

PIRL: Physics-Informed Regularization for PPO-RNN in Autonomous Obstacle Avoidance under Partial Observability

Ulan Sharipov, B.Eng.

**Submitted in fulfilment of the requirements
for the degree of Master of Science
in Electrical and Computer Engineering**



**NAZARBAYEV
UNIVERSITY**

**School of Engineering and Digital Sciences
Department of Electrical and Computer Engineering
Nazarbayev University**

53 Kabanbay Batyr Avenue,
Astana, Kazakhstan, 010000

Supervisors: Mehdi Shafiee, Galymzhan Nauryzbayev

April 2026

DECLARATION

I hereby declare that this manuscript, entitled “PIRL: Physics-Informed Regularization for PPO-RNN in Autonomous Obstacle Avoidance under Partial Observability”, is the result of my own work except for quotations and citations which have been duly acknowledged.

I also declare that, to the best of my knowledge and belief, it has not been previously or concurrently submitted, in whole or in part, for any other degree or diploma at Nazarbayev University or any other national or international institution.

Name: Ulan Sharipov

Signature:

A handwritten signature in black ink, consisting of a stylized 'U' followed by a loop and a vertical stroke.

Date: April 7, 2026

Contents

1	Abstract	8
2	Introduction	9
2.1	Background and Motivation	9
2.2	Conventional Navigation Algorithms	9
2.2.1	Global Path Planning	10
2.2.2	Local Path Planning	12
2.3	Reinforcement Learning in Navigation	14
2.4	Problem Statement	15
2.5	Research Objectives	17
2.6	Thesis Layout	18
3	Literature Review	19
3.1	Reinforcement Learning in Navigation	19
3.1.1	Proximal Policy Optimization	23
3.1.2	Sim-to-real	25
3.2	Physics-Informed Neural Networks in Autonomous Navigation	25
3.3	Combination of Reinforcement Learning and Physics-Informed Neural Networks in Autonomous Navigation	27
3.4	Research Gap	29
4	System Design	30
4.1	System Hardware Setup	30

4.2	Methodology	33
4.2.1	Robot approximation in simulation software	33
4.2.2	Modeling of an environment in IsaacSim software	35
4.2.3	Observation Space	39
4.2.4	Action Space	42
4.2.5	Reward function design	43
4.3	Reinforcement Learning Formulation	45
4.3.1	POMDP and Objective	45
4.3.2	PPO with Recurrent Policy	46
4.3.3	Value Function and GAE	46
4.3.4	Sequence Training	47
4.4	Model Architecture	47
4.5	Total Loss Function	50
4.5.1	Training setup	52
5	Results and Discussion	54
5.1	Experimental Design	54
5.2	Main Task-Level Outcomes	54
5.3	Tracking Quality and Motion Behavior	55
5.4	Termination and Safety Signals	55
5.5	Optimization Dynamics	56
5.6	Discussion	56
6	Conclusion	62
6.1	Conclusion	62
6.2	Future directions	63
A		65
A.1	Pre-Implementation Phase for Autonomous Navigation System – Es- tablishing Stable Networking Communication Infrastructure	65
A.1.1	Migration from Multicast-Based Discovery	65

A.1.2	NAT traversal Latency optimization within VPN environments	66
A.1.3	Network Security and Firewall Configuration	66
A.1.4	Maximum Transmission Unit (MTU) Alignment and UDP Packet Fragmentation	67
A.1.5	Summary of Infrastructure Readiness	67

List of Figures

4-1	The Hiwonder Jettank	31
4-2	The Hiwonder Jettank in a unified robot description format	34
4-3	The Hiwonder Jettank in a unified robot description format (i) front view, (ii) top view	34
4-4	The environment within the IsaacSim software	36
4-5	Actor Gaussian Policy Architecture(without Dynamic Head)	50
4-6	Critic Feed-Forward Architecture	51
5-1	Progress reward (Info/rew/progress). Run B rises to ~ 0.06 , while Run A remains near ~ 0.05	57
5-2	Progress ratio in total reward (Info/rew_ratio/progress). Run B approaches ~ 0.6 ; Run A remains around ~ 0.4	57
5-3	Heading reward (Info/rew/heading). Run B reaches ~ 0.035 versus ~ 0.025 for Run A.	58
5-4	Path-error reward (Info/rew/path_error). Less negative values in Run B indicate lower cross-track penalty.	58
5-5	Per-step total reward (Info/rew/total). Run B remains positive (~ 0.05 – 0.075); Run A is near -0.025 to 0	59
5-6	Mean episodic return (Reward/Total reward mean). Run B: ~ 30 – 40 ; Run A: ~ -20 to 0	59
5-7	Per-step success termination rate (Info/term/success). Run B reaches ~ 0.0008 , Run A ~ 0.0003	60

5-8	Per-step LiDAR collision-event rate (Info/collision/lidar). Run B shows moderate increase, consistent with more assertive obstacle negotiation.	60
5-9	Per-step collision termination rate (Info/term/collision). Collision frequency is somewhat higher in Run B, indicating a progress–safety trade-off.	61
5-10	Value loss (Loss/Value loss). Both runs converge near 0.2, indicating comparable critic optimization stability.	61

Chapter 1

Abstract

The problem of autonomous navigation in a partially observable environment that includes many local sources of noise, incompleteness, and variability remains. In this thesis we examine obstacle-avoidance in a warehouse-type environment in which there are both static and dynamic obstacles and in which the agents use a PPO based reinforcement-learning framework to select their actions.

Our approach for solving this problem included the application of three components: (1) geometry-based reward shaping; (2) A defined observation-space of local-path window characteristics and explicit geometric error measures; (3) An HJB residual as a soft-critic regularizer. We also applied two different network architectures to define our actor and critic: a recurrent Gaussian-policy and a feedforward-deterministic-value-model.

We trained the agent on two versions of the task: with and without HJB. Our results showed noticable improvement in total-reward and path-progress by including HJB into our value loss relative to the baseline. Additionally, enabling HJB improved the smoothness of our agent’s trajectories and overall control-quality while still maintaining stable training when properly scaled. Overall, these results indicate that physics-informed regularization is the primary contributor to improving performance, and can provide further benefits in the area of control-behavior.

Chapter 2

Introduction

2.1 Background and Motivation

There have been many developments in navigating Unmanned Vehicles (UAVs), but several remain. Traditional algorithms are very effective at solving the static version of the problem of avoiding obstacles, allowing for high precision global and local planning. However, they are limited as they do not account for the uncertainty associated with interactions within dynamic environments where multiple interacting objects can move unpredictably. In order to operate effectively in such environments, UAV's need to be able to perceive their environment; localize themselves; predict future states based on past experiences; and plan a path that will allow them to successfully navigate the environment while maintaining the ability to respond to new information in real time and operating with incomplete knowledge of their environment.

2.2 Conventional Navigation Algorithms

The field of navigation in robotics studies how to design a feasible path. This task usually involves solving several problems. Firstly, a global path must be generated to find a full trajectory from the robot's initial position to the target location. Next, the produced path must incorporate the obstacle avoidance within the operating trajectory. Eventually, the whole trajectory must be optimized [1, 2]. Optimization is

usually performed based on the minimization of the path length and traversal times, ensuring collision-free navigation and maintaining trajectory smoothness [3]. Classical navigation algorithms utilized in mobile robotics are typically classified into global path planning and local path planning.

2.2.1 Global Path Planning

Global path planning presents a fundamental stage of the whole navigation workflow. As the initial step of path planning, global path planning computes a feasible path from the starting point of the robot to a target point at the same time accounting for the static obstacles and any available configurations of the terrain available to the robot’s perception mechanisms [4]. The most common methods within the global path planning algorithms include but are not limited to: Rapidly Exploring Random Trees (RRT), A*, Dijkstra, Probabilistic Roadmap (PRM), etc [5, 6, 7] .

A* The A* algorithm utilizes a heuristic function to find the lowest cost path from a start node to the target node [8]. This method belongs to the graph-search algorithms. It evaluates nodes using the cost function $f(n) = g(n) + h(n)$, where $g(n)$ represents the cost from the start node to node n , and $h(n)$ is a heuristic estimate of the remaining cost from node n to the goal. The application of A* includes grid-based maps and occupancy grids and is implemented in many robotic navigation frameworks.

Rapidly-Exploring Random Tree (RRT) RRT is a sampling-based global path planning algorithm designed for motion planning in high-dimensional configuration spaces [9]. The main operating principle of RRT is that it creates a tree that expands towards unexplored regions. The underlying principle is the random sampling method. In this way, the algorithm investigates the configuration space and connects the existing nodes to the new states.

At each iteration, a random configuration x_{rand} is sampled from the configuration space X . The nearest node in the tree x_{nearest} is determined as

$$x_{\text{nearest}} = \arg \min_{x \in T} \|x - x_{\text{rand}}\|. \quad (2.1)$$

A new node x_{new} is then generated by extending the tree from x_{nearest} toward x_{rand} :

$$x_{\text{new}} = x_{\text{nearest}} + \eta \frac{x_{\text{rand}} - x_{\text{nearest}}}{\|x_{\text{rand}} - x_{\text{nearest}}\|}, \quad (2.2)$$

where η is the step size that limits the extension length. If the path between x_{nearest} and x_{new} is collision-free, the node is added to the tree. The application of RRT includes complex planning problems with large or continuous state spaces.

Dijkstra : Dijkstra’s algorithm is one of the fundamental methods. Unlike A*, it does not use a heuristic. Its main principle of operation involves computing the shortest paths from a start node to all other nodes within the graph with non-negative edge costs [10]. The algorithm expands nodes in order of increasing accumulated cost until the goal node is reached. The path cost from the start node s to a node v is defined as

$$d(v) = \min_{p \in P(s,v)} \sum_{(u,w) \in p} c(u,w), \quad (2.3)$$

where $P(s, v)$ is the set of all possible paths from s to v and $c(u, w)$ is the cost of edge (u, w) .

During the algorithm, distances are updated using the relaxation operation. For an edge (u, v) :

$$d(v) = \min(d(v), d(u) + c(u, v)). \quad (2.4)$$

In robotics, Dijkstra is typically applied to grid maps or navigation graphs derived from occupancy maps. Because it explores all nodes with lower cost before reaching the goal, it guarantees an optimal path. However, it is usually slower than A* in practice because it lacks a heuristic that directs the search toward the goal.

2.2.2 Local Path Planning

Local path planning generates short-horizon motion commands based on the robot's current state, local environment perception, and dynamic constraints, typically operating in real time for reactive obstacle avoidance [11]. In contrast to global path planning, it does not compute a complete path over the entire map but instead focuses on immediate, locally feasible motion decisions [12, 13].

Artificial Potential Field (APF): An Artificial Potential Field, also referred to as the Artificial Action field method, uses a virtual potential field to model the motion of the robot influenced by repulsive forces from the obstacles and the attractive force from the target point [14, 15]. This method involves the employment of the same principle of a particle moving in the direction towards lower potential energy [16]. Let q denote the robot position and q_g is the goal position. Mathematically, the attractive potential is formulated as:

$$U_{\text{att}}(q) = \frac{1}{2}k_{\text{att}}\|q - q_g\|^2. \quad (2.5)$$

And the repulsive potential is:

$$U_{\text{rep}}(q) = \begin{cases} \frac{1}{2}k_{\text{rep}} \left(\frac{1}{d(q)} - \frac{1}{d_0} \right)^2, & d(q) \leq d_0, \\ 0, & d(q) > d_0, \end{cases} \quad (2.6)$$

where $d(q)$ represents the distance between the robot and the obstacle, and d_0 is the obstacle influence radius.

The advantages of this method are the high computational efficiency and high suitability for real-time control. However, the associated complication involves the susceptibility to a local minima. This may happen in situations when the attractive and repulsive forces cancel. Another challenge is the vulnerability to oscillations near the obstacles. Moreover, the artificial potential field technique shows limited robustness in dynamic or cluttered environments [17].

Dynamic Window Approach (DWA): Proposed by D. Fow, W. Burgard, and S. Thrun, the Dynamic Window Approach performs local real-time collision avoidance by optimizing the feasible velocities [18]. The main operating principle is based on the selection of the control commands directly within the velocity space rather than the configuration space. The velocity is forward-simulated over a short horizon for each candidate velocity pair among the admissible velocities. The velocity is chosen by a score in the objective function maximization, which combines a measure of progress towards the goal location, a forward velocity of the robot, and the distance to the next obstacle in the trajectory. The objective function is defined as follows:

$$G(v, \omega) = \sigma(\alpha \text{ angle}(v, \omega) + \beta \text{ dist}(v, \omega) + \gamma \text{ vel}(v, \omega)). \quad (2.7)$$

Accordingly, this function evaluates candidate velocities according to three criteria relative to the robot’s current pose.

The first term, $\text{angle}(v, \omega)$, measures how well the resulting motion aligns with the direction toward the goal. It attains higher values when the robot moves directly toward the target. The second part of the equation, $\text{dist}(v, \omega)$, denotes the safety margins along the predicted trajectory, with higher values mean a safer path. Larger distances indicate safer trajectories. The third and final term, $\text{vel}(v, \omega)$, corresponds to the forward speed of the robot, favoring faster progress. The parameters α , β , and γ regulate the importance of goal alignment and velocity. The function $\sigma(\cdot)$ is applied to smooth the combined score and can bias the optimization toward safer trajectories with greater side clearance. Overall, this method balances the desire to move faster towards the goal and the desire to steer around the obstacles.

Timed Elastic Band (TEB): In its original formulation, TEB was introduced as an algorithm within the Model Predictive Control (MPC) framework, providing the parametrization and optimization structure [19]. As a local path planner, this method represents an optimization problem over a discretized trajectory [20]. The optimization is formulated as a nonlinear program:

$$V^*(\mathbf{b}) = \min_{\mathbf{b}} \sum_{k=1}^{n-1} \Delta T_k^2, \quad (2.8)$$

which is subject to robot kinematics, obstacle distances, and velocity and acceleration limits. Robot poses and time intervals are collected in a vector:

$$\mathbf{b} = [s_1, \Delta T_1, s_2, \Delta T_2, s_3, \dots, \Delta T_{n-1}, s_n]^\top. \quad (2.9)$$

The robot's pose at a discrete time step k is defined as

$$s_k = [x_k, y_k, \beta_k]^\top \in R^2 \times S^1, \quad (2.10)$$

where $x_k \in R$ and $y_k \in R$ denote the planar position of the robot, and $\beta_k \in S^1$ represents its orientation.

Since solving the constrained optimization for an online path planner presents a computational burden, the constraints are incorporated into the objective function using quadratic penalty terms, making the problem an unconstrained nonlinear least-squares optimization problem, which is then solved by the Levenberg–Marquardt algorithm.

2.3 Reinforcement Learning in Navigation

Reinforcement learning (RL) developed from ideas on optimal control, dynamic programming, and behavioral psychology [21, 22]. The mathematical foundations are based on Bellman's work, which introduced dynamic programming and the Bellman optimality equation, which formalized sequential decision making and optimal control problems [23]. These laid a theoretical basis for the Markov Decision Processes (MDPs), which is a commonly used framework in reinforcement learning [24]. The modern field of reinforcement learning began to take shape with the development of temporal-difference learning [25, 26], followed by a major breakthrough that emerged as a result of the introduction of neural networks [27]. A notable milestone was the

development of Deep Q-Networks (DQNs), demonstrating that RL agents could learn complex behaviors directly from high-dimensional sensory inputs such as images [28]. Since that time, the advanced algorithms of RL such as policy gradient methods [29], proximal policy optimization [30], and actor-critic [31] architectures have enabled reinforcement learning to be applied to robotics [32, 33, 34, 35], autonomous navigation [36, 37, 38], game playing [39, 40, 41], and control systems [42, 43].

The RL is highly applied to automated navigation [36]. RL permits agents to determine appropriate control strategies by interacting with their environment; this does not require an agent to be based upon pre-determined knowledge about how the world works. The ability of an agent to interact with its environment provides the means for the agent to respond to uncertainty, sensor noise, and changing obstacles. RL-based solutions for robot navigation have demonstrated excellent performance on many types of problems (e.g., end-to-end control, path planning, obstacle avoidance) that have shown excellent generalization abilities within diverse robotic settings. Traditional RL techniques suffer from several limitations including: needing significant amounts of training data, slow learning rates, and control task instability.

2.4 Problem Statement

The proposed project aims to develop obstacle avoidance strategies that can adapt to changing environmental conditions and provide robust decision-making capabilities. Deep reinforcement learning (RL)-based approaches have been explored for developing obstacle avoidance systems that can adapt to the dynamic conditions of a specific environment. The primary challenge lies in ensuring that the deep RL approach provides obstacle avoidance capabilities in addition to providing safety guarantees. The integration of PPO with RNNs was used to investigate the feasibility of achieving safe, efficient and adaptable obstacle avoidance in complex, dynamic environments. The main disadvantage of many RL methods is their dependence on learning policies from interacting data. However, this results in an enormous amount of time being spent in simulations over the course of days. A significant advantage of using PINNs

is that they can include prior physics based knowledge into the constraint set of the system dynamics within the neural network. Thus, the speed at which representative samples can be obtained is greatly increased as well as the speed at which the neural network can be trained. Additionally, the size of the dataset needed to train an accurate model is decreased. Another reason that PINNs are advantageous is because they can preclude generating physically impossible actions using solely data-driven models. For example, if an RL agent generates an optimal policy for its own movement, the optimal velocity or acceleration value may exceed the kinematic or dynamic limits for movement imposed by the robot’s design. For example, simulating the frictional behavior of a track wheel is very difficult using computer software. Since the differential equations describing motion and interaction with the environment can be included in the loss function of the PINN, when representative samples of actions are created using the learned optimal policy, they will follow the physics-based rules governing how the system operates.

Therefore, in summary, we propose to use PINNs as part of our framework for developing obstacle avoidance systems, since they allow us to combine prior knowledge about the physics of robot movement with machine learning techniques. This allows us to take full advantage of what each has to offer. In conclusion, we believe that a combination of PINNs and RL will enable us to create obstacle avoidance systems that are capable of adapting quickly to changes in the environment and operate safely and efficiently.

Summarizing the current limitations of existing approaches:

- Assumptions of static or slowly changing environments;
- Limited handling of dense traffic and highly dynamic obstacles;
- Insufficient long-term navigation efficiency in reactive methods;
- High computational cost and long training time of RL methods;
- Strong dependence on large interaction datasets;
- Limited sample efficiency of purely data-driven learning approaches;

- Difficulty transferring policies from simulation to real-world environments;
- Lack of explicit physical constraints in learned control policies;
- Risk of generating dynamically infeasible or unsafe actions;
- Limited integration of system dynamics and safety constraints into learning frameworks.

2.5 Research Objectives

Three major goals for this project are to create an RL framework that uses PPO to allow for local obstacle avoidance; to add a physics-informed neural network algorithm to help guide the learning process; and to test the performance of a physical robot.

Goal number one was to use PPO to create an RL-based framework for local obstacle avoidance. This goal included developing the appropriate dimensions for both the observation space and action space, creating a suitable reward function, and training a navigation policy that could produce safe velocity commands in changing environments.

In addition to creating a basic navigation policy using RL and PPO, the second goal of this research introduced a physics-informed neural network to aid the learning process. The idea behind using a physics-informed neural network is to include information about the physical world into the learning process by including environmental information and physical constraints. In doing so, researchers believe they will increase the learning stability of the RL system, make it less sensitive to sensor noise, and provide a more stable and robust output from the learned policy.

The final goal of this project is to verify how well the model developed during the simulation phase performs when transferred to a real-world robotic platform. The third goal of this project is to conduct a sim-to-real verification. The verification will determine if the model developed in the simulation phase will perform reliably as an obstacle avoidance and collision-free navigation system on a real-world robotic platform.

2.6 Thesis Layout

The thesis work is further structured in the following way:

- Chapter 2: This chapter provides a comprehensive literature review on the topics of reinforcement learning, physics-informed neural networks, and the application of both to autonomous navigation and obstacle avoidance. The literature review encompasses the state-of-the-art and the most recent advances within the field.
- Chapter 3: System Design and Methodology details the design of the physical setup, as well as the description of software configuration, model architecture, and system environment.
- Chapter 4: Experimental Setup describes the data collection process, training procedures, and testing methodologies for evaluating system performance. This chapter further summarizes the future directions for development.
- Chapter 5: This chapter presents the conclusion and future directions.

Chapter 3

Literature Review

Traditional navigation for mobile robots, autonomous vehicles, and drones typically relies upon classic planning algorithms (A^* , RRT, MPC), that have been very successful in planning motion given an appropriate representation of the robot’s dynamics, as well as an adequate representation of its environment. These methods are computationally inexpensive; however, they will be less effective when there exists either complexity in the environment or high levels of dynamism in the environment. This is because it can be difficult to obtain accurate representations of both the dynamics of the robot and its environment.

3.1 Reinforcement Learning in Navigation

Even though a reinforcement learning foundation is grounded in old mathematical and psychological concepts, its use in robotics has gained a huge amount of attention with the significant advances in computational power and gradual refinement of sensor and circuit technologies [44]. Special attention was given to integrating RL into the navigation pipeline due to the inherent uncertainty and highly unpredictable nature of this task. The summary of the papers focusing on reinforcement learning in local path planning is presented in Table 3.1.

Table 3.1: Summary of reinforcement learning approaches for obstacle avoidance and autonomous navigation.

Paper title	Overview	Sensor used	Software	Hardware
Using DRL with Automatic Curriculum Learning for Mapless Navigation in Intralogistics (2022) [45]	Develops an RL approach for mapless navigation in warehouse environments. Obstacle avoidance is modeled as a Markov Decision Process and trained using an automatic curriculum learning strategy.	Two 2D LiDARs, one front RGB camera	N/A	N/A
Sim-to-Real Transfer for Mobile Robots Using RL (2025) [46]	Presents a sim-to-real RL framework for local navigation. The learned policy directly maps LiDAR observations and robot state to velocity commands and is trained in simulation before deployment on physical robots.	2D LiDAR	NVIDIA Isaac Sim	Isaac Jetbot and TurtleBot3 Waffle (simulation); TurtleBot4 Lite (real robot)
RL-Based Dynamic Obstacle Avoidance and Integration of Path Planning (2021) [47]	Proposes a collision avoidance framework based on the Soft Actor-Critic algorithm. The method introduces the Mobile Robot Collision Avoidance Learning (MCAL) architecture and integrates RL with path planning.	LiDAR	Gazebo + Stage simulator	Industrial logistic robot SR7

Paper title	Overview	Sensor used	Software	Hardware
Learning to Drive as Humans Do: RL for Autonomous Navigation (2024) [48]	Develops an autonomous driving agent in the CARLA simulator. Visual observations from an RGB camera are encoded using a variational autoencoder to produce a latent state representation used by a Soft Actor-Critic policy.	RGB camera	CARLA simulator	N/A
Human-in-the-Loop Transfer Learning in Collision Avoidance of Autonomous Robots (2025) [49]	A human-guided pre-training phase with corrective feedback to initialize the RL policy. Claims to improve training stability and accelerate learning compared to purely autonomous RL training.	Four ultrasonic proximity sensors	–	Zumo robot platform
Benchmarking RL Techniques for Autonomous Navigation (2023) [50]	A large-scale benchmark for evaluating RL algorithms in autonomous navigation tasks. Comparison of different architectures and their performances in physical settings.	LiDAR	Gazebo + ROS + ODE engine	Clearpath Jackal

Paper title	Overview	Sensor used	Software	Hardware
CaRL: Learning Scalable Planning Policies with Simple Rewards (2025) [51]	PPO-based RL with simplified reward design. Uses multiplicative penalties for infractions, enabling large mini-batch training.	Bird’s-eye-view semantic segmentation (privileged perception input)	PyTorch, CARLA simulator, nuPlan simulator	Simulation-based autonomous vehicle platform (CARLA driving environment)
DRL-VO: Learning to Navigate Through Crowded Dynamic Scenes Using Velocity Obstacles (2023) [52]	DRL navigation policy based on the concept of velocity obstacles within the reward function to guide collision avoidance in crowded dynamic environments. The policy uses a combination of LiDAR history, pedestrian kinematics obtained from vision-based detection and tracking, and a sub-goal point to generate steering commands.	2D LiDAR, RGB-D camera	ROS, Gazebo, Stable-Baselines3 (PPO)	TurtleBot2 with Hokuyo UTM-30LX LiDAR and ZED stereo camera
CuRLA: Curriculum Learning Based Deep Reinforcement Learning for Autonomous Driving (2025) [53]	Reinforcement learning framework based on PPO n with curriculum learning and a Variational Autoencoder (VAE) for state representation. Uses traffic and collision penalties.	RGB camera	CARLA simulator, PPO, VAE	Simulation environment (CARLA Town 7)

Paper title	Overview	Sensor used	Software	Hardware
DRL for Sim-to-Real Robot Navigation with a Minimal Sensor Suite for Beach-Cleaning Applications (2025) [54]	Presents a sim-to-real deep reinforcement learning framework for autonomous robot navigation using a minimal sensor configuration. Policies based on PPO and PPO-Mask are trained in Gazebo and transferred directly to a real differential-drive robot operating on beach sand without hyperparameter retuning.	2D LiDAR, wheel encoders	ROS 2 Humble, Gazebo, Gymnasium, Stable-Baselines3	Custom four-wheel differential-drive robot (Raspberry Pi 4, ESP32)

3.1.1 Proximal Policy Optimization

PPO (Proximal Policy Optimization) is one of the most popular algorithms within the class of policy-gradient based methods that are used in robotics applications and autonomous navigation. PPO is an actor-critic type algorithm that enhances the reliability of training with respect to policy updates through either clipping the update magnitudes or using a KL divergence penalty. The advantage of this design allows for multiple epochs of optimization to take place from a single batch of on-policy experience; thus the risk of making overly large updates that can lead to unstable learning is minimized. The key PPO hyperparameters and their importance in navigation are described in Table 3.2.

Despite its popularity, PPO also presents several limitations when applied to robotic navigation. Since it is an on-policy algorithm, it generally requires a large number of environment interactions to achieve satisfactory performance. This requirement can be problematic when the training environment involves computationally expensive simulations such as Car Learning to Act (CARLA) or high-fidelity physics engines. Furthermore, empirical studies have shown that PPO performance is highly sensitive

Table 3.2: Key PPO hyperparameters influencing learning stability in navigation tasks.

Component	Parameters to Tune	Impact on Navigation Performance
Rollout length and batch size	n_steps /horizon, $train_batch_size$ /buffer, number of parallel environments	Dynamic environments and partial observability require sufficiently long trajectory fragments and stable statistics [55]. At the same time, the simulator throughput limits the feasible batch sizes.
Number of epochs and minibatches	n_epochs , minibatch size	Increasing the number of epochs improves data reuse, but also increases off-policy bias and the risk of destabilizing the learning process [30].
Clip range / KL control	ϵ (clipping parameter) or KL penalty / adaptive KL	In navigation tasks, large policy updates may lead to unsafe actions. However, too small update steps slow down learning [30, 56].
Discount factor and GAE	γ , λ (GAE- λ)	These parameters determine the bias-variance trade-off, which is especially important in long-horizon tasks with sparse penalties such as collision events [57].
Actor / critic / entropy balance	vf_coef , ent_coef	Navigation policies often converge to overly conservative but inefficient behaviors. Entropy encourages exploration, although excessive exploration may increase risk [30, 58].
Value clipping and reward scaling	vf_clip_param / value clipping, reward or advantage normalization	The value function is highly sensitive to reward scaling. Improper scaling may cause instability in the critic and degrade policy learning [59, 60, 30].
Gradient clipping	$grad_clip$ / max_grad_norm	Prevents gradient explosions that may occur due to rare events such as collisions or unusual environment configurations [59].

to implementation details, including reward scaling, advantage normalization, batch size, and clipping parameters. Small changes in these parameters can significantly affect convergence and final performance, highlighting the importance of careful hyperparameter tuning and environment design.

3.1.2 Sim-to-real

One of the most substantial challenges of RL in general is the sim-to-real transfer problem. This is a common case when a perfect simulation-based training experiences a performance degradation when exposed to real-life environments [61, 62]. This is mainly attributed to the perception problem and environmental conditions, but it may also extend to discrepancies in physical setups [63].

Some of the mitigation strategies include domain randomization [64] and curriculum learning [65]. Besides, researches propose validating the performance using sim-to-sim - a cross-validation strategy. In this way, policies are trained in one simulator and validated in another upon deployment on hardware. This was reported to soften the sim-to-real transfer and has become a common experimental protocol to evaluate generalization properties of the model.

3.2 Physics-Informed Neural Networks in Autonomous Navigation

Physics Informed Neural Networks (PINNs) are a class of neural networks that incorporate physical laws and physical equations. In this way, the model doesn't learn purely from data, but becomes constrained by physics. They are especially useful when working with noisy or limited data or when problem setup has a lot of unknown parameters. This is the main reason they fit the autonomous navigation problem and other tasks requiring the modeling of system dynamics. Table 3.3 provides insights into how the PINNs are typically integrated into navigation pipelines.

Table 3.3: PINN-centric and PDE/energy-constrained neural integrations with RL for navigation and obstacle avoidance

Paper	Physics integration type	PINN architecture / physics constraints	Environment & obstacle types
OA-PINN: Efficient Obstacle Avoidance for Autonomous Vehicle Safety with Physics-Informed Neural Networks (2024) [66]	CBF-based safety, HJB/PDE-informed obstacle-avoidance via PINN (OA-PINN)	Physics-informed NN constrained by safety priors (CBF); HJB/PDE-based formulation solved/approximated by PINN	Obstacle avoidance for autonomous vehicles; static and dynamic obstacles; LiDAR point-cloud based (no SLAM); validated via simulation and hardware experiments
MAD-PINN (2025) [67]	A Physics-Informed ML Framework for Safe and Optimal Control of Autonomous Systems (2025)	PINN-style training of $V(\cdot)$ via PDE residual and boundary loss; safe policy induced via Hamiltonian minimization	Simulated navigation: boat navigation; pursuer-evader with obstacles; multi-agent navigation (dynamic agents, static obstacles)
Physics-informed MARL for Distributed Multi-Robot Problems (2025) [68]	Port-Hamiltonian policy constraints, attention for graph neighbors	Port-Hamiltonian dynamics constraints; self-attention predicts controller terms; energy-structured policy	Multi-robot scenarios; dynamic obstacles/agents; real robot validation
Physics-informed IRL for Real-world Driving (2024) [69]	Port-Hamiltonian neural network as physics refinement module	PHNN dynamics (energy flow, dissipation/forcing) refine transformer outputs	Driving (traffic, dynamic obstacles)

Paper	Physics integration type	PINN architecture, physics constraints	Environment & obstacle types
Physics-informed Value Learner for Offline GCRL (2025) [70]	Eikonal PDE regularizer for value geometry	Eikonal PDE-derived regularizer on goal-conditioned value function	Large-scale navigation/maze (mostly static obstacles)
Physics-Informed Deep RL for Aircraft Conflict Resolution (2021) [71]	Physics-informed observation encoding (SSD conflict cones)	physics-informed input encoding grounded in conflict geometry	Air traffic conflict resolution; dynamic intruders
Safe & Efficient Multi-Agent Collision Avoidance with PIRL (2024) [72]	APF regularization, constrained safety optimization	PF loss in critic; safety layer via constrained optimization	Multi-agent collision avoidance; simulation and real-world claims

3.3 Combination of Reinforcement Learning and Physics-Informed Neural Networks in Autonomous Navigation

PPINNs incorporate physical laws, typically expressed as differential equations, directly into the training objective of neural networks. By enforcing these physical constraints, PINNs can improve model generalization and reduce reliance on large datasets. In the context of reinforcement learning, physics priors may be incorporated through regularization terms, physics-aware dynamics models, or constraints on actions and system dynamics. While there is a very limited number of papers focusing on the incorporation of PINNs to RL, one notable example of combining physics-informed learning with reinforcement learning is the HJBPO approach, in which the value function is formulated as a physics-informed neural network satisfying the Hamilton–Jacobi–Bellman (HJB) equation [73]. This formulation links the theoretical foundations of optimal control with modern deep reinforcement learning algorithms such as PPO. Experimental results on benchmark control tasks demonstrate potential improvements in performance and stability compared with

standard PPO implementations. However, applying such methods to complex navigation problems remains challenging due to the difficulty of accurately modeling the dynamics of multi-agent environments and human behavior.

Table 3.4: Comparison of physics-informed reinforcement learning approaches for obstacle avoidance in dynamic environments

Paper	Physics Integration Type	RL Method	Environment and Obstacles	Observations and Sensors
[67]	PINN-style: HJB-PDE residual with conformal verification	SC-OCP based value learning induced policy; comparison with C-SAC	Navigation with obstacles; dynamic pursuer–evader and obstacles	Full-state (controlled states), failure set as obstacles
[71]	Physics-informed representation: SSD role and imaginary data	PPO (actor–critic with CNN)	Aerial conflicts: intruders as dynamic obstacles	SSD images from position, velocity, and heading
[68]	Physics-structured policy: port-Hamiltonian with attention	SAC (self-attention pH policy)	Multi-robot navigation and avoidance; real Robotarium environment	Local state, neighbor information via graph
[69]	Physics refinement: PHNN energy-flow with sequence RL	Online imitative RL and Decision Transformer pipeline	Real driving: dynamic traffic, collisions, off-road	Scene: obstacles, map, traffic; goal-conditioned
[70]	PDE regularization: Eikonal-regularized value	Offline goal-conditioned RL (HIQL variants)	Navigation, goal-reaching; obstacle details limited (poster/abstract)	N/A

Paper	Physics Integration Type	RL Method	Environment and Obstacles	Observations and Sensors
[72]	Physics-informed loss in critic + constrained optimization for safe actions	Physics-informed PIRL framework	Multi-agent collision avoidance; simulation and real experiments	Local agent observations

3.4 Research Gap

Despite the advances in the field, the combination of both physics and reinforcement learning is still an emerging field. However, even given a small number of research studies, PINNS-based RL local navigation presents a promising direction. Evidence suggests that these two technologies are mutually complementary: while RL teaches the model to adapt and make decisions under uncertainties and complex conditions, PINNs provide insights into underlying physical laws, introduce structure, and ensure safety.

A full pipeline integration of the hybrid framework, incorporating both PINNS and RL for dynamic obstacle avoidance, remains an open research problem which is largely unexplored. This, along with the small evidence of broadening the research from simulation settings to real-world physical setups, brings out many possibilities for the scientific contributions.

This thesis aims to address this research gap by providing a full pipeline framework featuring PPO-based RL with integration of physics-informed neural networks inside the value network.

Chapter 4

System Design

4.1 System Hardware Setup

Mobile Robot Platform and Locomotion

The system hardware setup uses a HiWonder tank-type mobile robot designed for ground-based autonomous navigation experiments. The robot employs a tracked locomotion system with two continuous tracks, each driven by an independent electric motor, enabling differential non-holonomic drive motion. This type of locomotion is very useful for its stability in motion. Furthermore, since it allows for direct differentiation, the control of the translational and rotational velocities is easier than other platforms (omni- and mecanum wheels, ackerman steerings, etc.). Moreover, direct decoupling between rotational and translational velocity presents a simplified kinematic model and allows for the zero-radius steering. The smaller number of control inputs (only 2 motors) and a reduced number of geometrical constraints further simplify the task from the control perspective.

The robot presents a tracked ground vehicle equipped with an embedded computing unit and multiple perception sensors enabling autonomous navigation and SLAM capabilities. The main hardware characteristics of the platform are summarized in Table 4.1.

Proprioceptive Sensors: The robot is equipped with incremental wheel encoders mounted on the drive motors, which provide odometry measurements used for motion estimation. Besides, an inertial measurement unit (IMU) is also installed on the platform.

Table 4.1: Main hardware characteristics of the Hiwonder JetTank robot platform.

Property	Value
Type	Tracked mobile robot (differential track drive)
Dimensions	$310 \times 246 \times 265$ mm
Weight	2.9 kg
Chassis material	Aluminum alloy metal frame
Drive system	Dual motor tracked drive
Controller	NVIDIA Jetson Orin
Operating system	Ubuntu with ROS support
Battery	11.1 V 6000 mAh Li-Po
Camera	Orbbec Astra Pro Plus depth camera
Display	7-inch LCD touch screen



Figure 4-1: The Hiwonder Jettank

Exteroceptive Perception Suite: Environmental perception is achieved using a combination of vision- and range-based sensors. An Orbbec RGB-D camera provides synchronized color and depth information for obstacle perception, visualization, and environment understanding. Additionally, the robot is equipped with a two-dimensional LIDAR sensor, which supplies planar range measurements used for localization, mapping, and navigation within the ROS 2 framework.

Embedded Computing Unit: All sensing, control, and learning components are executed on an NVIDIA Jetson Orin onboard computing platform running Ubuntu Linux and ROS 2. The Jetson Orin provides the computational resources required for real-time perception, mapping, and execution of learning-based navigation algorithms. Sensor data acquisition, motor control, and inter-process communication are handled through the ROS 2 middleware, enabling modular integration of perception, planning, and control components.

This hardware configuration provides a stable and reproducible experimental platform for evaluating autonomous navigation in dynamic environments. For environmental perception and mapping, the platform is equipped with a two-dimensional LiDAR sensor. This sensor provides full 360° scanning of the surroundings and serves as the primary input for localization and mapping algorithms. The main LiDAR specifications used in this work are listed in Table 4.2.

Table 4.2: LiDAR sensor specifications used in the JetTank platform.

Property	Value
LiDAR model	SLAMTEC A1
Scan angle	360°
Measurement range	0.15 – 12 m
Scan frequency	~5.5 Hz
Angular resolution	~0.9°
Distance accuracy	±1%

4.2 Methodology

4.2.1 Robot approximation in simulation software

The initial step toward developing the environment where the robot will operate has been to transfer the physical robot as a unified robot description format (URDF) definition into the simulation environment. The URDF is an XML-based format that defines the kinematic and visual aspects of a robot in ROS. A URDF file describes all of the physical parts of the robot with links representing solid bodies and joints describing how these are physically connected. Other elements may include collision geometry, meshes representing the robots' appearance, inertia values, and mounting points for various sensors so that both the physical structure and motion constraints of the robot can be accurately replicated by the simulator. The details of the pre-implementation and preparatory procedures, including networking infrastructure setup, ROS 2 communication configuration, VPN optimization, firewall configuration, and MTU alignment, are provided in Appendix A.

This study created the URDF model based upon the physical configuration of the robot platform. Individual links were created to represent each part of the robot's chassis, sensors, and movable components. Appropriate joint types (i.e., fixed or rotational) were applied to define the mechanical interconnects between links. Visual and collision geometries were associated with each link to provide an accurate representation of the robot's physical configuration when simulated and checked for collisions.

However, for an efficient training procedure, necessary simplifications were made within the Isaac Sim software to reduce the computational cost of modeling the tracked wheel physics. The tracked drivetrain was approximated as a computationally efficient **differential-drive** system using two actuators: `left_wheel_joint` and `right_wheel_joint`. This was achieved via the `ImplicitActuatorCfg` with zero stiffness and high damping, effectively implementing a velocity control regime where the robot's movement mimics a differential-drive mechanism. The robot, loaded in URDF format, was configured to be a free-moving rigid body with zero damping, ensuring responsiveness to the applied actuator forces. The robot model in URDF format within the simulation environment is shown in Figure 4-2. Additional front and top views of the robot are presented in Figure 4-3.



Figure 4-2: The Hiwonder Jettank in a unified robot description format

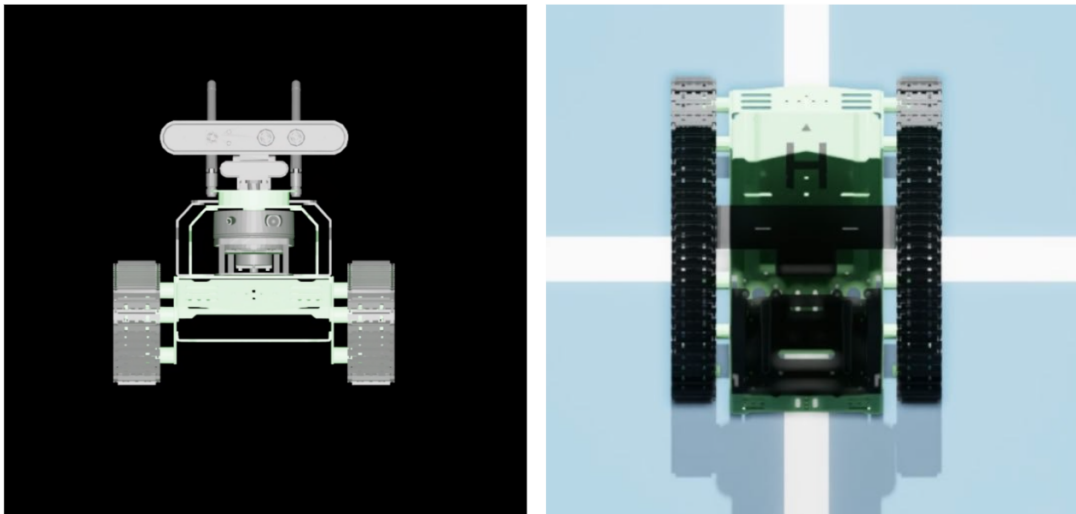


Figure 4-3: The Hiwonder Jettank in a unified robot description format (i) front view, (ii) top view

4.2.2 Modeling of an environment in IsaacSim software

Software Configuration

The simulation environment was set up using Isaac Sim 5.1.0, deployed via Docker container for consistency. A template generator script (`./isaacclab.sh -new`) configured dependencies by installing the `skr1` package, which provided the RL framework with ready building tools for the PPO algorithm. Figure 4-4 shows the warehouse environment used in Isaac Sim.

Scene Selection and Development

To imitate a real-world, pre-built Isaac Sim scene was chosen. It consists of a small warehouse with forklifts. Previously, a scene was created from scratch, which raised a lot of design questions that is outside of the current thesis scope.

Development Constraints and Implementation Scope

Moreover, to improve generalization, a procedural scene generation mechanism using the "SceneBlox" extension was considered. However, due to the rapid deprecation of software packages, implementing such a generation required a lot of effort to perform technical debugging because of incompatibility issues. There was also the idea of adding walking people animation in the simulation, but due to the same issues, it was ignored for the sake of simplicity.

Static Environment and Obstacle Layout

The environment consists of a flat industrial floor surface populated with several static objects that serve as obstacles. These include wooden pallets, cardboard boxes, stacked cargo, and warehouse equipment such as a forklift. The arrangement of these elements creates a cluttered layout with partially obstructed pathways and varying free-space widths. Such a configuration introduces realistic navigation constraints, requiring the robot to continuously adapt its trajectory while moving through the environment.

The placement of obstacles intentionally forms narrow passages and irregular obstacle distributions. This structure forces the robot to perform local obstacle avoidance while maintaining progress toward a navigation goal. The presence of different object shapes and sizes also produces diverse geometric features that must be detected by lidar and interpreted

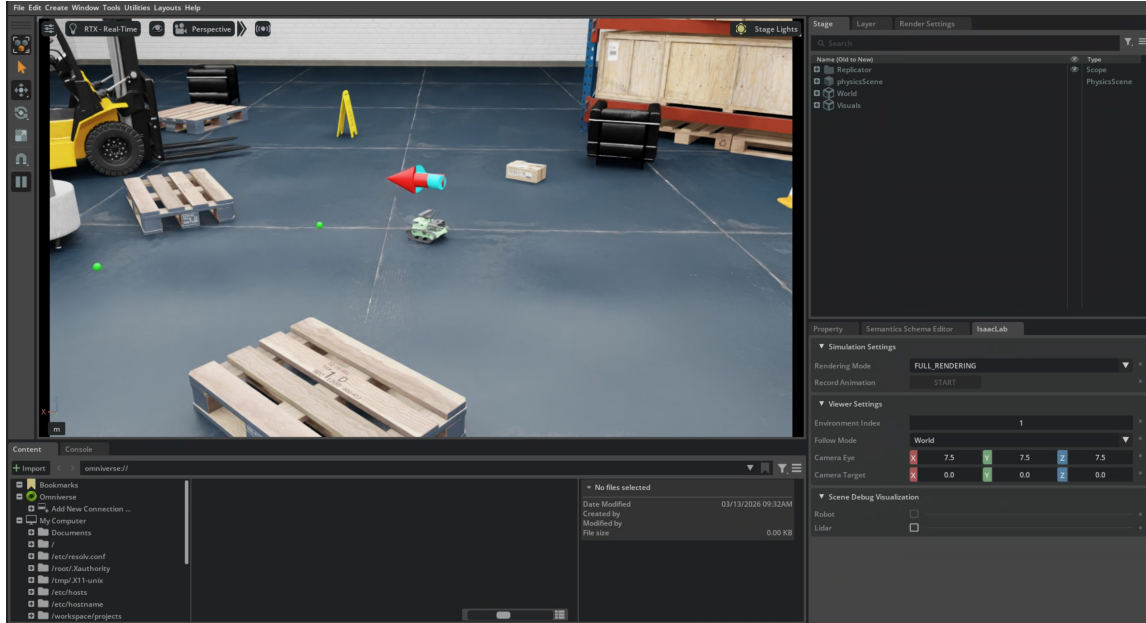


Figure 4-4: The environment within the IsaacSim software

by the robot’s perception system. Moreover, having diverse set of assets on simulation decrease chances of overfitting.

Procedural Clutter and Dynamic Agents

At the beginning of each episode, a random number of additional objects such as racks, pallets, cardboard boxes, traffic cones, and warning signs were placed at different positions and orientations inside the warehouse. Their placement was constrained by minimum separation distances and keep-out zones around the robot spawn area, which ensured both sufficient navigable space and meaningful obstacle diversity. As a result, the robot encountered different clutter distributions involving moving obstacles across episodes while still operating within the same warehouse-type context.

Dynamic obstacles were spawned from a separate asset set and followed deterministic local trajectories around sampled anchor points, producing moving obstacles with different sizes, shapes, motion radius, and angular velocities. This was done to simulate the non-stationary nature of real industrial environments, where robots must navigate not only around fixed infrastructure and stored goods, but also around moving entities and temporarily displaced objects. Important to note that despite the perceived randomness of human-populated environments, real-world motion traces usually follow a narrow, multi-modal distribution.

Table 4.3: Key environment and simulation parameters used in the Isaac Sim navigation environment.

Parameter	Value	Description
Simulation timestep (dt)	1/120 s	Physics simulation update interval.
Control frequency	60 Hz	Achieved through decimation factor of 2.
Episode length	15 s	Maximum duration of one training episode.
Number of parallel environments	10	Environments simulated simultaneously for faster RL training.
Maximum linear velocity	0.5 m/s	Upper bound of robot forward velocity.
Maximum angular velocity	3 rad/s	Maximum rotational velocity of the robot.
Wheel radius	0.03 m	Radius used for differential drive kinematics.
Track width	0.242 m	Distance between left and right wheels.
LiDAR field of view	$[-100^\circ, 100^\circ]$	Horizontal sensing range.
LiDAR angular resolution	1°	Angular spacing between adjacent rays.
LiDAR maximum range	18 m	Maximum measurable distance.
Costmap size	5 m $\times$ 5 m	Local grid map centered on the robot.
Grid resolution	0.05 m	Size of each costmap cell.
Inflation radius	0.55 m	Distance used for obstacle cost inflation.
Costmap history length	4 frames	Number of past grids stacked for temporal context.
Path Length	6 m	Planned path length from start.
Path point spacing	0.1 m	distance between consecutive path points
Path goal treshold	0.4 m	Distance for considering a final point as reached.
Dynamic obstacles per episode	12–24	Moving obstacles generated during simulation.

This implies that dynamic agents often move along well-defined paths rather than in a completely erratic manner. In the following work, the motion of a dynamic obstacle will be approximated with curvilinear trajectory.

Local Path Management and Target Generation To imitate global path planning behavior similar to ROS2's NavStack package, separate path planning module was implemented. For the sake of simplicity it randomly generates a path of fixed distance inside the warehouse that the robot should follow. Since obstacle are mostly non-static, there is a particular chance that at some time during the episode, the path will not intersect any obstacles. However, there are still static obstacle such as a forklift coming from the warehouse scene itself (it is baked), that may produce situations where path is blocked by a static obstacle. The thesis is only concerned with local obstacle avoidance, but further development would require improvement of the global path planning module to account for such edge cases.

During execution, the module performs three primary logical operations:

- **Geometric Projection:** The system continuously projects the robot's current position onto the active path segment. This allows for an calculation of the signed cross-track error (how far the robot has deviated laterally) and the cumulative arc-length distance traveled along the route.
- **Monotonic Progress Tracking:** Robot can only perform a one-way progression along the path, preventing "backward jumps" in the target index. Once the robot clears a specific section, the system locks that progress and shifts focus to the subsequent segments, giving a consistent flow toward the final destination.
- **Predictive Look-ahead:** The module calculates a heading signal as a tangential to a look-ahead target further along the path. In the implementation look-ahead point is taken as furthest point on path point segment that is provided to agent from state. This provides the policy with a "preview" of upcoming curvature, enabling the robot to proactively adjust its velocity and orientation before reaching a turn.

4.2.3 Observation Space

The policy observation is a two-branch dictionary collected at each RL step:

$$\mathcal{O}_t = \{\mathbf{vec}_t, \mathbf{costmap}_t\},$$

where $\mathbf{vec}_t$ is a dense kinematic-path-memory vector (with LiDAR sector-hit features), and $\mathbf{costmap}_t$ is a stacked Nav2-style local grid built from LiDAR scans.

1) Vector branch (vec)

The vector observation is

$$\mathbf{o}_t^{\text{vec}} \in R^{d_{\text{vec}}}, \quad d_{\text{vec}} = 2 + 2 + 2N + 2K + 2 + d_r,$$

with $N = \text{path_segment_len} = 12$, $K = \text{hjb_lidar_sector_count} = 16$, and $d_r = \text{reward_component_dim} = 6$. Hence:

$$d_{\text{vec}} = 2 + 2 + 24 + 32 + 2 + 6 = 68.$$

- Previous action and reward components are included as a short memory signal for partial observability; this is not required by a strictly Markovian formulation but is empirically useful in recurrent RL settings [74].
- The path window uses a fixed-length local segment of N points represented in the robot frame.

$$\mathbf{o}_t^{\text{vec}} = \left[\underbrace{v_{x,t}, \omega_{z,t}}_2, \underbrace{d_t, \psi_t}_2, \underbrace{\mathbf{P}_t^{\text{local}}}_{2N}, \underbrace{\mathbf{L}_t^{\text{sector}}}_{2K}, \underbrace{\mathbf{a}_{t-1}}_2, \underbrace{\mathbf{r}_{t-1}^{\text{comp}}}_6 \right].$$

Cross-track and heading error features.

$$d_t \in R, \quad \psi_t \in [-\pi, \pi].$$

Here, d_t is the signed cross-track error (signed distance from robot position to its projection on the path), and ψ_t is the wrapped heading error between robot yaw and the lookahead/path

heading target. These terms are also used by the HJB regularization state.

Ego-motion. $(v_{x,t}, \omega_{z,t})$ are body-frame forward linear speed and yaw rate.

Local path segment (resampled sliding window).

$$\mathbf{P}_t^{\text{local}} = \{(x_i^b, y_i^b)\}_{i=1}^N$$

is a fixed-size window sampled along the current path by arc length and expressed in `base_link`. For a world-frame path sample p_i^w and robot pose (r_t^w, θ_t) :

$$\Delta p_i^w = p_i^w - r_t^w, \quad \begin{bmatrix} x_i^b \\ y_i^b \end{bmatrix} = R(-\theta_t) \begin{bmatrix} \Delta x_i^w \\ \Delta y_i^w \end{bmatrix}.$$

If path length is insufficient, the final point is repeated/clamped.

LiDAR sector-hit features.

$$\mathbf{L}_t^{\text{sector}} = \{(x_k^b, y_k^b)\}_{k=1}^K.$$

The LiDAR FOV is split into K equal angular sectors. In each sector, the nearest valid range r_k is selected, then converted to body-frame coordinates:

$$x_k^b = r_k \cos \phi_k, \quad y_k^b = r_k \sin \phi_k.$$

These features expose obstacle geometry directly to the actor and to the extended HJB dynamics.

Previous action and reward decomposition.

$$\mathbf{a}_{t-1} = [a_{t-1}^{\text{lin}}, a_{t-1}^{\text{yaw}}],$$

$$\mathbf{r}_{t-1}^{\text{comp}} = [r_{t-1}^{\text{progress}}, r_{t-1}^{\text{path}}, r_{t-1}^{\text{heading}}, r_{t-1}^{\text{prox}}, r_{t-1}^{\text{col}}, r_{t-1}^{\text{rev}}].$$

Reward components are clipped before insertion into `vec` (implementation: $[-1, 1]$).

2) Spatial branch (costmap)

The costmap tensor is provided in CNN format:

$$\mathbf{o}_t^{\text{costmap}} \in R^{C \times H \times W},$$

with

$$C = 6, \quad H = W = 100.$$

At resolution 0.05 m/cell, this is a 5×5 m egocentric window.

Construction from LiDAR scans. At each update:

- LiDAR hits are converted to ranges $d_{t,j}$; non-finite values are replaced by $d_{\max}$, then clamped.
- Ray tracing marks free cells along beams as `grid_free_cost`.
- Beam endpoints inside the grid mark occupied cells as `grid_lethal_cost`.
- Nav2-style inflation is applied around lethal cells:

$$c_{\text{infl}}(x) = c_{\text{inscribed}} \exp(-\kappa d(x)),$$

within inflation radius r_{infl} , respecting lethal/unknown overwrite rules.

Temporal stacking. History length is 3 frames (newest, previous, oldest), with a new frame pushed every `grid_history_interval_steps` = 4 env steps. Each history frame is encoded as two channels:

`cost, known_mask`.

So the final channel order is:

`[newest_cost, newest_known, prev_cost, prev_known, oldest_cost, oldest_known]`,

therefore $C = 3 \times 2 = 6$.

Normalization. For each cell with Nav2-style integer cost c :

$$\text{cost} = \begin{cases} 0, & c = 255 \text{ (unknown)} \\ c/254, & c \neq 255 \end{cases}, \quad \text{known_mask} = \begin{cases} 0, & c = 255 \\ 1, & c \neq 255. \end{cases}$$

Thus both channels are in $[0, 1]$, and unknown space is represented explicitly by **known_mask** = 0.

4.2.4 Action Space

The policy generates a continuous two-dimensional action vector $\mathbf{a} \in [-1, 1]^2$ sampled from the probability distribution of the actor network. These normalized outputs are mapped to high-level kinematic commands (v, ω) , which are subsequently transformed into wheel-level velocities via differential drive robot kinematics.

1. **Velocity Scaling** The normalized actions are scaled to the robot’s physical limits using a linear mapping:

$$v = a_0 \cdot v_{\max}, \quad (4.1)$$

$$\omega = a_1 \cdot \omega_{\max}, \quad (4.2)$$

where $v_{\max} = 0.5 \text{ m/s}$ and $\omega_{\max} = 3.0 \text{ rad/s}$.

2. **Differential Drive Inverse Kinematics** To command the robot’s actuators, the target velocities (v, ω) are converted into angular velocities for the left and right wheels $(\dot{\phi}_L, \dot{\phi}_R)$:

$$\begin{bmatrix} \dot{\phi}_L \\ \dot{\phi}_R \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -L/2 \\ 1 & L/2 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}, \quad (4.3)$$

where:

- r is the wheel radius.
- L is the track width (distance between wheels).

This formulation ensures the policy remains bounded and robot-agnostic, while providing the necessary low-level setpoints for the joint velocity controllers.

4.2.5 Reward function design

To obtain stable path-following behavior in cluttered environments, the reward is defined from path geometry rather than from a moving look-ahead target.

A moving look-ahead point is not suitable as a progress objective because it introduces non-stationarity: the robot is rewarded for approaching a target that is itself changing. To avoid mixing robot progress with target-update logic, progress is measured directly along the reference path arc length.

Let the reference path be represented as a polyline

$$\mathcal{P} = \{p_i\}_{i=0}^{N-1}, \quad p_i \in R^2,$$

with cumulative arc-length coordinates

$$s_0 = 0, \quad s_i = s_{i-1} + \|p_i - p_{i-1}\|, \quad i \geq 1.$$

At time step t , given robot position $x_t \in R^2$, the nearest admissible anchor index k_t is selected on the unconsumed path suffix (monotonic pruning), i.e., $k_t \geq k_{t-1}$. A local segment $[p_{k_t}, p_{k_t+1}]$ is used to compute continuous geometric quantities.

For $v_t = p_{k_t+1} - p_{k_t}$, projection factor τ_t is

$$\tau_t = \text{clip}\left(\frac{(x_t - p_{k_t})^\top v_t}{\|v_t\|^2}, 0, 1\right).$$

The projected point on the segment is

$$\hat{p}_t = p_{k_t} + \tau_t v_t.$$

The continuous progress coordinate is

$$\tilde{s}_t = s_{k_t} + \tau_t \|v_t\|,$$

and the progress increment is

$$\Delta s_t = \tilde{s}_t - \tilde{s}_{t-1}.$$

The cross-track error is

$$d_{\text{path},t} = \|x_t - \hat{p}_t\|.$$

Let θ_t be robot yaw and $\theta_{\text{target},t}$ be the heading to the lookahead point (the last point of the local path window). The heading alignment term is

$$\cos(\Delta\theta_t), \quad \Delta\theta_t = \theta_t - \theta_{\text{target},t}.$$

To avoid exploiting heading reward while moving backward, heading reward is gated by forward body-frame speed:

$$g_t = \text{clip}\left(\frac{v_t^{\text{fwd}}}{v_{\text{max}}}, 0, 1\right).$$

The core path-following reward is

$$r_t^{\text{core}} = w_1 \Delta s_t - w_2 d_{\text{path},t}^2 + w_3 g_t \cos(\Delta\theta_t).$$

To account for safety constraints, two additional terms are added:

$$r_t = r_t^{\text{core}} + r_t^{\text{prox}} + r_t^{\text{col}}.$$

The proximity penalty uses LiDAR minimum range in the selected field of view:

$$\delta_t^{\text{prox}} = \max(0, d_{\text{act}} - d_t^{\text{min}}),$$

$$r_t^{\text{prox}} = -\alpha_{\text{prox}} \left(1 - e^{-\beta_{\text{prox}} \delta_t^{\text{prox}}}\right),$$

where $\alpha_{\text{prox}} > 0$ defines maximum penalty magnitude and β_{prox} controls steepness.

The collision penalty is sparse:

$$r_t^{\text{col}} = \begin{cases} w_{\text{col}}, & d_t^{\text{min}} < d_{\text{col}}, \\ 0, & \text{otherwise,} \end{cases} \quad w_{\text{col}} < 0.$$

Reverse-motion shaping is not used in the final reward:

$$w_{\text{rev}} = 0 \quad \Rightarrow \quad r_t^{\text{rev}} \equiv 0,$$

therefore it is omitted from r_t .

This design yields a reward that is (i) consistent with geometric path progress, (ii) robust to local target drift, and (iii) balanced between trajectory tracking and obstacle avoidance.

Table 4.4: Reward function parameters used during reinforcement learning training.

Reward term	Value	Description
Progress reward (w_1)	10.0	Encourages monotonic increase of arc-length progress Δs_t .
Path error penalty (w_2)	0.3	Quadratic cross-track penalty: $-w_2 d_{\text{path},t}^2$.
Heading reward (w_3)	0.05	Encourages alignment with path heading, gated by forward speed.
Proximity penalty (max)	-0.5	Exponential LiDAR-based penalty within activation distance $d_{\text{act}} = 0.9$ m.
Collision penalty (w_{col})	-60.0	Applied when minimum LiDAR range falls below collision radius.
Reverse motion penalty	disabled ($w_{\text{rev}} = 0$)	Not used in final reward expression.

4.3 Reinforcement Learning Formulation

4.3.1 POMDP and Objective

The navigation problem is modeled as a partially observable Markov decision process (POMDP), since the robot observes only a local and incomplete view of the scene (e.g., limited LiDAR field of view and local costmap), i.e. $o_t \neq s_t$. At each step, the policy outputs a continuous action

$$a_t = [a_t^{\text{lin}}, a_t^{\text{yaw}}] \in \mathcal{A},$$

and receives reward r_t from the environment.

The objective is to maximize the expected discounted return:

$$J(\theta) = E_{\pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right].$$

4.3.2 PPO with Recurrent Policy

We use PPO in an actor–critic setup. To handle partial observability, the policy is recurrent and uses a GRU memory state. Let $z_t = \phi(o_t)$ be the encoded observation, then

$$h_t = \text{GRU}(z_t, h_{t-1}).$$

The actor is conditioned on h_t :

$$\pi_\theta(a_t \mid o_{1:t}) = \pi_\theta(a_t \mid h_t).$$

The PPO probability ratio is

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid h_t)}{\pi_{\theta_{\text{old}}}(a_t \mid h_t)}.$$

The clipped surrogate objective is

$$L_{\text{clip}}(\theta) = E_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right].$$

An entropy term is added for exploration:

$$L_{\text{ent}}(\theta) = -c_{\text{ent}} E_t [\mathcal{H}(\pi_\theta(\cdot \mid h_t))].$$

4.3.3 Value Function and GAE

The critic estimates the state value:

$$V_\phi(o_t) \text{ (or } V_\phi(h_t) \text{ in recurrent critics).}$$

Temporal-difference residual:

$$\delta_t = r_t + \gamma(1 - d_t)V_\phi(o_{t+1}) - V_\phi(o_t),$$

where $d_t \in \{0, 1\}$ is the done flag.

Generalized Advantage Estimation (GAE):

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}.$$

Value regression target:

$$\hat{R}_t = \hat{A}_t + V_{\phi}(o_t).$$

The value loss is

$$L_V(\phi) = c_V E_t \left[\left(V_{\phi}(o_t) - \hat{R}_t \right)^2 \right].$$

4.3.4 Sequence Training

Training is performed on sequence mini-batches with truncated backpropagation through time (TBPTT), using rollout fragments of fixed sequence length. This stabilizes recurrent optimization and allows the policy to exploit temporal context from past observations.

This formulation keeps PPO as the primary optimization signal, while recurrent memory addresses partial observability and auxiliary terms can provide additional inductive bias when required.

4.4 Model Architecture

In the final implementation, actor and critic are parameterized by *separate* networks. The actor is a recurrent Gaussian policy, while the critic is a feed-forward deterministic value network. This design was chosen empirically: using a shared recurrent backbone degraded performance, likely because policy and value objectives induce different representation pressures in the same GRU state. A similar effect is reported in [74]. Therefore, actor and critic parameters are fully decoupled, and the critic is kept non-recurrent for simplicity and stability.

Input representation. At each step t , both networks consume the same observation:

$$o_t = [o_t^{\text{vec}}, o_t^{\text{map}}],$$

where

$$o_t^{\text{vec}} \in R^{d_{\text{vec}}}, \quad o_t^{\text{map}} \in R^{C \times H \times W}.$$

In the current setup (ObservationSchemaV2.1):

$$C = 6, \quad H = W = 100, \quad d_{\text{vec}} = 68.$$

Actor (Recurrent Gaussian Policy). The vector branch is split into a core part and an auxiliary tail:

$$o_t^{\text{vec}} = [o_t^{\text{core}}, o_t^{\text{aux}}], \quad \dim(o_t^{\text{aux}}) = 8.$$

Encoders:

$$z_t^{\text{core}} = f_{\text{core}}(o_t^{\text{core}}), \quad z_t^{\text{aux}} = f_{\text{aux}}(o_t^{\text{aux}}), \quad z_t^{\text{map}} = f_{\text{cnn}}(o_t^{\text{map}}).$$

Fusion and recurrence:

$$u_t = f_{\text{fus}}([z_t^{\text{core}}, z_t^{\text{aux}}, z_t^{\text{map}}]), \quad \tilde{u}_t = \text{LayerNorm}(u_t),$$

$$h_t = \text{GRU}(\tilde{u}_t, h_{t-1}), \quad \bar{h}_t = \text{LayerNorm}(h_t).$$

Policy head:

$$\mu_t = f_{\mu}(\bar{h}_t), \quad \log \sigma = \text{trainable parameter},$$

$$\pi_{\theta}(a_t \mid o_{1:t}) = \mathcal{N}(\mu_t, \text{diag}(\sigma^2)).$$

Critic (Feed-forward Deterministic Value). The critic uses separate encoders (no GRU):

$$\tilde{z}_t^{\text{vec}} = g_{\text{vec}}(o_t^{\text{vec}}), \quad \tilde{z}_t^{\text{map}} = g_{\text{cnn}}(o_t^{\text{map}}),$$

$$\tilde{u}_t = g_{\text{fus}}([\tilde{z}_t^{\text{vec}}, \tilde{z}_t^{\text{map}}]), \quad V_{\phi}(o_t) = g_V(\tilde{u}_t).$$

Table 4.5: Layer-wise architecture (final implementation)

Module	Layer	Input $\rightarrow$ Output	Activation
Actor: Recurrent Gaussian Policy			
Core vec branch	Linear 1	$(d_{\text{vec}} - 8) = (68 - 8) = 60 \rightarrow 64$	ELU
Core vec branch	Linear 2	$64 \rightarrow 64$	ELU
Aux vec branch	Linear 1	$8 \rightarrow 32$	ELU
Aux vec branch	Linear 2	$32 \rightarrow 32$	ELU
Costmap branch	Conv2d 1	$6 \times 100 \times 100 \rightarrow 16 \times 49 \times 49$	ELU
Costmap branch	Conv2d 2	$16 \times 49 \times 49 \rightarrow 32 \times 24 \times 24$	ELU
Costmap branch	Conv2d 3	$32 \times 24 \times 24 \rightarrow 64 \times 11 \times 11$	ELU
Costmap branch	Conv2d 4	$64 \times 11 \times 11 \rightarrow 64 \times 5 \times 5$	ELU
Costmap branch	Flatten	$64 \times 5 \times 5 \rightarrow 1600$	–
Fusion	Concatenation	$64 + 32 + 1600 = 1696$	–
Fusion	Linear 1	$1696 \rightarrow 256$	ELU
Fusion	Linear 2	$256 \rightarrow 128$	ELU
Pre-recurrent norm	LayerNorm	$128 \rightarrow 128$	–
Recurrent	GRU	$128 \rightarrow 256$	–
Post-recurrent norm	LayerNorm	$256 \rightarrow 256$	–
Policy head	Linear 1	$256 \rightarrow 128$	ELU
Policy head	Linear 2	$128 \rightarrow 64$	ELU
Policy head	Linear 3	$64 \rightarrow 2$ (mean)	–
Policy spread	Parameter	2 (log-std)	–
Critic: Feed-forward Deterministic Value			
Vec branch	Linear 1	$d_{\text{vec}} = 68 \rightarrow 64$	ELU
Vec branch	Linear 2	$64 \rightarrow 64$	ELU
Costmap branch	Conv stack + Flatten	$6 \times 100 \times 100 \rightarrow 1600$	ELU
Fusion	Concatenation	$64 + 1600 = 1664$	–
Fusion	Linear 1	$1664 \rightarrow 256$	ELU
Fusion	Linear 2	$256 \rightarrow 128$	ELU
Value head	Linear 1	$128 \rightarrow 64$	ELU
Value head	Linear 2	$64 \rightarrow 1$	–

4.5 Total Loss Function

PPO optimization is performed with a combined objective:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\pi} + \mathcal{L}_{\text{ent}} + \mathcal{L}_V + \mathcal{L}_{\text{HJB}}.$$

Policy loss (clipped PPO).

$$\mathcal{L}_{\pi} = -E_t \left[\min \left(r_t \hat{A}_t, \text{clip}(r_t, 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right], \quad r_t = \frac{\pi_{\theta}(a_t | h_t)}{\pi_{\theta_{\text{old}}}(a_t | h_t)}.$$

The Figure 4-5 presents a graphical representation of the actor architecture.

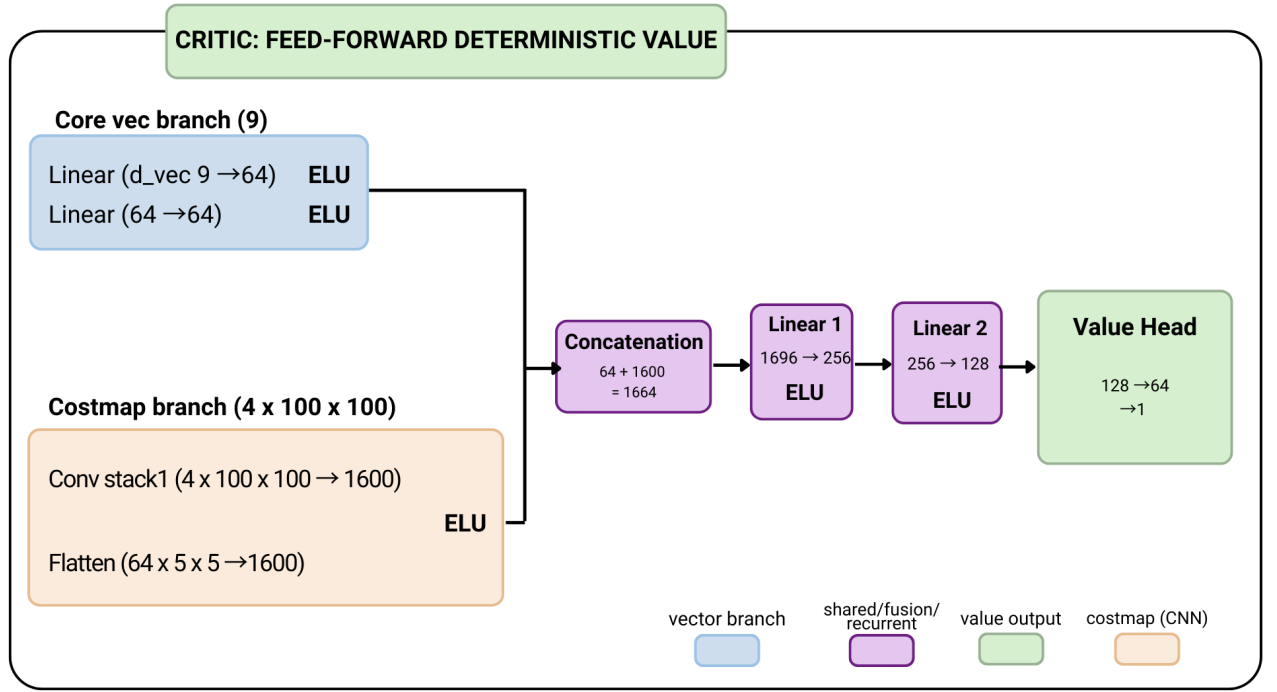


Figure 4-5: Actor Gaussian Policy Architecture(without Dynamic Head)

The Figure 4-6 presents a graphical representation of the critic feed-forward architecture.

Entropy regularization.

$$\mathcal{L}_{\text{ent}} = -c_{\text{ent}} E_t [\mathcal{H}(\pi_{\theta}(\cdot | h_t))].$$

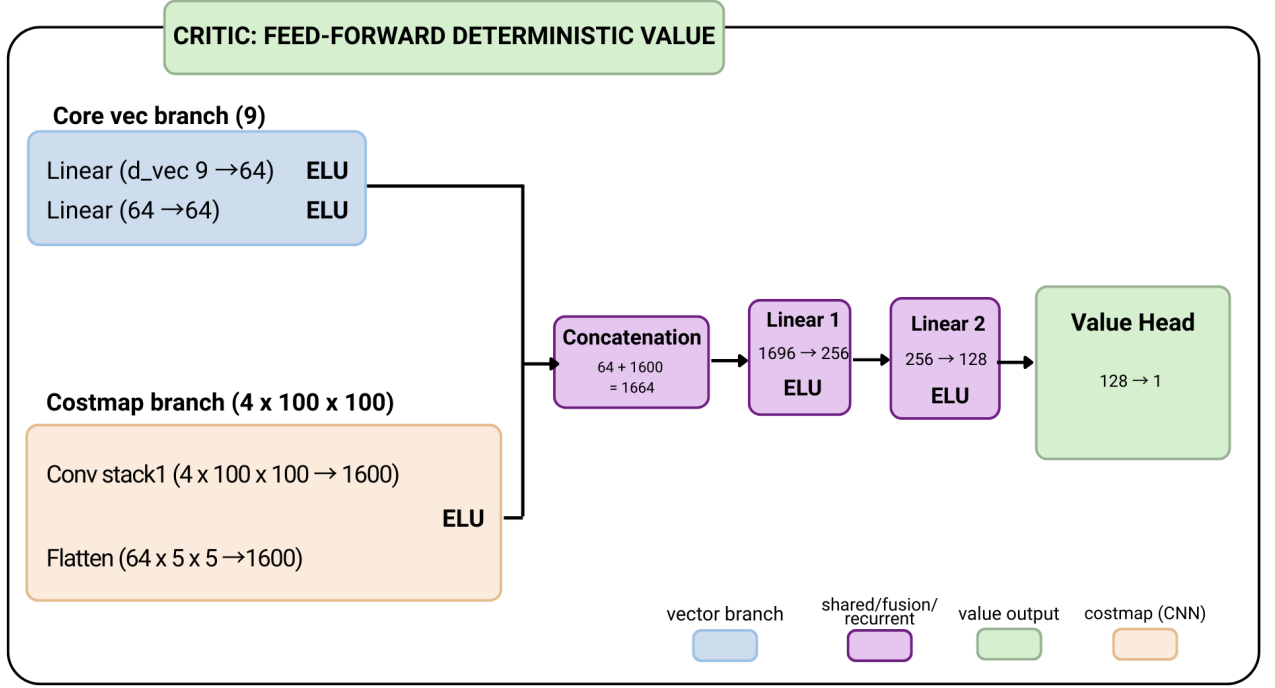


Figure 4-6: Critic Feed-Forward Architecture

Value loss.

$$\mathcal{L}_V = c_V E_t [(V_\phi(o_t) - R_t)^2],$$

with value clipping used in implementation.

HJB-PINN regularization term. In addition to the PPO objectives, we use a physics-informed soft regularizer for the critic based on a local continuous-time HJB residual (reward-max form).

For ObservationSchemaV2.1, the HJB state is

$$x_t = [d_t, \psi_t, x_{1,t}, y_{1,t}, \dots, x_{K,t}, y_{K,t}]^\top,$$

where (d_t, ψ_t) are path-tracking errors and $(x_{k,t}, y_{k,t})$ are sector-wise LiDAR hit positions in the robot body frame ($K = 16$ in current setup). Control is $u_t = (v_t, \omega_t)$ (Sec. Action Space).

The local dynamics used in the residual are:

$$\dot{d}_t = v_t \sin \psi_t, \quad \dot{\psi}_t = \omega_t,$$

$$\dot{x}_{k,t} = -v_t + \omega_t y_{k,t}, \quad \dot{y}_{k,t} = -\omega_t x_{k,t}, \quad k = 1, \dots, K.$$

Let $r_t = -\ell(x_t, u_t)$, where ℓ is the running cost (time, tracking, heading, control, and progress shaping). With discount-rate correction $\rho = -\ln(\gamma)/\Delta t$, the residual is

$$\mathcal{R}_t = r_t + \nabla_x V_\phi(o_t)^\top f(x_t, u_t) - \rho V_\phi(o_t).$$

The HJB loss is

$$\mathcal{L}_{\text{HJB}} = \lambda_{\text{HJB}} E_t[\mathcal{R}_t^2].$$

This term is used only as a *soft* critic regularizer (not a hard constraint). In the current experiments, $\lambda_{\text{HJB}} = 0$, i.e., the HJB term is ablated.

Parameter updates with separate: True. Even though actor and critic losses are summed in one optimizer step, gradient paths are separated by network parameters:

$$\theta \leftarrow \theta - \eta \nabla_\theta (\mathcal{L}_\pi + \mathcal{L}_{\text{ent}}),$$

$$\phi \leftarrow \phi - \eta \nabla_\phi (\mathcal{L}_V + \mathcal{L}_{\text{HJB}}).$$

Thus, actor and critic remain separate models with decoupled representations and objectives.

4.5.1 Training setup

All experiments use PPO with a recurrent Gaussian actor and a feed-forward deterministic value critic (**separate=True**). The training horizon is 100,000 timesteps (seed = 669). Each update is computed from rollouts of length 256, optimized for 6 epochs with 8 mini-batches. GAE is used with $\gamma = 0.99$ and $\lambda = 0.95$.

The optimizer learning rate is fixed at 3×10^{-4} (KL-adaptive scheduling is disabled in the current setup). Gradient norm clipping is set to 1.0. PPO clipping parameters are 0.2 for both policy ratio and value prediction. Entropy and value loss weights are 0.1 and 2.0, respectively. State and value targets are normalized online using **RunningStandardScaler**.

For the actor RNN, sequence length is 64, GRU hidden size is 256, and number of GRU layers is 1. The vector auxiliary tail size is **aux_dim** = 8 (used as an input split in the actor encoder, not as an auxiliary loss).

Auxiliary losses. No auxiliary dynamics loss is used in the current architecture (the previous $\mathcal{L}_{\text{dyn}}$ branch is removed).

HJB regularization configuration. The HJB regularizer is implemented in the codebase but ablated in the current experiments:

$$\lambda_{\text{HJB}} = 0.$$

Configured parameters (for ablation-ready runs) are:

$$v_{\text{max}} = 0.5, \quad \omega_{\text{max}} = 1.5, \quad (w_t, w_d, w_\psi, w_u, w_p) = (0.5, 0.5, 0.2, 0.05, 1.0),$$

with

$$(d, \psi) \mapsto (2, 3)$$

in the current `vec` layout. For `ObservationSchemaV2.1`, LiDAR-sector HJB slots are configured as:

$$\text{hjb_lidar_hits_start_index} = 28, \quad \text{hjb_lidar_sector_count} = 16.$$

Chapter 5

Results and Discussion

5.1 Experimental Design

This chapter compares two runs of the same training pipeline under identical environment and PPO settings, with only one controlled change: the HJB regularization scale.

- **Run A (blue, 2026-04-30):** $c_{\text{HJB}} = 0$.
- **Run B (orange, 2026-05-06):** $c_{\text{HJB}} = 1$.

Because all other settings are fixed, the observed differences can be attributed to enabling HJB regularization.

5.2 Main Task-Level Outcomes

Across all reward metrics, Run B has significantly better performance than Run A.

Run B’s progress reward (*Info/rew/progress*, as shown in Figure 5-1) reaches an average of about 0.06, and Run A averages about 0.05.

Additionally, Run B achieves nearly double the progress component of its total reward (as measured by *Info/rew_ratio/progress*, as illustrated in Figure 5-2), achieving about a 60% ratio; whereas, Run A achieves a much lower 40% ratio. It is clear that HJB results in a larger proportion of the agent’s optimization being used for advancing the state along the forward path (instead of just paying off penalties).

This same trend can be observed when looking at the total reward received per step (*Info/rew/total*, as depicted in Figure 5-5). The majority of the values for Run B remain between 0.05 and 0.075, while Run A sees most of its values fall within the range of -0.025 to 0.

At the level of episodes, the mean episodic return (as indicated by *Reward/Totalrewardmean*, as illustrated in Figure 5-6) is clearly greater in Run B (~ 30 -40) than it was in Run A (~ -20 to 0).

As such, this data shows that HJB does not merely improve only the individual-shaped-reward components, but rather also improves overall, end-to-end-task-usefulness.

5.3 Tracking Quality and Motion Behavior

The tracking-related reward terms that influence Run B are advantageous over Run A. Headings reward (*Info/rew/heading* in Figure 5-3) for Run B's run, as compared to Run A's run, is elevated from approximately 0.025 to approximately 0.035. Path-error reward (*Info/rew/path_error* in Figure 5-4), which is an adverse component of the reward function, has a less negative value in Run B (approximately -0.01 to -0.02), as opposed to Run A (approximately -0.02 to -0.03). The path-error reward is a penalty and therefore a smaller absolute value indicates reduced lateral error.

Collectively, an increased heading reward and a decreased path-error penalty contribute to better adherence to a desired path and better ability to correct direction. Run B's qualitative rollout assessment agrees with these numerical assessments; the policy enabled by HJB avoids obstacles sooner and performs fewer extended periods of "stop-and-wait" behavior close to obstacles.

5.4 Termination and Safety Signals

Termination and collision events are recorded at the same rate for each step in each environment, therefore termination and collision frequencies that fall within the 10^{-4} - 10^{-3} range can be anticipated and should be viewed as an event frequency rather than as a cumulative percentage.

The success-event frequency (*Info/term/success*, Figure 5-7) has increased from approxi-

mately 0.0003 to approximately 0.0008 in run B. This implies that there was an increased number of times when episodes were successfully completed.

There appears to have been a slight decrease in the average length of episodes (Episode/Total timesteps mean, Fig. ??) , which along with an increase in success and return, support the notion of completing successful episodes more quickly.

Termination and collision related frequencies (Info/Collision/Lidar and Info/Term/Collision, Figures 5-8, 5-9) appear to be greater in run B.

5.5 Optimization Dynamics

Value loss (Loss/Value loss, Fig. 5-10) follows a similar trajectory in both runs and converges near 0.2. Therefore, the behavioral performance gap is not explained by critic instability in either run. The primary difference is policy behavior: HJB regularization yields more effective control decisions for path-following under obstacle constraints.

5.6 Discussion

We see that when HJB regularization is enabled (i.e., $c_{\text{HJB}} = 1$), it shows an overall improvement over the non-HJB case:

1. It has a greater amount of progress-reward received as well as greater contributions to overall rewards due to progress;
2. The headings are aligned much better than those from the non-HJB case with less penalties paid for path tracking;
3. The episodic returns have increased significantly compared to the non-HJB case.
4. Success events per step have increased significantly.



Figure 5-1: Progress reward (Info/rew/progress). Run B rises to ~ 0.06 , while Run A remains near ~ 0.05 .

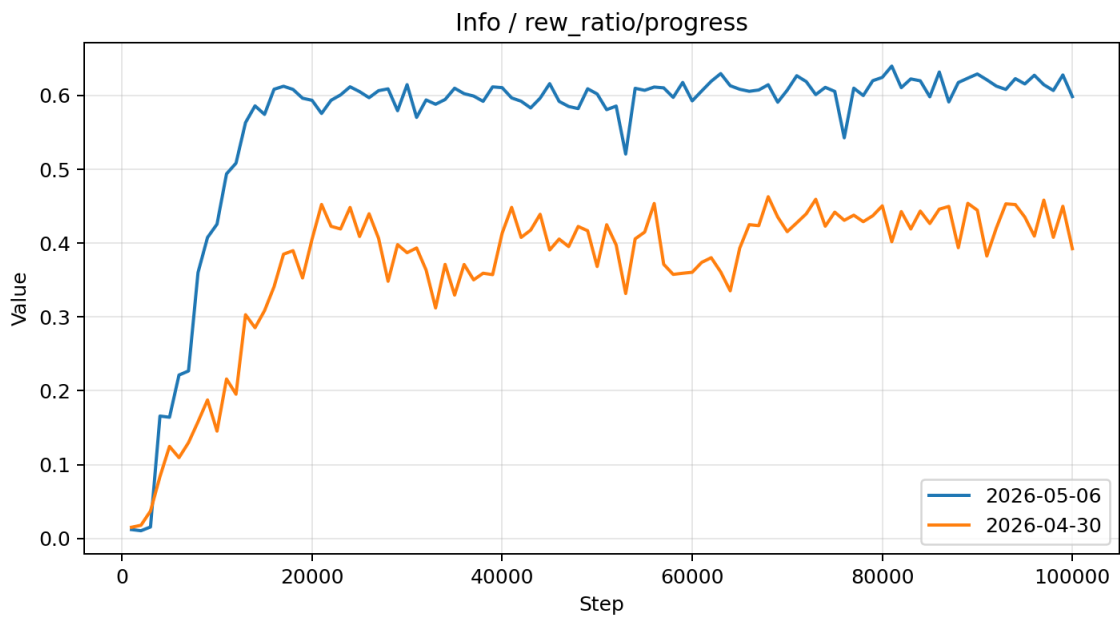


Figure 5-2: Progress ratio in total reward (Info/rew_ratio/progress). Run B approaches ~ 0.6 ; Run A remains around ~ 0.4 .

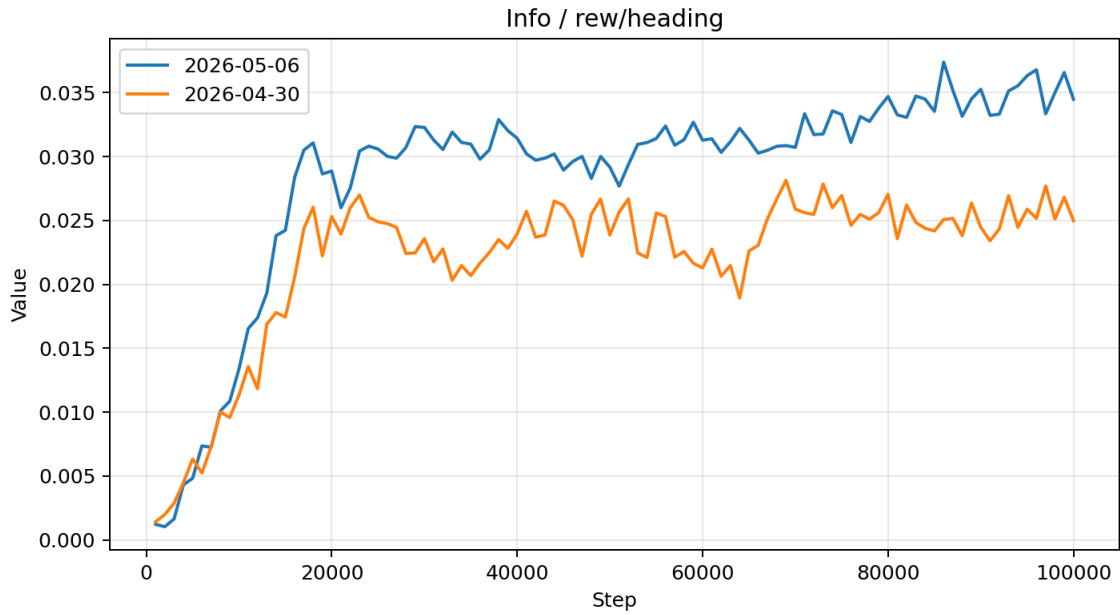


Figure 5-3: Heading reward (Info/rew/heading). Run B reaches ~ 0.035 versus ~ 0.025 for Run A.

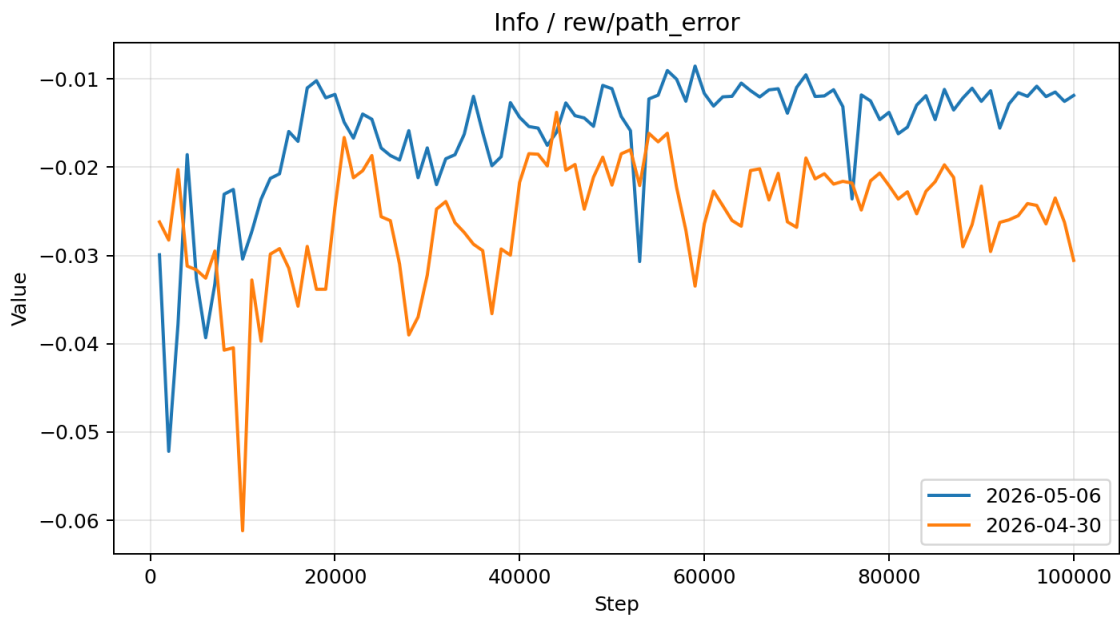


Figure 5-4: Path-error reward (Info/rew/path_error). Less negative values in Run B indicate lower cross-track penalty.



Figure 5-5: Per-step total reward (Info/rew/total). Run B remains positive (~ 0.05 – 0.075); Run A is near -0.025 to 0 .

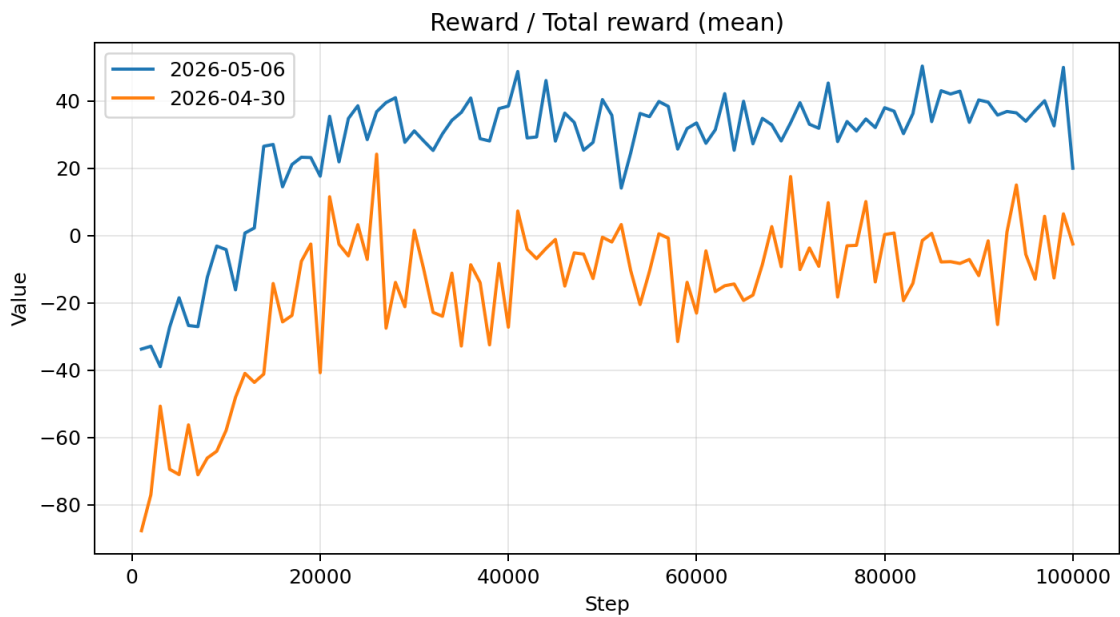


Figure 5-6: Mean episodic return (Reward/Total reward mean). Run B: ~ 30 – 40 ; Run A: ~ -20 to 0 .

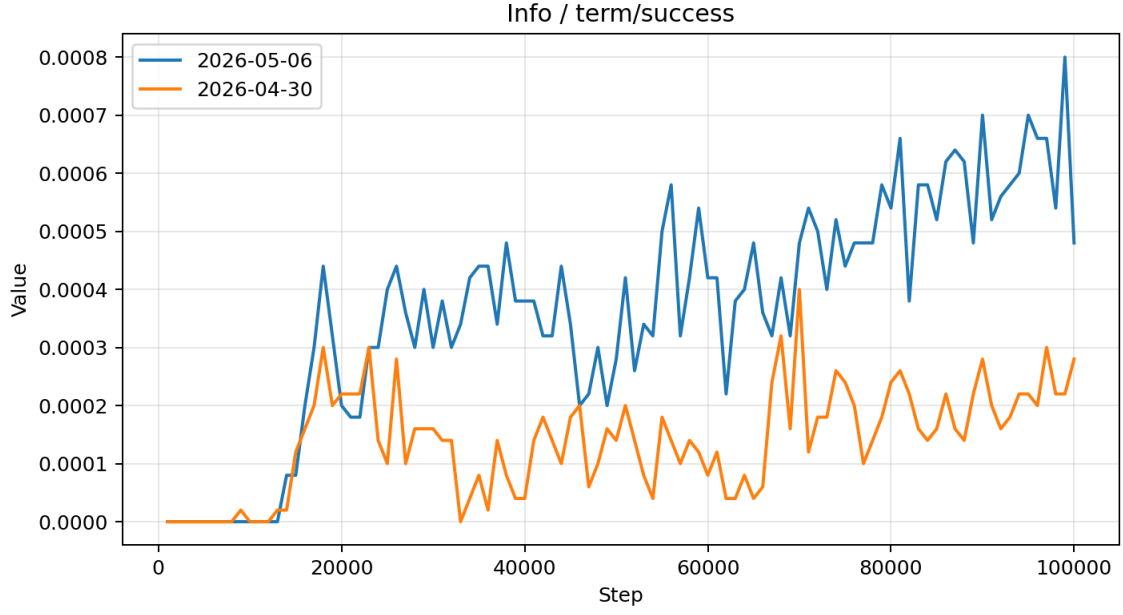


Figure 5-7: Per-step success termination rate (`Info/term/success`). Run B reaches ~ 0.0008 , Run A ~ 0.0003 .

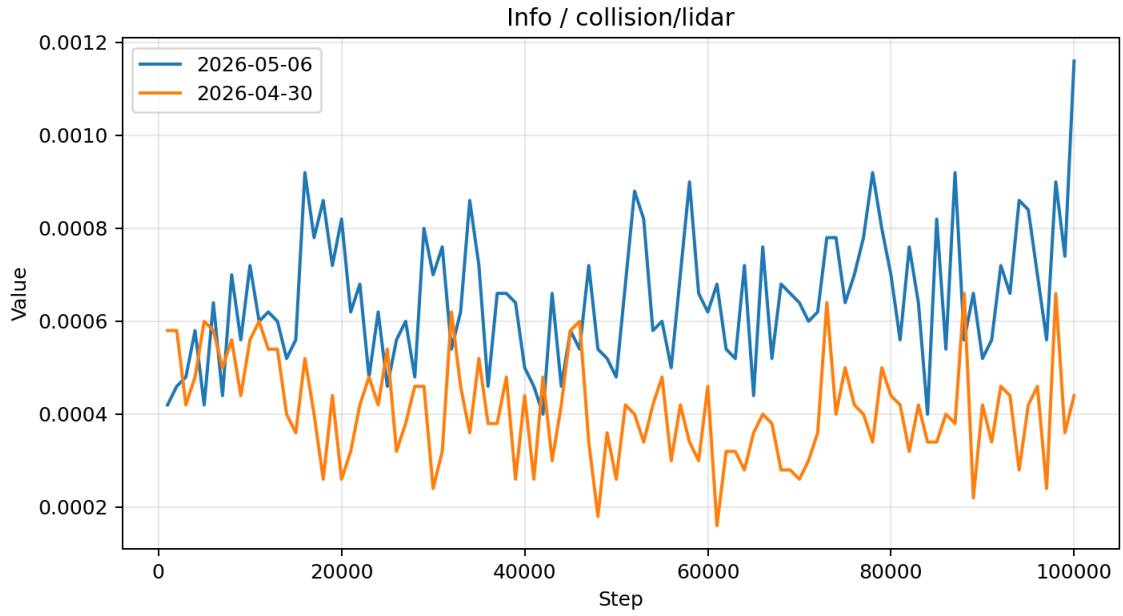


Figure 5-8: Per-step LiDAR collision-event rate (`Info/collision/lidar`). Run B shows moderate increase, consistent with more assertive obstacle negotiation.

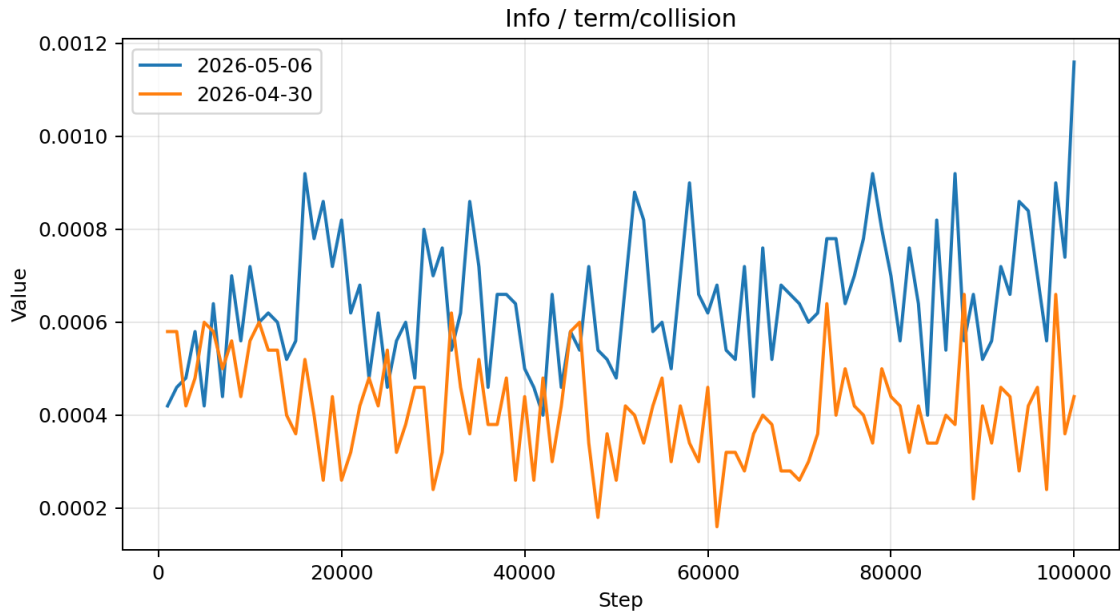


Figure 5-9: Per-step collision termination rate ($\text{Info}/\text{term}/\text{collision}$). Collision frequency is somewhat higher in Run B, indicating a progress–safety trade-off.



Figure 5-10: Value loss ($\text{Loss}/\text{Value loss}$). Both runs converge near 0.2, indicating comparable critic optimization stability.

Chapter 6

Conclusion

6.1 Conclusion

In this thesis I created and tested a local planner based on a recurrent reinforcement learning model for a differentially driven wheeled robot using a simulation-based learning approach (Isaac) that replaces traditional methods to generate local control by learning a policy from experience. The resulting system is composed of a PPO-RNN actor-critic pipeline (where both the actor and critic have separate parameters), a recurrent Gaussian actor for partially observable navigation, a feedforward deterministic critic, and a multi-branch observation method that combines vectorized geometric states with CNNs of the local cost map. An overarching goal for all aspects of this project has been to maintain a deployment-focused interface for the policy. The observation interface is structured so that it represents realistic data that would be available in ROS2/Nav2 systems, and the output pipeline was modified to produce ONNX artifacts suitable for use in real-time C++ applications.

In addition to the development of the overall architecture, another aspect of the research involved the use of physics-informed critic shaping through the incorporation of an HJB residual regularization term into the loss function.

When trained under identical conditions, the run that used the HJB residual demonstrated a higher average progress reward, better alignment of headings, less penalties due to errors along paths, and a significantly larger mean episodic returns than did the runs without the HJB residual.

However, the results of these tests also showed a trade off in terms of behavior: while the

HJB enabled policies were more decisive and proactive in their actions; they experienced an increased frequency of collisions during each episode. Thus, it is evident that in addition to developing appropriate architectures there will still need to be sufficient attention given to rewarding behaviors appropriately.

These tests demonstrate that the proposed framework effectively learns to perform tasks in stochastic, continuous environments and that when implemented as a soft prior critic the HJB residual provides additional improvements in task oriented behaviors beyond what would be achieved if it were implemented as a hard constraint.

As an engineer’s contribution to robotics, the work presented here provides not just a policy configuration but also a full working pipeline: environment definition, observation interface, recurrent training, diagnostic tools and export pathway.

It is this end-to-end view of development that makes possible successful simulations to realities; since it is not only necessary to reproduce pre-processing steps (i.e. how observations are ordered), but also at least equally as important is to ensure consistent access to those observations at runtime.

6.2 Future directions

Future research will occur in three interconnected areas of study.

Algorithmically, an evaluation of off-policy algorithms like SAC (Soft Actor-Critic) will be compared to PPO (Proximal Policy Optimization) on four key dimensions including; convergence speed, collision efficiency, stability and overall quality of final closed loop tracking.

An analysis of the physical performance of the robot deployed by using metrics that are meaningful in terms of physical deployments, which would include, but are limited to; success rates, time-to-goal, path deviations from goal location, dwell time at or near obstacles, number of collisions and smoothness of control inputs.

Simulation-based development will be enhanced through the use of a proxy-dynamics-layer, which will be trained based upon data logged directly from real-world robots. For robots utilizing tracked differentially driven platforms, there are a few issues specific to these types of systems that have been identified as being critical to improve, including; slip/friction asymmetry, acceleration responses and contact behaviors during turning and when close to boundary regions of obstacles. Domain randomization and system identification can also

be used together to improve robustness against over-fitting to the specifics of a simulated environment while simultaneously allowing for stable optimizations of policies.

Finally, the robotics testing track will include integrating the entire system using ONNX inference within a ROS2 C++ controller. The other components of this integrated system include; costmap/path adapters, managing the lifecycle of the hidden state associated with recurrent neural networks and providing safe supervisory functions related to the outputs of the inference module. A multi-step validation plan has been developed to measure the sim-to-real gap (i.e., log replay of offline data → controlled indoor experiments → long duration experiment of actual robotic missions). This process will allow for iterative refinement of both the design of the rewards and processing of observations.

Appendix A

A.1 Pre-Implementation Phase for Autonomous Navigation System – Establishing Stable Networking Communication Infrastructure

A substantial amount of effort went into establishing a reliable real-time communication system during the pre-implementation phase for the development of an autonomous vehicle navigation system. The focus of this pre-implementation phase was to establish a stable network infrastructure, a reliable middleware platform, and a secure configuration of network firewalls. Many issues related to latency, packet fragmentation, Virtual Private Network (VPN), and Robot Operating System (ROS) 2 discovery were established and addressed.

A.1.1 Migration from Multicast-Based Discovery

Mechanism for Fast Data Distribution Service (DDS) Server Architecture The default multicast-based discovery mechanism utilized by ROS 2 was migrated to utilize the Fast DDS Discovery Server Architecture. This migration was necessary due to the fact that most real-world networks (such as VPN's or Carrier Grade Network Address Translation (NAT)) do not reliably support multicast traffic. After configuring the ROS 2 daemon as a super client, the functionality of common introspection commands, such as ROS2 node list and ROS 2 topic list, functioned as intended. Nodes communicating with each other within the same host communicate utilizing Shared Memory (SHM) for high performance, while remote clients send and receive data traffic utilizing User Datagram Protocol (UDP) through the

Discovery Server. By eliminating the reliance on multicast traffic and improving the overall robustness of the system in constrained networks.

After configuring the ROS 2 daemon as a super client, standard introspection commands such as `ros2 node list` and `ros2 topic list` functioned correctly. Within the same host, ROS 2 nodes communicate using shared memory (SHM) for high performance, while remote clients exchange discovery and data traffic over UDP via the Discovery Server. This setup eliminated dependence on multicast and significantly improved robustness in constrained network environments.

A.1.2 NAT traversal Latency optimization within VPN environments

Tailscale showed us initial results where the connection fell back into DERP relays (Poland) with an associated 8-10 second delay. This greatly disrupted our ability to use ROS2 for communication; ROS Visualization Tool (RViz) visualizations would only update every one minute, and topic subscriptions would frequently drop. The problem stemmed from the configuration of NAT on the cellular modem along with Carrier Grade NAT (CGNAT). CGNAT blocked all incoming UDP traffic, so we could not establish a direct peer-to-peer connection. We changed the configuration of the Huawei modem to enable DMZ for the NVIDIA Jetson Orin NX. By enabling a Demilitarized Zone (DMZ) on the modem, we were able to successfully punch through UDP firewalls and establish a direct p2p connection using Tailscale. Network latency plummeted by approximately 80 percent to 20-40 ms, and now we have stable and continuous high-bandwidth ROS 2 data stream transfers, such as maps and point cloud imagery.

A.1.3 Network Security and Firewall Configuration

The use of a DMZ is risky when it is used as an unsecured port. The Uncomplicated Firewall (UFW) was set up to limit incoming traffic. Access to all ports on the Tailscale interface was allowed. Only the local network had Secure Shell (SSH) access. The firewall configuration provided the advantages of allowing the direct connection needed for real-time communication between robots and maintained a secure level of exposure of the Orin NX system.

A.1.4 Maximum Transmission Unit (MTU) Alignment and UDP Packet Fragmentation

During stress testing it was observed that the transmission of fast DDS used to send out UDP packets in the range of 8-9 kb; Tailscale has an MTU of 1280 bytes, and this caused double fragmentation because the Linux kernel is also a fragmenter; if any one part of the message was lost, the entire DDS message would be lost, and communication became unstable. This problem was solved by explicitly setting the maximum size of UDP messages (`maxmessagesize`) within fast DDS configuration. As a result, fast DDS stopped sending excessive amounts of fragments through the vpn and significantly improved reliability of packet delivery

The issue was resolved by explicitly limiting the maximum UDP message size (`maxMessageSize`) in the Fast DDS configuration. This adjustment eliminated excessive fragmentation and significantly improved packet delivery reliability over the VPN.

A.1.5 Summary of Infrastructure Readiness

The preparatory work for this experiment had to be done to make sure that deterministic (predictable), low latency (fast), and reliable (no lost packets, etc.) ROS 2 communication would occur. In addition to resolving limitations on discovery, VPN routing problems, NAT traversal, Firewall Security, and MTU mismatch, we were able to establish a stable platform to run the experiments from. A solid base is needed for the implementation and testing of the proposed Autonomous Navigation Framework in environments that are changing and subject to communication delay or packet loss as it could render the results of the learning and control invalid.

Bibliography

- [1] Javed Ahmad, Mohd Wahab, Ahmad Ramli, Md Yushalify Misro, Wan Ezza, and Wan Hasan. Enhancing performance of global path planning for mobile robot through alpha–beta guided particle swarm optimization (abgpso) algorithm. *Measurement*, 257:118633, 01 2026.
- [2] Fatin Hassan Ajeil, Ibraheem Ibraheem, Mouayad Sahib, and Amjad Humaidi. Multi-objective path planning of an autonomous mobile robot using hybrid pso-mfb optimization algorithm. *Applied Soft Computing*, 89:106076, 04 2020.
- [3] Muhammad Aatif, Muhammad Baig, Umar Adeel, and Ammar Rashid. Path planning with adaptive autonomy based on an improved a algorithm and dynamic programming for mobile robots. *Information*, 16:700, 08 2025.
- [4] Sangeeth Venu and Muralimohan Gurusamy. A comprehensive review of path planning algorithms for autonomous navigation. *Results in Engineering*, 28:107750, 2025.
- [5] Jie Yu, Toan Tan Nguyen, Thi-Kien Dao, and Trong-The Nguyen. Global path planning algorithm for mobile robots: A review. *Journal of Network Intelligence*, 9(2):643–657, May 2023. Received July 11, 2023; revised September 19, 2023; accepted November 24, 2023.
- [6] Aggrey Shitsukane, Eng Mathews, Calvins Otieno, and James Obuhuma. A systematic literature review of path-planning strategies for robot navigation in unknown environment. *SSRN Electronic Journal*, 01 2025.
- [7] Qiang Yu and Jun Zhou. A review of global and local path planning algorithms for mobile robots. In *2024 8th International Conference on Robotics, Control and Automation (ICRCA)*, pages 84–90, 2024.
- [8] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- [9] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. In *Technical Report*, number TR 98-11, 1998.
- [10] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [11] Kornél Katona, Husam A. Neamah, and Péter Korondi. Obstacle avoidance and path planning methods for autonomous navigation of mobile robot. *Sensors*, 24(11), 2024.

- [12] Federico Peralta, Mario Arzamendia Lopez, Derlis Gregor, Daniel Gutiérrez, and S.L. Toral. A comparison of local path planning techniques of autonomous surface vehicles for monitoring applications: The ypacarai lake case-study. *Sensors*, 20, 03 2020.
- [13] Tagor Hossain, H. Habibullah, Rafiqul Islam, and Ricardo Padilla. Local path planning for autonomous mobile robots by integrating modified dynamic-window approach and improved follow the gap method. 06 2024.
- [14] Yang Zhou, Winnie Daamen, Tiedo Vellinga, and Serge Hoogendoorn. Review of maritime traffic models from vessel behavior modeling perspective. *Transportation Research Part C: Emerging Technologies*, 105:323–345, 2019.
- [15] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, 1985.
- [16] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.
- [17] Yonghao Shu, Dehan Shen, and Mohan Chen. Apf method of dynamic obstacle avoidance with relative velocity. *Applied and Computational Engineering*, 199:167–172, 12 2025.
- [18] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.
- [19] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control. In *European Control Conference (ECC)*, 2015.
- [20] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Integrated online trajectory planning and optimization in distinctive topologies. *Robotics and Autonomous Systems*, 88:142–153, 2017.
- [21] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.
- [22] Arthur L. Samuel. Some studies in machine learning using the game of checkers. ii—recent progress. *IBM Journal of Research and Development*, 11(6):601–617, 1967.
- [23] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.
- [24] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.
- [25] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 1998.
- [26] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.
- [27] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

- [28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharmashan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [29] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992.
- [30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [31] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey 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)*, pages 1861–1870, 2018.
- [32] Jens Kober, J. Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32:1238–1274, 09 2013.
- [33] Angelo Moroncelli, Vishal Soni, Marco Forgione, Dario Piga, Blerina Spahiu, and Loris Roveda. The duality of generative ai and reinforcement learning in robotics: A review. *Information Fusion*, 129:104003, 2026.
- [34] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes. *arXiv preprint arXiv:2408.03539*, 2024.
- [35] Kumater Ter, Ore-Ofé Ajayi, and Daniel Udekwe. Taxonomy and trends in reinforcement learning for robotics and control systems: A structured review. *arXiv preprint arXiv:2510.21758*, 2025.
- [36] Dantong Xiang. Reinforcement learning in autonomous driving. *Applied and Computational Engineering*, 48:17–23, 03 2024.
- [37] Ruiqi Zhang, Jing Hou, Florian Walter, Shangding Gu, Jiayi Guan, Florian Röhrbein, Yali Du, Panpan Cai, Guang Chen, and Alois Knoll. Multi-agent reinforcement learning for autonomous driving: A survey. *arXiv preprint arXiv:2408.09675*, 2024.
- [38] Elahe Delavari, Feeza Khan Khanzada, and Jaerock Kwon. A comprehensive review of reinforcement learning for autonomous driving in the carla simulator. *arXiv preprint arXiv:2509.08221*, 2025.
- [39] István Szita. *Reinforcement Learning in Games*, volume 12, pages 539–577. 01 2012.
- [40] Gilzamir Gomes, Creto A. Vidal, Joaquim B. Cavalcante-Neto, and Yuri L.B. Nogueira. A modeling environment for reinforcement learning in games. *Entertainment Computing*, 43:100516, 2022.
- [41] Yi Liao, Yu Gu, Yuan Sui, Zining Zhu, Yifan Lu, Guohua Tang, Zhongqian Sun, and Wei Yang. Think in games: Learning to reason in games via reinforcement learning with large language models. *arXiv preprint arXiv:2508.21365*, 2025.

- [42] M. Waltz and K. Fu. A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10(4):390–398, 1965.
- [43] Armando Alves Neto. Reinforcement learning for control systems with time delays: A comprehensive survey. *arXiv preprint arXiv:2602.00399*, 2026.
- [44] Yingjie Zhu, W. Wan Hasan, Hafiz Ramli, Nor Mohd Haziq Norsahperi, Muhamad Kassim, and Yiduo Yao. Deep reinforcement learning of mobile robot navigation in dynamic environment: A review. *Sensors*, 25:3394, 05 2025.
- [45] Honghu Xue, Benedikt Hein, Mohamed Bakr, Georg Schildbach, Bengt Abel, and Elmar Rueckert. Using deep reinforcement learning with automatic curriculum learning for mapless navigation in intralogistics. *Applied Sciences*, 12:3153, 03 2022.
- [46] Sahar Salimpour, Jorge Peña Queralta, Diego Paez-Granados, Jukka Heikkonen, and Tomi Westerlund. Sim-to-real transfer for mobile robots with reinforcement learning: from nvidia isaac sim to gazebo and real ros 2 robots, 01 2025.
- [47] Jaewan Choi, Geonhee Lee, and Chibum Lee. Reinforcement learning-based dynamic obstacle avoidance and integration of path planning. *Intelligent Service Robotics*, 14:663–677, 10 2021.
- [48] Ge Lun, Zhou Xiaoguang, and Wang Yongcong. Learning to drive as humans do: Reinforcement learning for autonomous navigation. *International Journal of Advanced Robotic Systems*, 21, 10 2024.
- [49] Minako Oriyama, Pitoyo Hartono, and Hideyuki Sawada. Human-in-the-loop transfer learning in collision avoidance of autonomous robots. *Biomimetic Intelligence and Robotics*, 5(1):100215, 2025.
- [50] Zifan Xu, Bo Liu, Xuesu Xiao, Anirudh Nair, and Peter Stone. Benchmarking reinforcement learning techniques for autonomous navigation. pages 9224–9230, 05 2023.
- [51] Bernhard Jaeger, Daniel Dauner, Jens Beißwenger, Simon Gerstenecker, Kashyap Chitta, and Andreas Geiger. Carl: Learning scalable planning policies with simple rewards. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2025. arXiv:2504.17838.
- [52] Zhanteng Xie and Philip Dames. Drl-vo: Learning to navigate through crowded dynamic scenes using velocity obstacles. *arXiv preprint arXiv:2301.06512*, 2023.
- [53] Bhargava Uppuluri, Anjel Patel, Neil Mehta, Sridhar Kamath, and Pratyush Chakraborty. Curla: Curriculum learning based deep reinforcement learning for autonomous driving. *arXiv preprint arXiv:2501.04982*, 2025.
- [54] Guillermo Cid Ampuero, Gabriel Hermosilla, Germán Varas, and Matías Toribio Clark. Deep reinforcement learning for sim-to-real robot navigation with a minimal sensor suite for beach-cleaning applications. *Applied Sciences*, 15(19):10719, 2025.
- [55] Stable-Baselines3 Contributors. Ppo — stable-baselines3 documentation. <https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html>, 2026. Accessed: 2026-03-12.

- [56] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep reinforcement learning: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*, 2020.
- [57] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.
- [58] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning*, 2018.
- [59] Ray Team. Rllib algorithms. <https://docs.ray.io/en/latest/rllib/rllib-algorithms.html>, 2026. Ray Documentation. Accessed: 2026-03-12.
- [60] Jiwon Kim, Moon-Ju Kang, KangHun Lee, HyungJun Moon, and Bo-Kwan Jeon. Deep reinforcement learning for asset allocation: Reward clipping. 2023.
- [61] Longchao Da, Justin Turnau, Thirulogasankar Kutralingam, Alvaro Velasquez, Paulo Shakarian, and Hua Wei. A survey of sim-to-real methods in rl: Progress, prospects and challenges with foundation models, 02 2025.
- [62] Wei Zhu, Xian Guo, Dai Owaki, Kyo Kutsuzawa, and Mitsuhiro Hayashibe. A survey of sim-to-real transfer techniques applied to reinforcement learning for bioinspired robots. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–16, 09 2021.
- [63] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. pages 737–744, 12 2020.
- [64] Xiaoyu Chen, Jiachen Hu, Chi Jin, Lihong Li, and Liwei Wang. Understanding domain randomization for sim-to-real transfer. In *International Conference on Learning Representations (ICLR)*, 2022.
- [65] Yuki Kadokawa, Lingwei Zhu, Yoshihisa Tsurumine, and Takamitsu Matsubara. Cyclic policy distillation: Sample-efficient sim-to-real reinforcement learning with domain randomization. *arXiv preprint arXiv:2207.14561*, 2022.
- [66] Rudrashis Majumder, Sri Siddarth Chakaravarthy, Samahith S A, Suresh Sundaram, and Hemanth Patel. Oa-pinn: Efficient obstacle avoidance for autonomous vehicle safety with physics-informed neural networks. In *2024 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages 1–6, 2024.
- [67] Manan Tayal, Aditya Singh, Shishir Kolathaya, and Somil Bansal. A physics-informed machine learning framework for safe and optimal control of autonomous systems, 02 2025.
- [68] Eduardo Sebastian, Thai Duong, Nikolay Atanasov, Eduardo Montijano, and Carlos Sagüés. Physics-informed multi-agent reinforcement learning for distributed multi-robot problems. *arXiv preprint arXiv:2401.00212*, 2024.
- [69] Hang Zhou, Yihao Qin, Dan Xu, and Yiding Ji. Physics-informed imitative reinforcement learning for real-world driving. *arXiv preprint arXiv:2407.02508*, 2024.

- [70] Vittorio Giammarino, Ruiqi Ni, and Ahmed H. Qureshi. Physics-informed value learner for offline goal-conditioned reinforcement learning. *arXiv preprint arXiv:2509.06782*, 2025.
- [71] Peng Zhao and Yongming Liu. Physics informed deep reinforcement learning for aircraft conflict resolution. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):8288–8301, 2022.
- [72] Pu Feng, Rongye Shi, Size Wang, Junkang Liang, Xin Yu, Simin Li, and Wenjun Wu. Safe and efficient multi-agent collision avoidance with physics-informed reinforcement learning. *IEEE Robotics and Automation Letters*, 9(12):11138–11145, 2024.
- [73] Amartya Mukherjee and Jun Liu. Bridging physics-informed neural networks with reinforcement learning: Hamilton-jacobi-bellman proximal policy optimization (hjbppo), 2023.
- [74] Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 16691–16700, Baltimore, Maryland, USA, 2022. PMLR.