore routes control through the RL bridge instead of the planner/MPC chain. The two are mutually exclusive at run time and are selected at launch. Figure 5.2 shows which modules are retained and which are replaced.

5.3.3 RL Bridge Node Architecture

The learned control path comprises the following principal nodes:

`lane_reference_node` Loads the Lanelet2 map, selects the active lane from the ego odometry and the current goal, and publishes the lane centerline as a reference path together with a `drive_enabled` flag and a `drive_direction` flag (forward or reverse). This

node supplies the path geometry against which the policy's tracking observation is computed.

`rl_bridge_node` The core of the deployment. It loads the trained forward and reverse TD3 policies (Stable-Baselines3 [69] / PyTorch [83] checkpoints) and runs a 10 Hz control loop. On each tick it assembles the policy observation in exactly the layout used during training, supporting all three state, LiDAR, and BEV observation variants, queries the active policy deterministically, and converts its 2-dimensional action (a steering rate and a stop/go signal) into the steering-angle and speed commands expected by the vehicle interface, with the stop/go signal mapped onto the task's fixed speed or a halt exactly as the simulation interface layer does during training. Direction-dependent post-processing (action smoothing and kinematic rate scaling) is applied for the reverse policy, whose outputs are inherently more oscillatory than the forward policy's.

`hitch_angle_estimator` Estimates the articulation angle from the tractor LiDAR by rectangle fitting and Kalman filtering, and publishes it on `/vehicle/trailer_state`. This node was developed by S. Huang, and its method is documented in chapter B.

`initialpose_shim` Bridges the RViz `/initialpose` message to the Autoware pose-initializer service, allowing the operator to seed localization interactively.

`controller_canbridge` Translates `autoware_control_msgs/Control` commands into Hunter SE CAN frames and parses vehicle feedback (velocity, steering angle) back into ROS2 topics.

The forward and reverse policies are loaded from separate checkpoints and selected according to the `drive_direction` flag, so that the same bridge handles both maneuvering regimes without retraining or reconfiguration.

Figure 5.3 shows how these nodes connect.

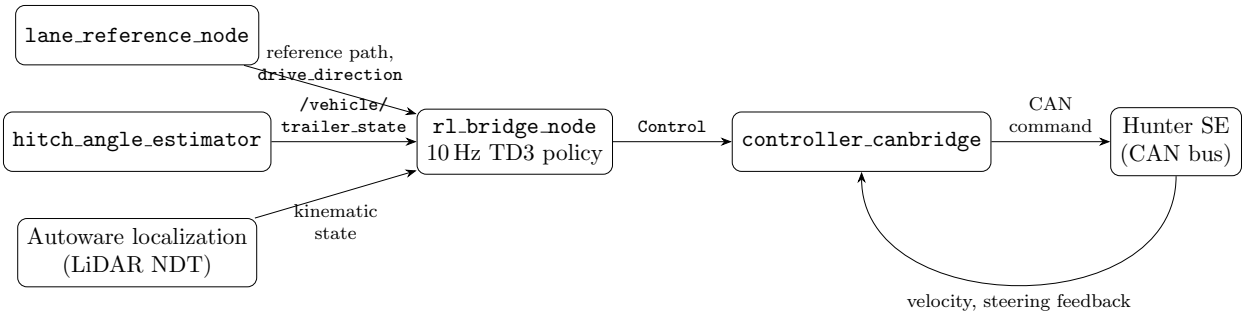


Figure 5.3: RL bridge dataflow. The `lane_reference_node`, the LiDAR-based `hitch_angle_estimator`, and Autoware localization supply the policy observation to `rl_bridge_node`, which queries the TD3 policy at 10 Hz and emits a control command that `controller_canbridge` translates into Hunter SE CAN frames. Vehicle velocity and steering feedback return through the same bridge.

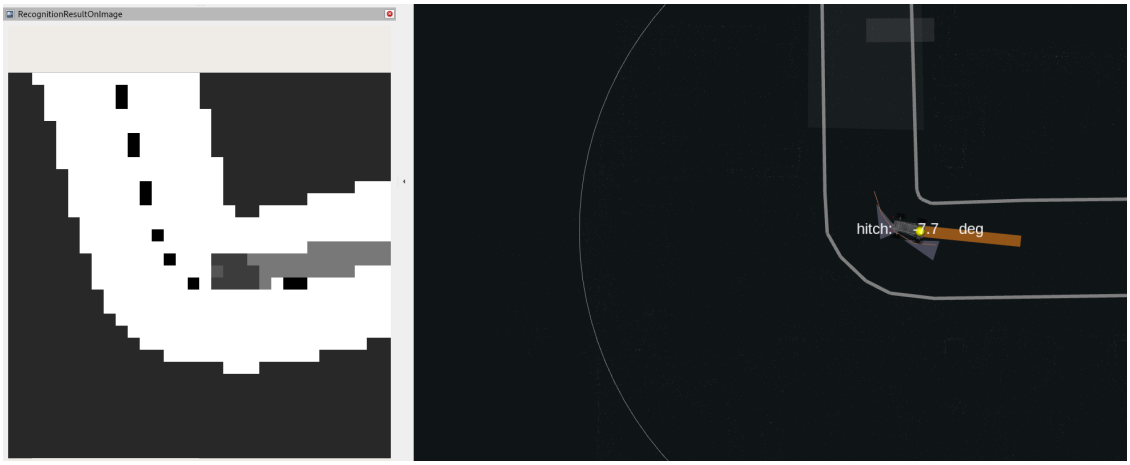


Figure 5.4: The deployment running in RViz. The left panel shows the occupancy image the policy consumes, drivable in white and blocked in black, built from the lanelet map and the infrastructure obstacle list. The right panel shows the same scene in the map frame, with the tractor and trailer, the LiDAR return field, and the estimated hitch angle annotated on the vehicle.

5.3.4 Trailer State Estimation

Because the trailer carries no independent localization, its full pose is reconstructed from the tractor's localized kinematic state and the estimated articulation angle. Given the tractor's rear-axle position and heading ($\mathbf{p}_{\text{truck}}$, ψ_{truck}) from `/localization/kinematic_state`, the hitch-point position is

$$\mathbf{p}_{\text{hitch}} = \mathbf{p}_{\text{truck}} + d_h \begin{bmatrix} \cos\psi_{\text{truck}} \\ \sin\psi_{\text{truck}} \end{bmatrix}, \tag{5.1}$$

where d_h is the (small, near-zero) signed hitch offset relative to the tractor rear axle. The trailer heading follows directly from the articulation angle, $\psi_{\text{trailer}} = \psi_{\text{truck}} - \gamma$, and the trailer rear axle is located at

$$\mathbf{p}_{\text{trailer}} = \mathbf{p}_{\text{hitch}} - L_t \begin{bmatrix} \cos\psi_{\text{trailer}} \\ \sin\psi_{\text{trailer}} \end{bmatrix}, \tag{5.2}$$

where $L_t \approx 2.8\,\text{m}$ is the trailer wheelbase. This reconstruction mirrors the hitch-kinematics model derived in Chapter 3, confirming that the simulation formulation is directly reusable in the deployment stack, and it provides the trailer-referenced tracking errors that form part of the policy observation.

5.4 Sim-to-Real Gap Analysis

Transferring a simulation-trained policy to physical hardware introduces discrepancies between the simulated and real-world dynamics. The following gap sources were identified and addressed during the deployment process.

Vehicle Scale and Parameterization

The most fundamental difference between the simulation studies and the physical platform is one of scale. The simulation of chapter 3 is parameterized for a full-size shunt truck (table 3.1), whereas the deployment vehicle is the compact AgileX Hunter SE (wheelbase 0.65 m, maximum steering angle 0.436 rad), with under a quarter of the wheelbase and half the steering range of the reference terminal tractor. Rather than transferring the shunt-truck-scale policy onto the smaller vehicle and relying on it to generalize across scale, the deployment closes this gap by construction. The model structure of chapter 3 is retained unchanged and re-parameterized to the Hunter's wheelbase, steering limit, and speed range, and a fresh policy is trained against that re-parameterized model. Because the governing equations are identical and only the parameter set changes, the observation and action interfaces the policy was trained against are preserved, and the scale mismatch is removed before deployment rather than being left as a residual gap to be compensated at run time.

Command Latency and Actuation Dynamics

The simulation model assumes an instantaneous steering response, whereas the deployed loop responds with a measurable delay. A bidirectional step-response test on the stationary vehicle, with the steering status logged at 50 Hz, separates that delay into a transport dead time of approximately 0.09 s and a first-order actuator lag of approximately 0.11 s, alongside

a steering rate limit of 1.18 rad/s and unity steady-state gain. The dead time is digital rather than mechanical, the cost of a networked command path through the CAN link and the discrete control loop, and it is of the same order as the 10 Hz policy period, so a command is already about one control tick old when it reaches the actuator. On a production vehicle with an embedded steering controller this contribution is negligible and is rightly omitted from a vehicle-dynamics model; on a research platform whose control loop closes through ROS2 it is the highest-impact discrepancy for a policy trained on instantaneous actuation. Training deliberately contains no such delay, so it is reconciled at deployment rather than learned: the Smith-predictor compensator presents a delay-compensated state to the policy, and a steering-rate limit consistent with the $\dot{\delta}_{\max}$ constraint used during simulation training is applied to the commanded output.

Hitch-Angle Sensing

In simulation, the articulation angle γ is available exactly from the vehicle state. On hardware it is estimated from LiDAR returns and is therefore subject to noise from sparse or partial returns on the trailer face, mis-segmentation of the region of interest, and momentary loss of the face at large articulation. These error modes are addressed within the estimator itself rather than by a downstream filter. The rectangle-fitting formulation of chapter B is robust to partial face visibility, and the constant-angular-velocity Kalman filter with innovation gating rejects outlier scans and smooths the published signal. If the trailer face is lost entirely the estimator withholds new estimates, and the bridge continues from the last published value until the face is reacquired. No automatic disengagement on stale data is currently implemented.

Path Distribution

The policies are trained on smooth, spline-generated reference paths, whereas the deployment reference is the centerline of a Lanelet2 lane, which is piecewise-linear and may contain comparatively sharp vertices. Re-sampling the centerline does not fully reconcile the difference in path-curvature distribution between training and deployment, and this mismatch is a contributing factor to any residual tracking error observed on hardware. It is noted here as a characterization of the gap rather than something fully eliminated in the current deployment.

Jackknife Prevention

The simulation introduced a terminal episode condition when $|\gamma| > 90°$ but provided no explicit recovery mechanism, since the episode simply resets. To account for this, a recovery-scenario reset distribution spawns a fraction of reverse episodes already in the near-jackknife region (articulation magnitudes of roughly 0.35–0.80 rad, i.e. 20–45°), so the policy is repeatedly exposed to high-articulation states its nominal training distribution rarely visits and must learn to steer the trailer back toward alignment. Additionally, an anti-jackknife override is applied in the RL bridge. Whenever the articulation exceeds approximately 0.7 rad (40°), the policy's steering command is replaced by a maximum counter-steer that drives the articulation back toward zero.

Localization Dependency

The simulation provides ground-truth vehicle pose at every step. The hardware stack instead depends on the Autoware localization pipeline, which fuses RTK-GNSS, gyro-based odometry, and LiDAR NDT scan matching through an extended Kalman filter. GNSS outages or localization divergence therefore represent a failure mode that does not exist in simulation. The current deployment treats localization failure as an emergency-stop condition.

5.5 Infrastructure Sensor Nodes

A structural limitation of the onboard sensing arrangement is that the trailer occludes the region directly behind it. The tractor-mounted LiDAR cannot see past the trailer, and instrumenting the trailer itself is undesirable. It would add cabling, power, calibration, and a wireless link to an otherwise passive, interchangeable body, and would have to be repeated for every trailer. This is most acute during reverse maneuvering, precisely when visibility of the space behind the trailer matters most.

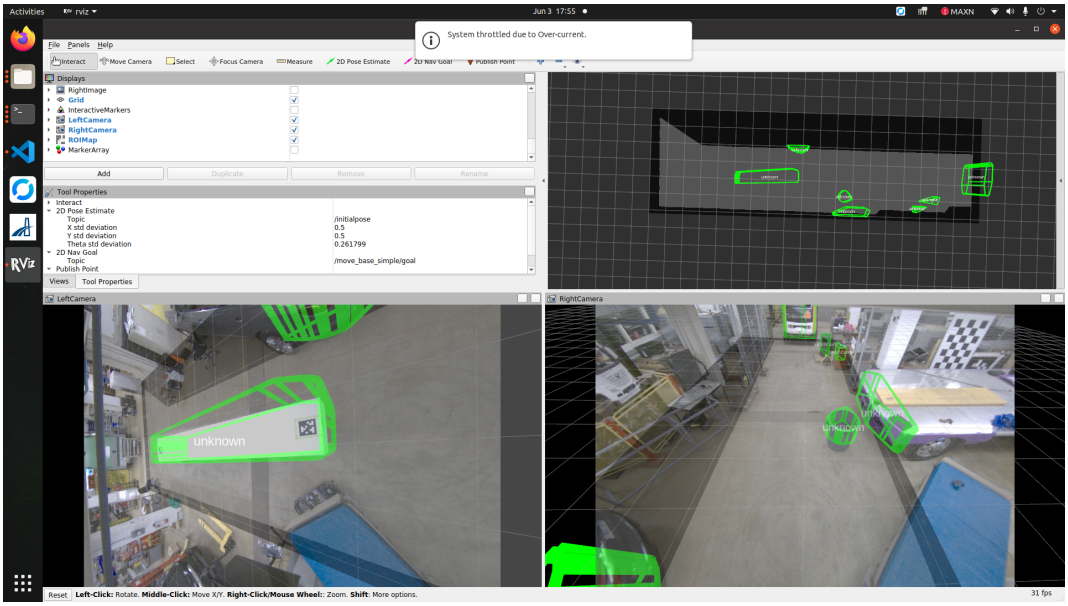
This limitation is addressed with off-vehicle infrastructure sensor nodes, fixed LiDAR units placed at surveyed vantage points around the operating area, each providing a bird's-eye view of obstacles (including the region behind the trailer) without any sensor being mounted on the trailer. Each node is calibrated into a shared global frame so that its detections can be expressed in the same coordinates as the onboard perception output and fused into Autoware's perception and occupancy-grid representation, yielding a persistent, occlusion-resilient view of the obstacles around the articulated vehicle that complements the limited onboard field of view.

This capability is implemented and running in the laboratory as the indoor-perception

stack. The infrastructure sensing system described here was developed separately by Cao and colleagues in the laboratory [84], and this thesis discusses only its integration boundary with the ROS2, Autoware, and RL-bridge deployment stack. Each infrastructure node runs Euclidean clustering on its own LiDAR returns to detect candidate objects. An offboard fusion pipeline transforms every node's detections into the shared global frame, merges detections that refer to the same physical object across nodes by their overlapping footprint polygons, and time-aligns the streams against each node's modeled latency before passing the merged detections to an extended-Kalman-filter multi-object tracker. The tracker emits a single stable, deduplicated obstacle list for the whole monitored area.

Because the laboratory sensor nodes and their perception stack run under ROS1 (Noetic) while the vehicle runs under ROS2 (Humble), the two are joined by a dedicated interface node. It subscribes to the ROS1 perception outputs and republishes the fused obstacle list over ROS2, where it is consumed by the vehicle's Autoware perception and occupancy-grid representation and thus reaches the RL bridge. Critically, this interface node runs offboard on the laboratory compute rather than on the vehicle. The vehicle's onboard Jetson therefore runs only ROS2, and never has to host a dual ROS1/ROS2 stack. The ROS1 dependency is confined entirely to the off-vehicle infrastructure, which both simplifies the embedded software image and keeps the version-sensitive ROS1 perception packages off the resource-constrained onboard computer.

In laboratory operation, the infrastructure nodes recover the region directly behind the trailer that the onboard LiDAR cannot observe.



5.6 Deployment Results

5.6.1 Trial Protocol

The deployment was exercised on the laboratory site as a point-to-point lane-following maneuver run in both directions, from a fixed start pose A to a fixed goal pose B along a surveyed lane centerline of roughly 23 m published in the map frame by the lane-reference node. The BEV with geometry encoder policy ran through the bridge at 10 Hz with the articulation estimate supplied by the onboard LiDAR hitch estimator of chapter B, and the bridge commanded a constant speed of 0.59 m/s forward and 0.25 m/s in reverse, so the policy controlled steering rate alone. Four moving trials were recorded, two in each direction, and all four reached point B under policy control.

Each trial was recorded to a rosbag carrying the localization estimate, the articulation estimate, the steering feedback, the issued control command, the reference centerline with its drive-enable and direction flags, and the bridge's own observation vector and raw policy action. Statistics are reported over the policy-controlled segment of each trial, the ticks where the lane-reference node reports drive-enabled in the intended direction and the bridge is commanding a non-zero velocity in that direction. The reported errors are geometric and recomputed offline from the localization pose and the trailer reconstruction of section 5.3.4, rather than read back from the policy input, which has passed through the Smith predictor and a frame mirroring step and is therefore not a measurement of where the vehicle actually was. Cross-track error is the signed perpendicular offset from the reference point to the reference polyline, and heading error is the body heading relative to the path tangent taken along the direction of travel. The offline reconstruction agrees with the bridge's own logged observation to a magnitude correlation of at least 0.98 on articulation, tractor error, and trailer error.

5.6.2 Tracking Performance

Table 5.1: Lane following on the physical platform, the four A-to-B trials recorded on 2026-08-05, two in each direction, over the same surveyed lane segment. Statistics cover the policy-controlled segment of each trial and are computed geometrically from the localization estimate against the reference centerline, independently of the policy's own observation. The final column is the fraction of control ticks at which the commanded steering rate sat on its saturation limit.

Trial	Driven (m)	Speed (m/s)	Tractor CTE (m)		Trailer CTE (m)		$\lvert\gamma\rvert$ (deg)		$\dot{\delta}$ sat (%)
			mean	max	mean	max	mean	max	
Forward									
F1 (10:07:56)	23.6	0.59	0.045	0.10	0.223	1.05	9.1	38.7	0
F2 (10:18:02)	23.6	0.59	0.078	0.16	0.263	1.03	9.4	39.0	0
Reverse									
R1 (12:09:51)	22.6	0.25	0.964	2.39	0.687	1.91	20.9	51.6	80
R2 (12:45:49)	22.6	0.25	1.102	2.35	0.764	1.86	21.8	51.1	79

Forward tracking transferred essentially intact and was consistent across the two trials (table 5.1). The tractor held a mean cross-track error of 0.045 and 0.078 m, with worst-case deviations of 0.10 and 0.16 m, and the trailer followed at 0.223 and 0.263 m mean error without leaving the lane corridor in either trial. Mean heading error was 1.3° for the tractor and 3.8 to 4.0° for the trailer, and the two tracks agree closely enough that they overlay in fig. 5.5. Almost all of the forward error is concentrated in the final curve, where the trailer error rises from a mean of 0.10 m over the first 18 m to 0.60 m over the last 5 m and articulation reaches 39°.

Reverse tracking degraded substantially. The trailer held a mean cross-track error of 0.687 and 0.764 m with worst excursions near 1.9 m, and spent 11% of the maneuver outside the 1.41 m lane corridor. The tractor error is larger still, at 0.964 and 1.102 m mean against 2.39 and 2.35 m peak, and is the one metric where the tractor is worse than the trailer. That inversion is the expected signature of a reverse policy rewarded on trailer placement, which will carry the tractor wide to put the trailer where the path wants it. Mean articulation is 21° against 9° forward, and peak articulation reaches 51.6°, still comfortably inside the 80° operational limit and the 90° jackknife condition.

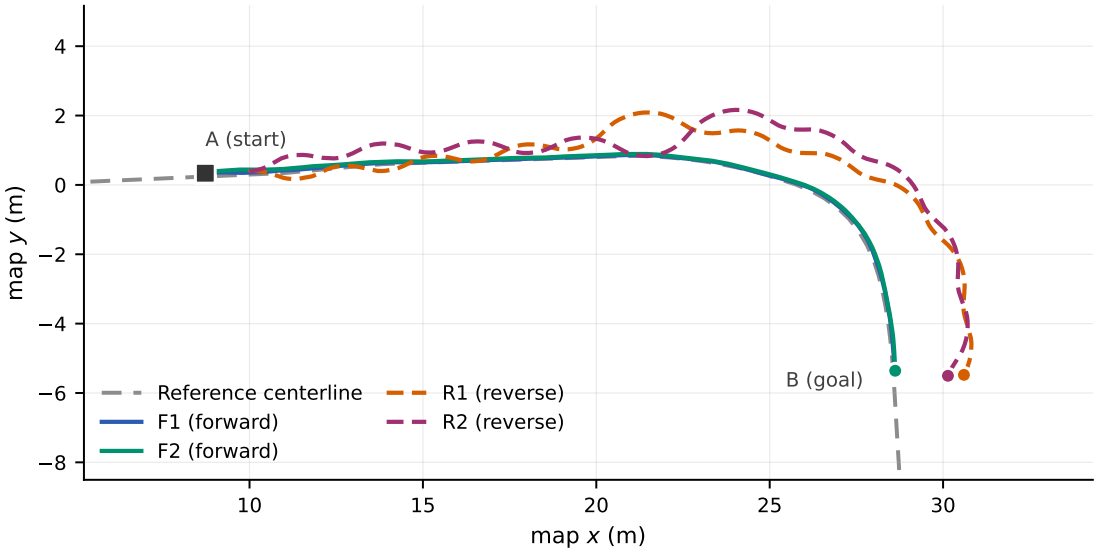


Figure 5.5: Tractor tracks for the four deployment trials against the surveyed lane centerline. Solid tracks are forward trials, dashed tracks are reverse. The two forward tracks are nearly coincident and largely hide one another. The forward trials sit on the reference lane throughout, while the reverse trials bow outward by up to two meters before converging near point B.

5.6.3 Articulation Limit Cycle

The reverse trials also exhibit a behavior the forward trials do not show at all, and that simulation does not reproduce, a sustained oscillation in the articulation angle. Figure 5.6 makes the contrast plain. In the forward trials the articulation trace is smooth and drifts slowly, moving off zero only for the final curve. In the reverse trials the hitch angle swings back and forth continuously, with a median peak-to-peak amplitude of 44 to 46° and a spatial wavelength of 2.4 to 2.6 m, roughly one trailer length per cycle, sustained from the start of the maneuver to the end.

The commanded steering rate follows the same split. In the forward trials it never once reaches its 0.436 rad/s saturation limit, while in the reverse trials it sits on that limit for 79 to 80% of the control ticks, so the reverse policy is operating as a rate-limited, approximately bang-bang controller rather than regulating articulation smoothly. Both directions ran on the same vehicle, the same actuator, the same articulation estimator, and the same bridge on the same day, and the contrast is consistent with residual actuation lag and articulation-estimate noise being damped by the stable forward articulation dynamics and amplified by the unstable reverse ones. These trials do not separate the relative contribution of the two effects.

On the straight section the oscillation is symmetric and largely averages out, and both

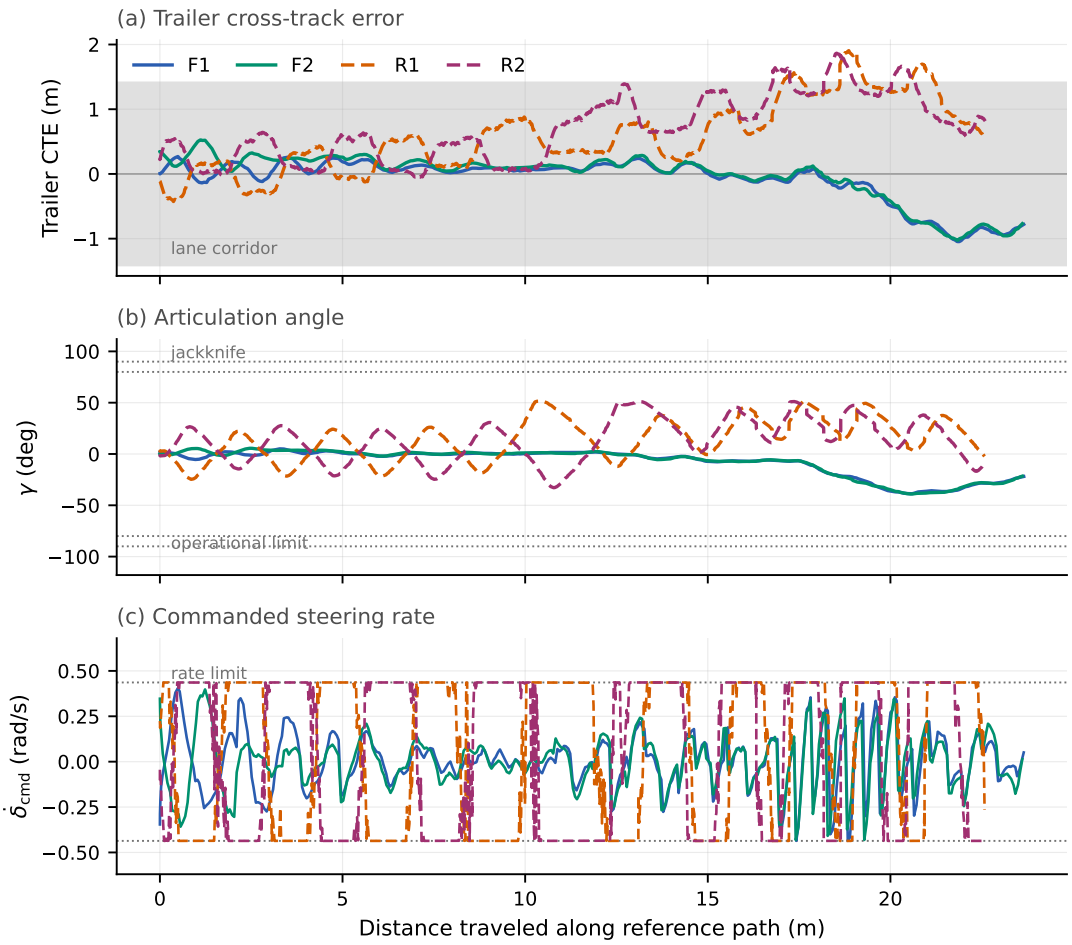


Figure 5.6: The four deployment trials against distance traveled along the reference path. (a) Trailer cross-track error, with the lane corridor shaded. (b) Articulation angle against the operational and jackknife limits. (c) Commanded steering rate against its saturation limit. Solid traces are forward trials, dashed traces are reverse. The reverse trials oscillate continuously and hold the steering rate on its limit, while the forward trials do neither.

reverse trials hold the trailer inside the lane corridor throughout it. The cost appears at the final curve, where the policy has to hold a sustained articulation offset while most of its steering-rate authority is already spent on the oscillation, and where the reference curvature saturates the $0.3\,\mathrm{m}^{-1}$ clip applied to the curvature observation, limiting the policy's ability to distinguish the sharpest part of the turn from a merely tight one. The largest trailer excursions of both reverse trials occur there.

Read narrowly, the four trials show that policies trained entirely in the 2D simulator drive a physical articulated vehicle along a real surveyed lane without any retraining on hardware. Forward lane following transferred cleanly, while reverse control completed the maneuver with a rate-saturated articulation oscillation that the training simulator does not reproduce.

Narrowing that gap is a matter of modeling the actuator lag and the articulation-estimate noise during training rather than compensating for them at deployment, which is taken up in chapter 6.

5.7 Summary

This chapter demonstrated that the tractor-trailer control architecture developed in simulation can be ported to a physical platform by executing the trained RL policies directly through a bridging layer, rather than by re-deriving an analytical controller on the vehicle. The deployment retains Autoware's standard localization and perception augmented with the infrastructure sensor node obstacle list while replacing planning and control with an RL bridge that runs the forward and reverse policies at 10 Hz over the same observation and action interfaces used in training. The principal engineering contribution is this RL bridge and its integration into a stripped-down Autoware stack, drawing the articulation angle from the onboard LiDAR estimator rather than from any trailer-mounted sensor. The gap analysis confirms that the simulation captures the dominant dynamics relevant to controller design, while identifying actuation delay, hitch-angle sensing, path-distribution mismatch, and localization reliability as the primary points of divergence between the simulated and physical systems. Finally, the infrastructure sensor nodes, implemented and running in the laboratory, provide a route to occlusion-resilient perception behind the trailer without adding sensors to the trailer itself.

Chapter 6

Conclusion

6.1 Summary of Findings

This thesis introduces a learning-based approach to automated articulated tractor-trailer control that integrates policy learning, high-dimensional perception, obstacle-aware decision making, and physical deployment within a common framework. By carrying the approach from simulation through hardware evaluation, the work establishes a foundation toward a more general framework for complex autonomous tractor-trailer systems.

On the compact engineered state, TD3 under a multiplicative reward that jointly penalizes tracking error and hitch articulation was the only combination that solved the unstable reverse task reliably, which fixed the algorithm and the reward for everything that followed. Opening the observation up to planar LiDAR and then to the raw bird's-eye-view image reversed the intuition that richer sensing is better, since reliability fell as the observation grew.

Inserting a raw image branch into the policy, whether learned end to end or pre-trained as an autoencoder or a UNet, does not by itself produce a trainable vision policy. That failure proved to be an optimization pathology of off-policy actor-critic learning rather than a shortage of perceptual information, isolated by a ladder of experiments in which representation-side fixes changed nothing and only freezing the actor rescued a cloned policy. Distilling the BEV image into the vehicle's path geometry with a frozen supervised encoder removes the pathology at its root and restores the from-scratch learning behavior of the engineered-state baseline, and the recoverability measurements confirm that the image encodes the geometry the engineered state summarizes.

The same distillation carries the obstacle task, where the engineered state is genuinely insufficient. The policy follows an avoidance path inferred from the image and gates its attempt-versus-stop decision on a windowed difficulty rating, which has to be made state-

dependent in reverse because tracking well at one moment does not imply the trailer will clear the coming gap. Against tuned pure-pursuit, PID, and MPC baselines the learned policy reaches comparable tracking competence, and its distinctive value is the learned decision to decline an infeasible layout. Carried onto the Hunter SE through the RL bridge of chapter 5, the forward policy held the tractor within a tenth of a meter of the surveyed lane, while the reverse policy completed the same route with a steering-rate-saturated articulation oscillation that the training simulator does not reproduce.

6.2 Limitations

The study is subject to the following constraints:

- The bulk of the experimental evaluation is conducted within the custom 2D Pygame/-Gymnasium simulation environment. The hardware evidence of chapter 5 covers lane following on a single surveyed route, so sim-to-real performance across the full range of training tasks, and behavior in higher-fidelity physics simulators, remain to be characterized.
- The 2D simulation approximates the vehicle dynamics but does not capture all effects present in a real tractor-trailer system, such as load transfer, tire saturation at high slip angles, or hitch joint compliance.
- The framework assumes a fixed payload and a fixed trailer geometry. Hitch dynamics change significantly with trailer mass, and the policies are trained against a single wheelbase and trailer length, so their sensitivity to a different tractor or trailer size has not been characterized. The hardware deployment sidesteps this by re-parameterizing the model and retraining (section 5.4) rather than by transferring a policy across configurations.
- Obstacle avoidance is limited to static obstacles.
- Generalization beyond the training distribution has not been quantified, including robustness to sensor degradation and to stronger path-distribution shift. Command latency, measured on the platform at about 0.09 s of transport dead time ahead of a 0.11 s actuator lag, is compensated on hardware by the Smith predictor, and the reverse trials still developed the articulation oscillation reported there.

6.3 Future Work

Several steps remain to further develop the proposed framework and establish its performance beyond the conditions evaluated in this thesis:

- **Geometry-agnostic policies**: Training a single policy that holds up across trailer lengths, wheelbases, and hitch offsets rather than against one fixed configuration, so that a new vehicle does not require a re-parameterized simulator and a fresh training run. The batched simulator already supports sweeping these parameters, which makes both the measurement of how a policy degrades away from its training geometry and the training of a policy conditioned on it practical.

- **Full-scale hardware deployment**: Carrying the policies onto the laboratory's full-size electric shunt truck (fig. 3.2), the vehicle class the simulation model represents and the scale at which the confined yard maneuvers this thesis targets are actually performed. The hardware trials reported here were run on a compact platform, so the step up in scale exercises the geometry-agnostic training above and the actuation and articulation-sensing gaps of chapter 5 together.

- **Dynamic obstacles**: Extending the observation to carry how the scene evolves rather than only how it looks at the current instant, so that the policy can infer where a moving obstacle is heading and avoid it predictively. The same temporal information gives the controller the internal state it needs to compensate dynamically on the unstable reverse task.

- **Geographic simulation and behavior integration**: Exercising the trained policies in the georeferenced Gazebo "digital-twin" environment of chapter C, which reconstructs real sites from Blender-authored, GIS-anchored worlds and runs the full Autoware perception and localization stack. On top of that, a behavior layer that composes the existing lane-following capability into complete tasks, so that a goal such as parking a trailer in a designated space is decomposed into the forward and reverse maneuvers the policies already perform.

Bibliography

[1] R. Tymkow and B. Schnapp, "Application of deep deterministic policy gradient (ddpg) and decision transformer reinforcement learning to vehicle path following," Tech. Rep. ME 780 Course Project Report, University of Waterloo, Dept. of Mechanical and Mechatronics Engineering, 2024. Unpublished graduate course report.

[2] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2016.

[4] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[5] B. Schnapp, "LQR control of vehicle bicycle model," Tech. Rep. ECE 682 Course Project Report, University of Waterloo, Dept. of Mechanical and Mechatronics Engineering, 2024. Unpublished graduate course report.

[6] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 1587–1596, 2018.

[7] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Tech. Rep. CMU-RI-TR-92-01, Robotics Institute, Carnegie Mellon University, 1992.

[8] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Madison, WI: Nob Hill Publishing, 2 ed., 2017.

[9] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, 2017.

[10] P. Fancher and C. Winkler, "Directional performance issues in evaluation and design of articulated heavy vehicles," *Vehicle System Dynamics*, vol. 45, pp. 607–647, 2007.

[11] C. Altafini, A. Speranzon, and B. Wahlberg, "A feedback control scheme for reversing a truck and trailer vehicle," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 915–922, 2001.

[12] B. D. O. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[13] A. M. C. Odhams, R. L. Roebuck, B. A. Jujnovich, and D. Cebon, "Active steering of a tractor–semi-trailer," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 225, no. 7, pp. 847–869, 2011.

[14] B. A. Jujnovich and D. Cebon, "Path-following steering control for articulated vehicles," *Journal of Dynamic Systems, Measurement, and Control*, vol. 135, no. 3, p. 031006, 2013.

[15] M. Beglini, L. Lanari, and G. Oriolo, "Anti-jackknifing control of tractor-trailer vehicles via intrinsically stable MPC," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8806–8812, 2020.

[16] M. Fliess, J. Lévine, P. Martin, and P. Rouchon, "Flatness and defect of non-linear systems: Introductory theory and examples," *International Journal of Control*, vol. 61, no. 6, pp. 1327–1361, 1995.

[17] C. Samson, "Control of chained systems: Application to path following and time-varying point-stabilization of mobile robots," *IEEE Transactions on Automatic Control*, vol. 40, no. 1, pp. 64–77, 1995.

[18] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[19] S. M. LaValle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[20] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[21] O. Ljungqvist, N. Evestedt, M. Cirillo, D. Axehill, and O. Holmer, "Lattice-based motion planning for a general 2-trailer system," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 819–824, 2017.

[22] R. Oliveira, O. Ljungqvist, P. F. Lima, and B. Wahlberg, "Optimization-based on-road path planning for articulated vehicles," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15572–15579, 2020.

[23] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *American Control Conference (ACC)*, pp. 2296–2301, 2007.

[24] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 2 ed., 2018.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[26] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pp. 387–395, 2014.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 1861–1870, 2018.

[29] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2022.

[30] F. Codevilla, M. Müller, A. M. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[31] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8248–8254, 2019.

[32] A. El Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging, Autonomous Vehicles and Machines*, vol. 29, pp. 70–76, 2017.

[33] D. H. Nguyen and B. Widrow, "Neural networks for self-learning control systems," *IEEE Control Systems Magazine*, vol. 10, no. 3, pp. 18–23, 1990.

[34] E. Bejar and A. Morán, "Backing up control of a self-driving truck-trailer vehicle with deep reinforcement learning and fuzzy logic," in *2018 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pp. 202–207, 2018.

[35] Q. Kang, A. Hartmannsgruber, S.-H. Tan, X. Zhang, and C.-M. Chew, "Deep reinforcement learning based tractor-trailer tracking control," in *2024 IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3147–3153, 2024.

[36] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the 16th International Conference on Machine Learning (ICML)*, pp. 278–287, 1999.

[37] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 15084–15097, 2021.

[38] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pp. 41–48, 2009.

[39] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *Journal of Machine Learning Research*, vol. 21, no. 181, pp. 1–50, 2020.

[40] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations (ICLR)*, 2016.

[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, pp. 1097–1105, 2012.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[44] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[45] J. Philion and S. Fidler, "Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d," in *European Conference on Computer Vision (ECCV)*, pp. 194–210, 2020.

[46] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[47] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[48] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2010.

[49] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations (ICLR)*, 2014.

[50] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 512–519, 2016.

[51] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 234–241, 2015.

[52] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3d classification and segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 652–660, 2017.

[53] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020.

[54] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017.

[55] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[56] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, 2019.

[57] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4243–4250, 2018.

[58] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "CARLA: An open urban driving simulator," in *Conference on Robot Learning (CoRL)*, pp. 1–16, 2017.

[59] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, pp. 1437–1480, 2015.

[60] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *18th European Control Conference (ECC)*, pp. 3420–3431, 2019.

[61] International Organization for Standardization, "ISO 21448:2022 road vehicles — safety of the intended functionality." ISO Standard, 2022.

[62] International Organization for Standardization, "ISO 26262: Road vehicles — functional safety." ISO Standard, 2018.

[63] Underwriters Laboratories, "UL 4600: Standard for safety for the evaluation of autonomous products." UL Standard, 2022.

[64] SAE International, "J3016: Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles." SAE Standard, 2021.

[65] R. Rajamani, *Vehicle Dynamics and Control*. New York: Springer, 2 ed., 2011.

[66] Kalmar Ottawa, *Ottawa 4x2 DOT/EPA Terminal Tractor: Standard Specifications*. Kalmar Ottawa, 2022. Manufacturer specification sheet.

[67] D. D. MacInnis, W. E. Cliff, and K. W. Ising, "A comparison of moment of inertia estimation techniques for vehicle dynamics simulation," in *SAE Technical Paper Series*, no. 970951, (Warrendale, PA), pp. 99–117, SAE International, 1997.

[68] R. D. Ervin, C. B. Winkler, J. E. Bernard, and R. K. Gupta, "Effects of tire properties on truck and bus handling, volume IV: Final report," Tech. Rep. DOT-HS-802-143, Highway Safety Research Institute, University of Michigan, Ann Arbor, MI, 1976. Prepared for the National Highway Traffic Safety Administration, U.S. Department of Transportation.

[69] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[70] Pygame community, "pygame: Python game development library." https://www.pygame.org. Cross-platform set of Python modules built on SDL.

[71] H. B. Pacejka, *Tyre and Vehicle Dynamics*. Oxford: Butterworth-Heinemann, 3 ed., 2012.

[72] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[73] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "CuPy: A NumPy-compatible library for NVIDIA GPU calculations," in *Proceedings of Workshop on Machine Learning Systems (LearningSys) at the 31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

[74] N. J. Higham, "The scaling and squaring method for the matrix exponential revisited," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 4, pp. 1179–1193, 2005.

[75] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 2052–2062, 2019.

[76] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 20132–20145, 2021.

[77] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving sample efficiency in model-free reinforcement learning from images," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 10674–10681, 2021.

[78] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Proceedings of the Conference on Robot Learning (CoRL)*, pp. 66–75, 2020.

[79] O. J. M. Smith, "Closer control of loops with dead time," *Chemical Engineering Progress*, vol. 53, no. 5, pp. 217–219, 1957.

[80] P. Biber and W. Straßer, "The normal distributions transform: A new approach to laser scan matching," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2743–2748, 2003.

[81] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[82] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pp. 287–296, 2018.

[83] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep

learning library," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, pp. 8024–8035, 2019.

[84] Y. Cao, "Object segmentation and reconstruction using infrastructure sensor nodes for autonomous mobility," Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2023. MASc thesis.

[85] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[86] X. Zhang, W. Xu, C. Dong, and J. M. Dolan, "Efficient L-shape fitting for vehicle detection using laser scanners," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 54–59, 2017.

[87] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[88] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2149–2154, 2004.

[89] B. Perseghetti and Rudis Laboratories, "rudislabs_gazebo_worlds: Georeferenced Gazebo world models." https://github.com/rudislabs/rudislabs_gazebo_worlds, 2021. Open-source software repository, Apache-2.0 licence.

[90] Blender Online Community, *Blender — a 3D Modelling and Rendering Package*. Blender Foundation, Amsterdam. https://www.blender.org.

[91] OpenStreetMap contributors, "OpenStreetMap." https://www.openstreetmap.org, 2024. Map data available under the Open Database License (ODbL).

APPENDICES

Appendix A

Classical Controller Implementation

The deep reinforcement-learning policy is benchmarked against four classical path-tracking controllers. This appendix documents their implementation: the exact control laws, the forward and reverse variants, the way each controller was tuned, and the harness used to evaluate them. The four controllers are chosen to span the classical control spectrum, from a purely geometric law through to constrained optimal control with preview:

- **Pure Pursuit**, a geometric proportional law;
- **PID**, classical error feedback with an integral term;
- **LQR**, optimal static linear feedback on the error dynamics;
- **Kinematic LTV MPC**, optimal constrained control with preview.

All four are implemented in the batched simulator (`tractor_trailer_rl_cupy`) so that they run inside the same vectorized environment as the learned policies. Crucially, every controller emits the same action as the RL agent, a bounded steering rate, so the comparison between classical and learned control is made on identical terms.

A.1 Common framework

A.1.1 Observation layout and shared action mapping

Each controller is a function of the observation vector already produced by the environment. For the trailer tasks this vector is

$$o = [\, s,\ \gamma,\ e_y,\ e_\psi,\ e_{y,t},\ e_{\psi,t},\ \kappa,\ \ldots\,], \tag{A.1}$$

where s is the current steering angle, γ the hitch (articulation) angle, and the error pairs (e_y, e_ψ) and $(e_{y,t}, e_{\psi,t})$ are the cross-track and heading errors of the tractor and of the trailer respectively; κ is the reference path curvature at the lookahead point. The trailing entries carry additional task-specific channels (for example LiDAR returns) that the classical controllers do not use.

Every controller computes a desired steering angle δ_{des} and then converts it to the environment's steering-rate action through a common saturating map,

$$\dot{\delta} = \text{clip}\left(\frac{\delta_{\text{des}} - s}{\Delta t},\ -\dot{\delta}_{\max},\ +\dot{\delta}_{\max}\right), \tag{A.2}$$

where Δt is the simulation step and $\dot{\delta}_{\max}$ the steering-rate limit. Because the learned policy also outputs $\dot{\delta}$, and eq. (A.2) applies to all four baselines identically, the classical and learned controllers act through exactly the same interface.

A.1.2 Forward and reverse tracking

Each controller has a forward and a reverse variant, reflecting a structural asymmetry of the articulated vehicle. Driving forward, the tractor leads and the trailer follows passively, so it suffices to regulate the tractor errors (e_y, e_ψ) towards zero. Driving in reverse, the trailer leads and the coupling inverts. The hitch angle γ becomes open-loop unstable (small articulation errors grow and the vehicle jackknifes), so the reverse controllers regulate the trailer errors $(e_{y,t}, e_{\psi,t})$ and must additionally include an explicit hitch-stabilizing action. This asymmetry is why the reverse variants below are not simply the forward laws with a sign change, and why their gains had to be tuned separately (section A.6).

A.1.3 Vehicle parameters

The benchmark uses the full-size shunt-truck configuration. The parameters that enter the controllers are summarized in table A.1. Here L_1 is the tractor wheelbase, L_2 the effective trailer length (hitch to trailer axle), and V the fixed longitudinal speed held constant throughout the fixed-speed path-tracking experiments. The large length ratio $L_2/L_1 \approx 4.2$ is what makes the reverse hitch dynamics difficult to stabilize.

Table A.1: Shunt-truck parameters used by the classical controllers.

Symbol	Quantity	Value
L_1	Tractor wheelbase	2.95 m
L_2	Trailer length (hitch to axle)	12.5 m
V	Fixed longitudinal speed	5.0 m/s
Δt	Simulation step	0.1 s
$\delta_{\max}$	Steering-angle limit	0.87 rad ($\approx 50°$)
$\dot{\delta}_{\max}$	Steering-rate limit	25°/s

In the reverse variants the speed enters with a sign, $V < 0$. Because the environment's tractor yaw response to steering does not itself flip sign in reverse, the along-path kinematics use the magnitude $|V|$ while the yaw dynamics retain the signed V. This distinction is carried consistently through the LQR and MPC models below.

A.2 Pure Pursuit

The pure-pursuit controller [7] is a fixed proportional geometric law. Forward, it combines a curvature feed-forward with proportional cross-track and heading feedback on the tractor errors,

$$\delta_{\text{des}} = -\big(k_{\text{ff}} \arctan(L_1 \kappa) + k_y\, e_y + k_\theta\, e_\psi\big). \tag{A.3}$$

Reverse, it regulates the trailer errors and adds an explicit hitch-stabilizing term $k_{\text{hitch}}\, \gamma$ to counter the open-loop-unstable articulation,

$$\delta_{\text{des}} = k_{\text{hitch}}\, \gamma + k_{\text{ff}}\, \kappa + k_y\, e_{y,t} + k_\theta\, e_{\psi,t}. \tag{A.4}$$

Both laws are stateless functions of the observation, so the controller is fully vectorized across the parallel environments. Besides serving as a benchmark, the tuned pure-pursuit controller doubles as the reference ("teacher") signal for the guided reward.

A.3 PID

The PID controller extends the proportional law with an integral term that removes steady-state cross-track offset. The integral is accumulated per environment and clamped to prevent

windup, and it is reset at episode boundaries. Forward,

$$\delta_{\text{des}} = -k_{\text{ff}}\,\kappa - K_p\,e_y - K_i\int e_y - K_d\,e_\psi + K_{p,t}\,e_{y,t}, \tag{A.5}$$

and reverse (regulating the trailer errors, with the hitch stabilizer),

$$\delta_{\text{des}} = k_{\text{hitch}}\,\gamma + k_{\text{ff}}\,\kappa + K_p\,e_{y,t} + K_i\int e_{y,t} + K_d\,e_{\psi,t}. \tag{A.6}$$

The derivative gain K_d multiplies the heading error, so it acts as a heading-damping term rather than a numerical time derivative. The integrator uses the anti-windup update

$$I \leftarrow \text{clip}\big(I + e\,\Delta t,\ -I_{\max},\ +I_{\max}\big), \tag{A.7}$$

where e is the regulated cross-track error (e_y forward, $e_{y,t}$ reverse). Because the controller carries this per-environment state, the evaluation loop passes a termination mask so that I is zeroed for any environment that resets on the current step.

A.4 LQR

The LQR baseline is the canonical optimal-linear controller, static feedback $u = -Kx$ on a linear model of the tracking-error dynamics, plus a curvature feed-forward [65]. The error state is

$$x = [\,e_{\text{lat}},\ e_{\text{head}},\ \gamma\,]^\top, \tag{A.8}$$

the lateral and heading error of the leading unit (tractor forward, trailer reverse) together with the hitch angle. The input is the steering angle $u = \delta$. The continuous error dynamics $\dot{x} = A_c x + B_c u$ are direction-specific. Forward,

$$A_c = \begin{bmatrix} 0 & V & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\dfrac{V}{L_2} \end{bmatrix}, \qquad B_c = \begin{bmatrix} 0 \\ \dfrac{V}{L_1} \\ \dfrac{V}{L_1} \end{bmatrix}, \tag{A.9}$$

and reverse, where the leading trailer progresses along the path at speed $|V|$ while the yaw dynamics keep the signed V,

$$A_c = \begin{bmatrix} 0 & |V| & 0 \\ 0 & 0 & \dfrac{V}{L_2} \\ 0 & 0 & -\dfrac{V}{L_2} \end{bmatrix}, \qquad B_c = \begin{bmatrix} 0 \\ 0 \\ \dfrac{|V|}{L_1} \end{bmatrix}. \tag{A.10}$$

The pair (A_c, B_c) is discretized by zero-order hold at the step Δt to give (A_d, B_d). With state and input weights $Q = \mathrm{diag}(q_{\mathrm{lat}}, q_{\mathrm{head}}, q_\gamma)$ and R, the infinite-horizon gain follows from the discrete algebraic Riccati equation. Its stabilizing solution P yields

$$K = \left(R + B_d^\top P B_d\right)^{-1} B_d^\top P A_d. \tag{A.11}$$

Because the speed is constant, K is computed once at construction and then applied to every environment, so the controller stays fully vectorized. The command adds a steady-state feed-forward that holds the vehicle on a constant-curvature arc: a feed-forward steer $\delta_{\mathrm{ff}} = L_1 \kappa$ and a hitch reference $\gamma_{\mathrm{ref}} = \mathrm{sign}(V)\, L_2\, \kappa$, giving

$$\delta = \delta_{\mathrm{ff}} - K\,(x - x_{\mathrm{ref}}), \qquad x_{\mathrm{ref}} = [\,0,\ 0,\ \gamma_{\mathrm{ref}}\,]^\top, \tag{A.12}$$

which is then saturated to $\pm\delta_{\max}$ and mapped to a steering rate through eq. (A.2). The reverse weights place a heavy penalty on control effort and hitch deviation, which is what damps the otherwise unstable reverse plant.

A.5 Kinematic LTV MPC

The final baseline is a linear time-varying model-predictive controller [8] that adds a finite preview horizon and explicit input constraints. The MPC uses the same error state $x = [\,e_{\mathrm{lat}}, e_{\mathrm{head}}, \gamma\,]^\top$ as the LQR but plans over an N-step horizon.

A.5.1 Prediction models

Forward, the tractor leads, so the MPC tracks the tractor errors with the steering angle δ as the input, giving the same (A_c, B_c) as the forward LQR model (A.9). Reverse, servoing the trailer pose with the unstable hitch dynamics inside the optimization causes the long shunt trailer to oscillate and jackknife. The reverse controller therefore uses a flatness / virtual-

input reformulation (following the Autoware trailer LTV-MPC): the QP plans the trailer as a stable bicycle driven by a virtual trailer steer δ_T, and the hitch enters only through an inner loop that has been closed to the stable first-order dynamics $\dot{\gamma} = K(\delta_T - \gamma)$,

$$A_c = \begin{bmatrix} 0 & |V| & 0 \\ 0 & 0 & \dfrac{V}{L_2} \\ 0 & 0 & -K \end{bmatrix}, \qquad B_c = \begin{bmatrix} 0 \\ 0 \\ K \end{bmatrix}. \tag{A.13}$$

The reference curvature enters both models as a measured disturbance through $E_c = [\,0,\ -|V|,\ 0\,]^\top$. After the QP returns the virtual input, an analytic inner map realizes it on the physical tractor and supplies the anti-jackknife counter-steer,

$$\delta_f = \arctan\!\left(\frac{L_1}{L_2}\,\operatorname{sign}(V)\,\sin\gamma + \frac{L_1 K}{|V|}\,(\delta_T - \gamma)\right). \tag{A.14}$$

The $1/|V|$ factor is what flips the feedback sign in reverse and produces the stabilizing counter-steer.

A.5.2 Condensed quadratic program

The continuous models are discretized by zero-order hold and the state trajectory is condensed onto the input sequence, so that the optimization variable is the vector of N future steering inputs. With the stacked weight $\bar{Q} = I_N \otimes Q$, a control penalty R, an input-rate penalty R_d, and the first-difference operator $T = I - I^{(-1)}$ (so that T forms Δu), the cost Hessian is

$$H = 2\big(B_u^\top \bar{Q}\, B_u + R\, I + R_d\, T^\top T\big), \tag{A.15}$$

where B_u is the condensed input-to-state map. The problem is subject to a steering-rate box on the increments, $|\Delta u| \le \dot{\delta}_{\max}\,\Delta t$, and a steering box $|u| \le \delta_{\max}$. It is solved per environment with OSQP. The rate-penalty term $R_d\,T^\top T$ is weighted heavily in reverse so the controller does not chase cross-track measurement ripple. The overall per-step procedure is summarized in algorithm 1.

Algorithm 1 Kinematic LTV MPC step (per environment)

Precompute (once): discretize (A_c, B_c, E_c) by ZOH; build the condensed maps A_x, B_u, E_w and the constant Hessian H from (A.15).
On episode reset: clear the warm-start input $u_{\text{prev}} \leftarrow 0$.
Read $x_0 = [\, e_{\text{lat}}, e_{\text{head}}, \gamma \,]$ and curvature κ from the observation.
Form the linear cost term from x_0, the previewed curvature disturbance, and u_{prev}.
Solve the QP with OSQP subject to the steering-rate and steering boxes.
if solver fails **then** $u \leftarrow u_{\text{prev}}$ ▷ safe fallback
else $u \leftarrow$ first element of the optimal input sequence
end if
$u_{\text{prev}} \leftarrow u$
$\delta \leftarrow u$ (forward) **or** apply inner map (A.14) to obtain δ_f (reverse).
return steering rate from (A.2).

Only the first input of the optimal sequence is applied (receding horizon). The solved input is retained as the warm start for the next step and, like the PID integrator, is reset at episode boundaries. If the solver fails to converge, the controller falls back to the previous command.

A.6 Gain-tuning methodology

The controllers above are defined by their gains and weights: the proportional and feed-forward gains of pure pursuit, the PID gains, the LQR state/input weights, and the MPC horizon and cost weights. Rather than report a specific set of numbers, which are particular to one vehicle configuration, the point of interest is the procedure used to obtain them, which is the same for every controller.

Each controller is tuned by a bounded random search (a gain sweep) run directly against the simulator [85]. On each iteration a candidate gain set is sampled uniformly from hand-set ranges, the controller is rolled out over a batch of parallel environments, and the candidate is scored by the lexicographic key

$$\big(\text{completion rate},\ -\,\text{mean cross-track error}\big), \tag{A.16}$$

so that reliably completing the path is maximized first and, among gains that complete, the one with the tightest tracking is preferred. The best candidate is retained. This search is run separately for each controller and each direction, because the forward and reverse plants

differ so strongly.

The reverse variants in particular could not simply inherit the gains tuned on the small laboratory vehicle. On the shunt truck those gains destabilize the vehicle, and several gains (notably the hitch-stabilizing terms) change sign. This is a direct consequence of the open-loop-unstable reverse hitch dynamics and the large trailer-to-tractor length ratio $L_2/L_1 \approx 4.2$ discussed in section A.1.2. The reverse controllers were therefore re-tuned from scratch for the target geometry.

A.7 Evaluation harness

All four controllers are evaluated by a single harness that reuses the same batched environment as the reinforcement-learning policies, so that the classical baselines and the learned policy are measured on identical paths under identical dynamics. The harness runs both driving directions and, for each controller, records three per-episode metrics:

- **Completion**, the fraction of episodes that reach the goal (the mean success flag);
- **Trailer cross-track error**, the mean absolute lateral error of the trailer over the episode, a measure of tracking accuracy;
- **Peak hitch angle**, the largest articulation angle reached, a measure of how close the vehicle came to jackknifing (the jackknife threshold is 1.4 rad).

The harness handles the stateful controllers by passing a per-environment termination mask into the controller at each step, so the PID integrator (A.7) and the MPC warm start are reset exactly when an environment restarts. Evaluating every controller in the same loop, on the same unfiltered evaluation pool, is what makes the reinforcement-learning comparison in the results chapter fair. The measured numbers are reported there rather than in this appendix.

Appendix B

Onboard Hitch Angle Estimation

The articulation angle γ that the deployed policies of chapter 5 consume is estimated onboard rather than measured by any trailer-mounted sensor. Because the trailer carries no articulation encoder and no sensing of its own, the angle is recovered geometrically from the tractor's front LiDAR, which observes the front (and, at larger angles, the side) face of the trailer. This estimator was developed by S. Huang, and it operates on each incoming point cloud as follows.

Region of interest

Points are first cropped to a box in the LiDAR frame that brackets the expected location of the trailer face. The lateral extent of this region is shifted to track the face as the trailer swings. Using the current filtered estimate γ, the lateral offset is set to

$$\Delta y = -\, D_{\text{face}} \sin \gamma, \tag{B.1}$$

where D_{face} is the approximate longitudinal distance from the hitch pivot to the face. This keeps the cropped points on the trailer face across the full range of articulation rather than only near $\gamma = 0$.

Rectangle fitting

Rather than fitting a single planar normal, the estimator fits the orientation of the trailer body to the cropped points using the minimum-closeness rectangle criterion of Zhang et al. [86]. For a candidate orientation θ, the points are rotated into axis-aligned coordinates

(u, v) and scored by their closeness to the nearest edges of the bounding rectangle,

$$C(\theta) = \sum_i \min\big(d_i^u,\, d_i^v\big)^2, \qquad d_i^u = \min\big(|u_i - u_{\min}|,\, |u_i - u_{\max}|\big), \tag{B.2}$$

with d_i^v defined analogously. The fitted orientation minimizes $C(\theta)$ augmented by a dimension-prior penalty that favors rectangles consistent with the known trailer width and length,

$$\theta^\star = \arg\min_{\theta \in [0,\pi)} \; \big[\, C(\theta) + \lambda\, P_{\dim}(\theta) \,\big]. \tag{B.3}$$

This formulation is robust to whether only the front face, both the front and a side face (an "L" shape), or only a side face is visible, the dominant cause of failure for a naive single-face normal estimate at large articulation. The four 90° heading candidates implied by $\theta^\star$ are disambiguated by choosing the one closest to the previous filtered estimate and most consistent with the dimension prior, yielding a trailer longitudinal direction $\hat{\mathbf{d}}_t = (\cos\theta^\star, \sin\theta^\star)$.

Articulation angle and filtering

The raw articulation angle is the signed angle between the truck-forward axis $\hat{\mathbf{x}} = (1, 0)$ and $\hat{\mathbf{d}}_t$,

$$\tilde{\gamma} = \operatorname{atan2}\big(\hat{x}_1 \hat{d}_{t,2} - \hat{x}_2 \hat{d}_{t,1},\; \hat{\mathbf{x}} \cdot \hat{\mathbf{d}}_t\big), \tag{B.4}$$

to which a field-calibration sign s and offset γ_0 are applied and the result clamped to the operational range, $\gamma = \operatorname{clip}(s\,\tilde{\gamma} + \gamma_0,\; -\gamma_{\max},\; \gamma_{\max})$. The sign convention is chosen so that the published value matches Autoware's $\gamma = \psi_{\text{truck}} - \psi_{\text{trailer}}$. The raw per-scan estimate is then passed through a constant-angular-velocity Kalman filter [87] on the state $[\gamma,\; \dot{\gamma}]$, with Mahalanobis innovation gating to reject outlier scans, and a short moving-average is maintained in parallel as a diagnostic. The filtered angle and its rate populate the `hitch_angle` and `hitch_rate` fields of the `TrailerState` message published on `/vehicle/trailer_state`.

This onboard, geometry-plus-filter estimate is the actual mechanism behind that signal, and it is not an external measurement. Its noise characteristics, and the way the deployment handles them, are taken up in section 5.4.

Appendix C

High-Fidelity Geographic Simulation

The 2D Pygame environment of chapter 3 is deliberately abstract. It captures the tractor-trailer kinematics and lane geometry but not the three-dimensional and photometric structure that the deployed perception and localization stack actually consumes. A complementary, higher-fidelity simulation environment was therefore built to exercise the full Autoware stack (LiDAR-based NDT localization, perception, and the RL bridge) inside a geographically faithful "digital twin" of a real operating site as a complement to the physical trials.

This environment is built on the Gazebo simulator [88], extending the open-source Rudis Laboratories georeferenced world-generation toolchain [89]. Real locations are reconstructed as textured 3D meshes authored in Blender [90], whose physically based materials are baked into texture maps for efficient real-time rendering. Building on that toolchain, the target distribution-center site was rebuilt in Blender directly from public geographic data, with both layers imported through the Blender GIS plugin. Building footprints were pulled from OpenStreetMap [91] and extruded to form the 3D building meshes, and these were draped over aerial orthoimagery from the ESRI World Imagery basemap, replacing the lower-resolution ground textures of the original toolchain with the higher-resolution ESRI imagery (fig. C.1). Because the reconstruction is driven entirely by the OpenStreetMap footprints and the ESRI imagery layer, the same procedure generates a Gazebo world for any real location, rather than being tied to a single hand-authored site. Each world is anchored to GIS coordinates through the SDF `spherical_coordinates` element, which fixes the simulation origin to a real latitude, longitude, and elevation. The `electrans` tractor and trailer models are then spawned into these worlds, so that the same articulated vehicle can be driven through a virtual replica of the target environment (fig. C.3).

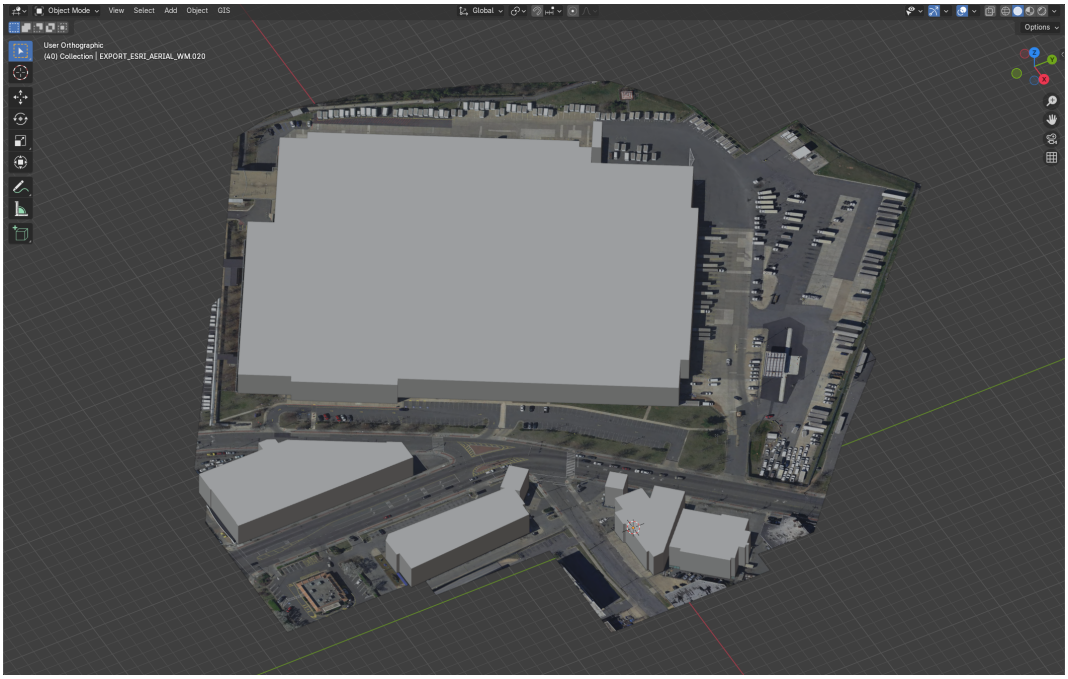


Figure C.1: Reconstruction of the target distribution-center site in Blender. OpenStreetMap building footprints are extruded to form the 3D building meshes and draped over high-resolution aerial orthoimagery from the ESRI World Imagery basemap, both layers imported through the Blender GIS plugin. The same GIS-driven procedure reconstructs any real distribution center from its geographic coordinates.

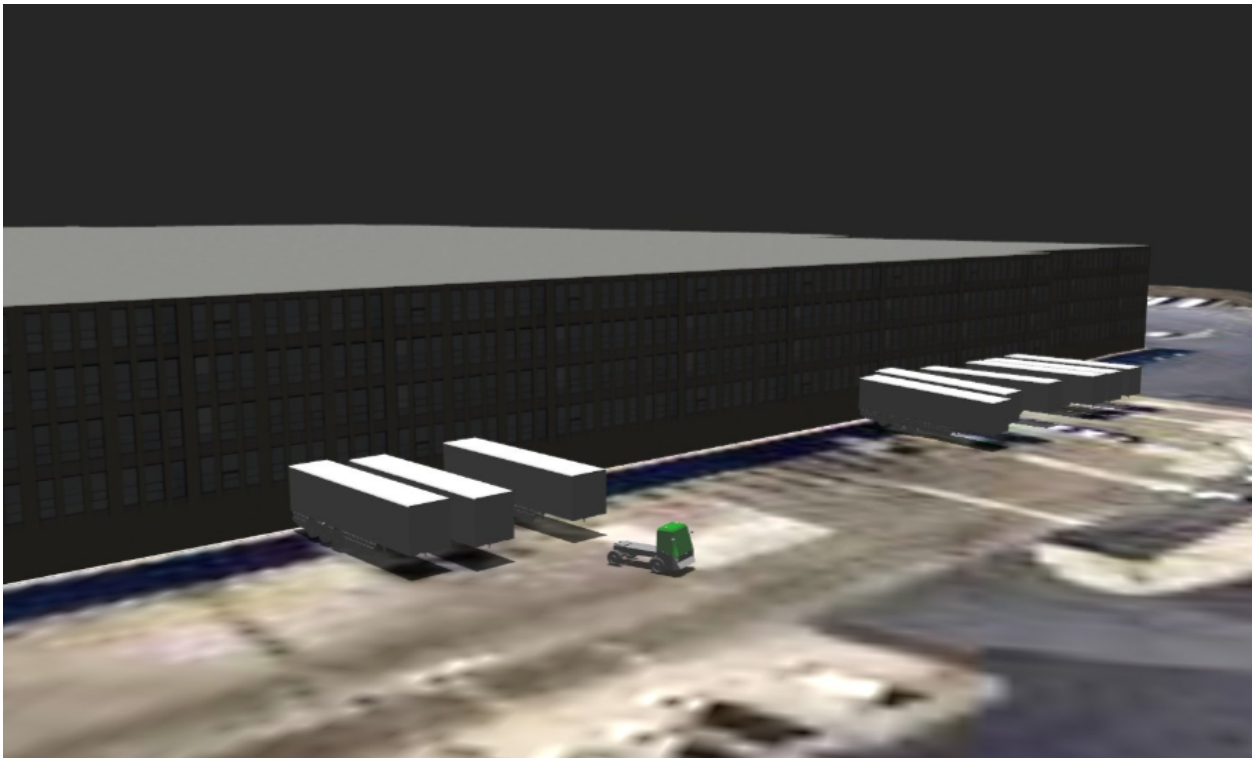


Figure C.2: The `electrans` tractor spawned at a loading dock within the world.

Because the world is georeferenced and rendered in three dimensions, this environment directly targets several of the sim-to-real gaps catalogued above. A LiDAR sensor model produces point clouds against real building and ground geometry, so NDT scan matching and the onboard hitch-angle estimator (chapter B) can be exercised against realistic returns; the geographic anchoring lets the GNSS and map frames be reproduced consistently with the physical site; and the photorealistic rendering makes it possible to evaluate camera-based perception that the abstract 2D environment cannot represent. It is also the natural setting in which to prototype the infrastructure sensor nodes of section 5.5, since virtual LiDAR units can be placed at arbitrary surveyed locations in the georeferenced world. The environment and its GIS-driven world-generation pipeline are complete. What remains is the quantitative characterization of policy behavior within it, which is identified as future work in chapter 6.

C.1 Articulated Vehicle Model

The articulated vehicle spawned into the georeferenced worlds is the `electrans` tractor-trailer, shown in fig. C.3. It is defined in the conventional ROS/Gazebo manner as a URDF assembled from `xacro` macros, rather than as a hand-written SDF model. The tractor shares

its entire `xacro` description with the AgileX Hunter model used elsewhere in the project. The same parameterized macros define an Ackermann steering geometry (a pair of revolute front-steering links bounded by the maximum steering angle), continuous-rotation rear wheels coupled by a mimic joint, the link inertials, and the Gazebo friction and `ros2_control` blocks. The two vehicles differ only in scale and in the body mesh. The Hunter's compact chassis mesh is replaced by a cab-over terminal-tractor body, and the model is scaled accordingly. Reusing a single `xacro` description in this way keeps the simulated tractor kinematically consistent with the physical Hunter platform on which the policies are deployed, while presenting the larger visual form of the terminal tractor this thesis targets.

The trailer is coupled to the tractor through a revolute hitch joint at the tractor rear axle, matching the hitch convention of the kinematic model in chapter 3. It is modeled as a single box body carried on one axle and reuses the same wheel meshes and inertial macros as the tractor, so that the complete articulated system is expressed through the shared macro library. Because the model is authored as standard xacro, it drops directly into both the Gazebo digital twin of this section and the wider Autoware simulation tooling without modification.

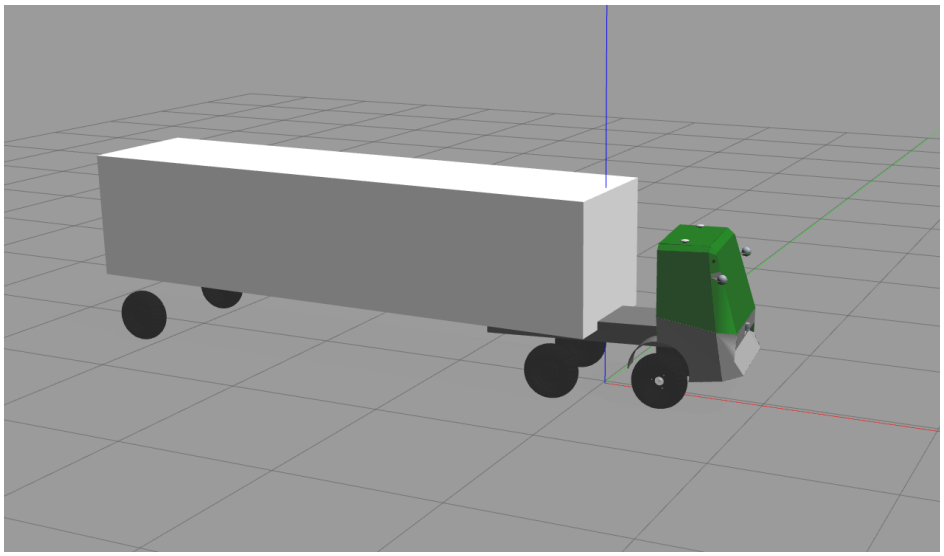


Figure C.3: The `electrans` tractor-trailer model. The tractor reuses the AgileX Hunter `xacro` description (Ackermann front steering, continuous rear wheels, and `ros2_control` blocks), rescaled and given a cab-over terminal-tractor body mesh. The box trailer is attached through a revolute hitch joint at the tractor rear axle.