The City College of New York
Grove School of Engineering

Reinforcement Learning Control for Mobile Robot Parking with Safety Constraints

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering

in

Mechanical Engineering

by

Junquan Wu

Department of Mechanical Engineering

The City College of New York

Advisor: Prof. Bo Wang

December 2025

Abstract

This thesis studies a hybrid framework that combines reinforcement learning (RL) with control barrier function (CBF)-based methods to achieve safe autonomous vehicle control, focusing on parking with obstacle avoidance. We apply Deep Deterministic Policy Gradient (DDPG) methods for continuous control and evaluate policies across three Simulink environments of increasing fidelity: a kinematic model, a dynamic model, and a dynamic model with actuator disturbance. In parking tasks, DDPG learns smooth, stable trajectories and maintains performance under modeling uncertainty and input noise.

To address hard safety requirements in obstacle-rich settings, we augment the RL policy with a CBF safety filter that enforces forward invariance of a state-based safe set in real time. Experiments show that (i) reward shaping alone yields "soft safety" (avoidance behavior without guarantees), (ii) post-hoc CBF filtering prevents collisions but can cause abrupt corrections if the policy was not trained with the filter in the loop, and (iii) retraining the agent with the CBF filter active achieves smooth, collision-free navigation with formal constraint satisfaction.

In general, the RL-CBF approach preserves the adaptability of learning while providing control-theoretic safety guarantees, pointing to a practical path for reliable autonomous control in uncertain and nonlinear environments.

Contents

Chapter 1

Introduction

1.1 Reinforcement Learning & Control: Background and Motivation

1.1.1 Limitations of Classical Control Approaches

Classical control theory has long served as the foundation of modern automation and engineering systems. Techniques such as PID control, sliding mode control, and optimal control have provided robust and interpretable solutions for decades, especially when accurate mathematical models of the system dynamics are available. However, as control applications expand toward increasingly complex, nonlinear, and uncertain environments, such as autonomous driving, robotic manipulation, and energy systems, the assumptions underpinning classical control become less tenable. [1]

Real-world control problems routinely violate the assumptions under which many classical designs are derived [2]. Practical systems exhibit uncertainties, strong nonlinearities, unmodeled dynamics, time variation, and hard constraints on states and inputs [3]. Constructing a precise model for high-dimensional plants with coupled dynamics or unknown parameters is difficult; even small modeling errors can degrade performance or compromise stability [4]. In safety-critical settings, these gaps can lead to constraint violations or outright failure.

Common real-world violations of design assumptions include:

- *Parametric uncertainty and drift:* Physical properties of systems may change over time due to payload variation, temperature, or other environmental conditions, leading to model inaccuracies and degraded performance. In [5] analyzed how parametric uncertainty and unmodeled dynamics affect robust stability, motivating the need for adaptive and uncertainty-aware control strategies.

- *Unmodeled nonlinearities:* Effects such as friction, backlash, dead zones, saturation, hysteresis, and rate limits are difficult to model precisely, and they can significantly affect closed-loop stability. In [6] emphasized that nonlinearities and unmodeled input dynamics often lead to unpredictable transient responses and degraded tracking performance, posing major challenges for accurate control system design.

- *Time variation and delays:* Sensor and actuator delays, sampling jitter, and hybrid switching behaviors introduce time-dependent uncertainty and can destabilize otherwise well-tuned controllers. In [7] provided a comprehensive overview of delay-system theory, showing that time delays not only degrade phase margins but can also lead to oscillation and instability.

- *High dimensionality and coupling:* Multi-input–multi-output (MIMO) systems exhibit strong coupling and high dimensionality, making low-order linear approximations inadequate for accurate modeling. In [8] reviewed decoupling strategies for industrial MIMO processes and highlighted how system interactions complicate independent control loop design.

- *Hard constraints:* State and input limits can cause actuator saturation and integrator windup, resulting in nonlinear behavior or instability without proper anti-windup design. In [9] were among the first to formalize anti-windup techniques, demonstrating how saturation-induced nonlinearities can be mitigated through feedback correction.

Classical controllers such as PID, Linear–Quadratic Regulator(LQR), and H_∞ can be highly effective when accurate models are available and operating conditions remain near the

design point [2]. Robust and adaptive methods expand the envelope but still require structured uncertainty descriptions, sufficient excitation for parameter convergence, and meticulous tuning; performance may deteriorate if these conditions are not met. Model Predictive Control (MPC) directly handles constraints, yet hinges on model fidelity and can become computationally burdensome for fast, high-dimensional systems unless substantial model reduction and tailored solvers are employed [3]. These limitations motivate data-driven and learning-based approaches that can complement or enhance traditional methods by adapting to uncertainties and capturing nonlinear behaviors from interaction data [10].

1.1.2 Motivation of Reinforcement Learning Control

There are many model-based approaches, like in [11] authors present a cascaded control framework for underactuated surface vessels, where kinematic and dynamic subsystems are analytically modeled to achieve stable leader–follower formation and trajectory tracking under nonlinear dynamics. In contrast, RL offers a model-free, data-driven paradigm in which a controller is learned directly from system interaction, rather than derived from a fully specified plant model [12]. In RL, an agent observes the system state, selects actions, receives feedback in the form of rewards, and iteratively improves its policy to maximize long-term task performance [4]. With deep neural networks serving as function approximators, deep reinforcement learning (DRL) extends these principles to nonlinear, high-dimensional, and tightly coupled systems [3].

Why RL helps in complex control:

- *Reduced dependence on exact models:* model-free methods learn feedback directly from the data and maintain performance under unmodeled dynamics and parameter drift [10].
- *Expressiveness for nonlinear, high-DOF plants:* neural policies and critics capture strong nonlinearities and cross-couplings that challenge linear or low-order designs [2].
- *Direct task optimization:* RL targets end objectives—tracking accuracy, energy use, safety margins—without relying on analytic proxies; multi-objective trade-offs can be encoded in rewards or constraints [12].

- *Robustness and adaptation:* domain randomization, disturbance injection, and adversarial training improve generalization in payload changes, delays, and environmental variability; online fine-tuning can adapt policies with new data [3].

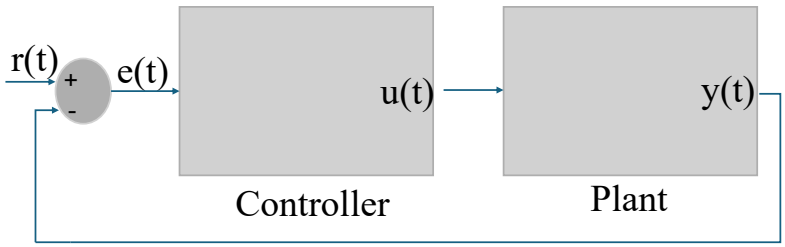
- *Hybridization with models and constraints:* model-based RL improves sample efficiency using approximate dynamics; MPC+RL schemes use learned policies as warm starts or terminal controllers; safety filters and backup policies provide constraint handling [12].

- *Sim-to-real transfer:* training in high-fidelity simulators with calibration, randomization, and residual learning enables deployment to hardware despite the reality gap [10].

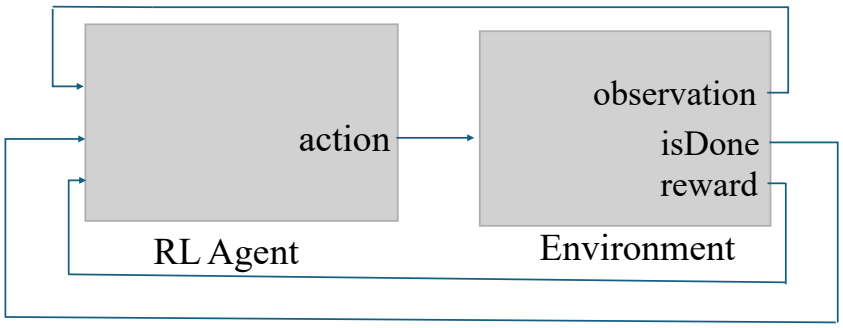
In summary, RL complements classical, robust, and predictive control by learning feedback laws that accommodate nonlinearities, uncertainty, and high-dimensional couplings without requiring exact models [2]. Evidence from continuous control benchmarks and emerging industrial applications indicates that DRL can deliver scalable and performance controllers with real-world variability [3].

Control Loop Comparison: Classical vs RL

In order to better understand the role of RL as a control paradigm, it is helpful to compare it directly with the classical feedback control loop. Both frameworks share the same high-level goal: to generate control actions that guide the system output toward a desired objective. However, the way they are structured and how information flows through them are fundamentally different, as illustrated in Figure 1.1. In the traditional loop, $r(t)$ the reference input, $y(t)$ the system output, $e(t) = r(t) - y(t)$ the error signal, and $u(t)$ the control input are applied to the plant. In the RL loop, the agent receives an *observation* (system state), selects an *action* to influence the environment, obtains a scalar *reward* measuring task performance, and checks *isDone* to determine whether the episode has ended.



(a) Traditional control loop.



(b) Reinforcement learning control loop.

Figure 1.1: Comparison between a traditional control system and a reinforcement learning agent.

The correspondence between the two paradigms can be summarized as follows:

- In classical control, $r(t)$ encodes the desired behavior. In RL, this goal is encoded within the reward function design.

- The output $y(t)$ of the plant corresponds to the *observation* that the agent receives.
- The error $e(t)$ is used explicitly in traditional control, but in RL the same concept is embedded implicitly in the reward, where smaller errors yield higher rewards.
- The control input $u(t)$ in classical control is analogous to the *action* chosen by the RL agent.
- The controller in classical systems is a designed feedback law (e.g., PID, LQR, MPC), whereas in RL the agent learns a *policy* mapping observations to actions through repeated interaction.
- Long-term performance in control is typically measured via steady-state error, overshoot, or energy usage; in RL, this corresponds to the *return*, the accumulated sum of rewards over an episode.

In summary, traditional control relies on precise models and explicit error correction, while reinforcement learning provides a data-driven framework that learns policies from experience. This makes RL particularly attractive for systems with nonlinearities, uncertainties, or hard-to-model dynamics, where classical methods struggle.

1.2 Literature review

RL has evolved from theoretical development to practical deployment across multiple engineering domains, including aerial robotics, safety-critical systems, autonomous ground vehicles, and energy systems. The reviewed literature highlights both the diversity of applications and the growing maturity of RL-based control strategies.

1.2.1 Aerial Robotics

In aerial robotics, RL has been employed to address highly nonlinear and fault-prone dynamics. Studies have shown that RL outperforms model predictive control in quadrotor morpho-transition maneuvers, where adaptability to actuator faults and contact uncertainties is crucial [13]. Other works improved quadrotor robustness by integrating disturbance

observers into RL controllers, significantly reducing hover error in real-world deployments [14]. Transformer-assisted RL frameworks have also been proposed for fault-tolerant quadrotor control, enabling policies to adapt online to actuator failures without retraining [15]. In addition, real-time UAV operation has been validated through deployment of simulated annealing–optimized TD3 controllers on hardware drones, outperforming backstepping and sliding mode control baselines [16].

1.2.2 Autonomous Ground Vehicles and Mobile Robots

In autonomous ground vehicles and mobile robots, RL has demonstrated strong adaptability and learning efficiency. Applications of the DDPG algorithm to vehicle path following have achieved smoother and more accurate steering control compared to classical methods [17]. RL frameworks based on end-to-end vision have also been introduced to avoid obstacles, mapping the input of the monocular camera directly to actions, and depth prediction improves robustness in navigation tasks [18].

Beyond steering control, deep RL has been used to improve trajectory tracking, lane keeping, and parking maneuvers under complex and uncertain dynamics.In [19] a model-free DDPG controller for autonomous driving was proposed that outperformed conventional PID and MPC methods in dynamic lane change tasks. In [20] designed a hierarchical RL framework that combines high-level decision-making with low-level continuous control to allow safe overtaking and intersection navigation.

1.2.3 Safety-Critical Control Systems

For safety-critical and constrained dynamic systems, RL has been extended to include formal guarantees on both stability and constraint satisfaction. Model-free frameworks such as fitted Q-iteration have been developed to manage state and control constraints without explicit models, while reachability-based methods like the Reachability-based Trajectory Safeguard (RTS) integrate offline reachability analysis with online RL to ensure safe action selection in continuous control tasks [21]. More recent studies have embedded control-theoretic safety mechanisms directly into the RL framework: for example, a framework that uses Gaus-

sian process-learned dynamics and a safety index to provide probabilistic forward-invariance guarantees has been proposed [22]. Other work employs disturbance-observer-based CBF filters as safety layers above RL to reduce violations under uncertainty [23]. Additional methods combine CLF and CBF within actor-critic RL structures to jointly guarantee stability and safety [24]. Taken together, these advances illustrate how modern RL methods are moving beyond purely performance-driven optimization toward frameworks that embed formal safety certificates, constraint satisfaction, and safe exploration, thus paving the way for deployment in real-world safety-critical control applications.

1.2.4 Industrial Robotics and Power Systems

In industrial robotics and power systems, RL has shown promise for real-time adaptability under uncertainties. Online RL controllers have been developed for robotic manipulators subject to large friction variations, outperforming computed torque, PD, iterative learning, and adaptive neural controllers [25]. Comprehensive reviews of RL in power system optimization highlight how model-free methods enable data-driven energy management, while model-based methods improve interpretability and sample efficiency [26].

1.2.5 Summary

In general, these applications demonstrate that RL is capable of handling nonlinearities, uncertainties, and safety constraints in diverse engineering domains. From UAVs and ground vehicles to industrial manipulators and smart grids, RL is increasingly transitioning from simulation to real-world deployment with measurable improvements over classical control methods.

1.3 Thesis Organization

This thesis is organized into five chapters that progressively develop the integration of reinforcement learning with safety-critical control principles.

- **Chapter 1** introduces the background and motivation for applying reinforcement

learning in control systems. It reviews the limitations of classical control approaches in complex and uncertain environments and highlights the growing need for adaptive and data-driven methods. The research gap and main contributions of this thesis are also presented.

- **Chapter 2** provides a theoretical foundation for reinforcement learning in the context of control. It reviews the Bellman equation, model-based and model-free methods, and key algorithms such as Monte Carlo (MC) methods, Temporal Difference (TD) methods, and Actor–Critic methods.

- **Chapter 3** presents the design and implementation of a reinforcement learning based controller for an autonomous parking robot. The chapter formulates the environment dynamics, defines the reward function, and details the training process of the DDPG agent. Simulation results demonstrate how the agent learns goal-directed behavior through interaction.

- **Chapter 4** extends the RL framework by incorporating CBF to ensure safety during motion. A hybrid RL–CBF scheme is proposed, and its performance is evaluated under obstacle-avoidance scenarios. Comparative results between baseline RL and RL-CBF controllers are discussed to illustrate the improvement in safety and stability.

- **Chapter 5** concludes the thesis with a discussion of key findings, limitations, and potential future directions. Suggestions are made for real-world implementation, multi-agent extensions, and adaptive safety-critical learning frameworks.

Chapter 2

Methodology

2.1 Basic Principles

2.1.1 State, Action, and Reward

In RL, the interaction between an agent and its environment is modeled as a sequence of discrete decisions [10]. At the core of this interaction are three key elements: **state** (s), **action** (a), and **reward** (r) [12].

At each discrete time step t, the agent observes the current state $s_t \in S$, selects an action $a_t \in A$, and receives a reward $r_{t+1} \in \mathbb{R}$. The environment then transitions to a new state s_{t+1}, governed by the transition probability $P(s_{t+1} \mid s_t, a_t)$[4].

The agent's goal is to learn a policy $\pi(a \mid s)$, which maps states to probabilities of selecting each action, to maximize the expected cumulative reward [10, 12]. Return of time step t is defined as

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

Where $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards [3].

2.1.2 Bellman Equation and Bellman Optimality Equation

The **Bellman equation** provides a recursive formulation of the value function, which measures the expected return when starting in a state s and following the policy π [10, 3].

The state-value function $V^\pi(s)$ is defined as

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid s_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right] \tag{2.2}$$

The **Bellman expectation equation** for the value function is:

$$V^\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^\pi(s') \right] \tag{2.3}$$

The corresponding **action-value function** $Q^\pi(s, a)$ is defined as

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid s_t = s, a_t = a \right] \tag{2.4}$$

The **Bellman optimality equation** expresses the value under the optimal policy π^* as:

$$V^*(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^*(s') \right] \tag{2.5}$$

$$Q^*(s, a) = \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right] \tag{2.6}$$

These equations serve as the foundation for most RL algorithms, including value iteration, Q-learning, and actor-critic methods [2, 3, 10].

2.2 Model-Based Method vs. Model-Free Method

RL algorithms can be broadly categorized into two classes: model-based methods and model-free methods [10, 3]. The key difference lies in whether the agent has access to or learns a model of the dynamics of the environment [4].

2.2.1 Model-Based Method

Model-based reinforcement learning assumes knowledge of the environment's transition dynamics and reward structure [4, 12]. The model typically consists of the state transition probability $P(s' \mid s, a)$ and the reward function $R(s, a)$. With this model, the agent can plan by simulating future trajectories, making model-based methods generally more sample-efficient compared to model-free approaches [10, 3].

Two foundational algorithms in this category are **value iteration** and **policy iteration**, both of which rely on dynamic programming principles [4].

Value Iteration

Value iteration updates the value function directly by repeatedly applying the Bellman optimality operator [10, 3]. At each step, the algorithm estimates the expected return of taking an action in a given state and then updates the state value accordingly. This process continues until convergence, after which an optimal policy can be derived by acting greedily with respect to the learned value function. Value iteration is guaranteed to converge to the optimal value function for finite state and action spaces [4].

Policy Iteration

Policy iteration alternates between two key steps: policy evaluation and policy improvement [4]. In the evaluation step, the value of the current policy is estimated, often through iterative updates. In the improvement step, a new policy is derived by acting greedily with respect to the estimated value function. This process continues until the policy stabilizes, resulting in the optimal policy. Compared to value iteration, policy iteration can converge more quickly in some cases because it refines the policy directly [3].

2.2.2 Model-Free Method

Model-free reinforcement learning does not rely on explicit knowledge of the environment's transition probabilities or reward function [10, 12]. Instead, the agent learns directly from experience by interacting with the environment. This makes model-free methods more

flexible but often less sample-efficient. Three widely used approaches in model-free RL are Monte Carlo methods, Temporal Difference methods, and Actor-Critic methods[10].

Monte Carlo Methods

Monte Carlo methods estimate value functions based on complete episodes of experience [27]. By averaging the returns observed from many episodes, the agent approximates the expected return for each state or state-action pair. Monte Carlo methods do not require knowledge of the environment's dynamics, but they rely on episodic tasks and can be slow to converge because updates occur only at the end of an episode.

Temporal Difference Methods

Temporal Difference methods combine ideas from Monte Carlo and dynamic programming [27]. Instead of waiting until the end of an episode, TD methods update value estimates after each time step by bootstrapping the estimated value of the next state. This makes TD methods more efficient and more suitable for online learning compared to Monte Carlo. Popular TD algorithms include State–Action–Reward–State–Action (SARSA) and Q-learning.

Actor-Critic methods

Actor-Critic methods combine the advantages of both value-based and policy-based approaches [28]. The actor is responsible for selecting actions according to a parameterized policy, while the critic evaluates these actions by estimating value functions. The critic provides feedback to the actor by computing gradients that guide policy improvement. This separation allows actor-critic methods to handle large or continuous action spaces more effectively than purely value-based methods. Well-known examples include Advantage Actor-Critic (A2C) and Asynchronous Advantage Actor-Critic (A3C) [29].

2.3 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient is an actor-critic reinforcement learning algorithm designed for environments with continuous action spaces. It was introduced by Lillicrap et al.

[30] and extends the Deterministic Policy Gradient (DPG) theorem [31] by integrating deep neural networks. DDPG has been widely applied to robotic control, autonomous driving, and other continuous control tasks.

2.3.1 Core Idea

DDPG builds upon the Deterministic Policy Gradient theorem, which states that the gradient of the expected return with respect to policy parameters can be written as [30]

$$\nabla_{\theta^\mu} J(\theta^\mu) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_a Q^\mu(s, a \mid \theta^Q)\big|_{a=\mu(s)} \nabla_{\theta^\mu}\mu(s \mid \theta^\mu)\right], \tag{2.7}$$

where $\mu(s \mid \theta^\mu)$ is a deterministic policy parameterized by θ^μ, and $Q^\mu(s, a \mid \theta^Q)$ is the action value function parameterized by θ^Q.

Unlike stochastic policy gradient methods, which require sampling from a distribution over actions, DDPG directly learns a deterministic mapping from states to actions, making it suitable for high-dimensional continuous control.

Architecture

DDPG employs an actor-critic framework with four neural networks [30]:

- **Actor network** $\mu(s \mid \theta^\mu)$: outputs a deterministic action given a state.
- **Critic network** $Q(s, a \mid \theta^Q)$: estimates the action-value function.
- **Target actor network** $\mu'(s \mid \theta^{\mu'})$: a delayed copy of the actor for stable learning.
- **Target critic network** $Q'(s, a \mid \theta^{Q'})$: a delayed copy of the critic.

Training Mechanism

The training of the DDPG agent involves several interconnected components designed to ensure stable learning in continuous control environments.

Experience Replay. All observed transitions, consisting of the current state, action, reward, and next state, are stored in a replay buffer. During training, the agent samples random mini-batches from this buffer to break temporal correlations between consecutive experiences. This process improves data efficiency and stabilizes the learning updates.

Critic Update. The critic network, which estimates the action-value function, is trained to minimize the mean-squared error between its predictions and a target value computed from the reward and the next-state estimate. This iterative update refines the critic's ability to evaluate how good a particular action is in a given state.

Actor Update. The actor network, representing the policy, is updated using the gradient of the critic's estimated value with respect to the action. This allows the actor to adjust its parameters in the direction that increases the expected return, effectively learning how to choose better actions based on the critic's feedback.

Target Network Update. To prevent instability caused by rapidly changing value targets, DDPG maintains separate target networks for both the actor and critic. These target networks are updated gradually using a soft-update mechanism that blends the new weights with previous ones at a small rate. This slow update process smooths learning and improves convergence.

Exploration. Because the policy in DDPG is deterministic, an external noise process is added to the actor's output during training to encourage exploration. This noise is often generated using an Ornstein–Uhlenbeck process, which produces temporally correlated fluctuations suitable for physical control tasks that require smooth changes in action values.

2.3.2 Advantages and Limitations

DDPG offers several advantages in the context of continuous control problems. It can efficiently handle high-dimensional continuous action spaces, making it suitable for robotic and autonomous systems that require smooth and precise control input [32]. Using deterministic policy gradients, DDPG achieves higher sample efficiency compared to purely

stochastic policy methods, as it directly optimizes a deterministic policy without relying on action sampling noise. The integration of experience replay and target networks further stabilizes the learning process by breaking the correlations between consecutive samples and providing slowly updated targets.

Despite these strengths, DDPG also exhibits notable limitations. The algorithm is highly sensitive to hyperparameter choices, network initialization, and exploration noise. Improper tuning can lead to divergence, suboptimal convergence, or overly conservative behavior. Additionally, the deterministic nature of its policy makes exploration challenging, particularly in environments with sparse rewards or multi-model dynamics. The stability of DDPG heavily depends on the careful implementation of target network updates and replay buffer management.

In practice, DDPG has been successfully applied to a wide range of domains, such as robotic manipulation, autonomous driving, UAV control, and energy management systems, where continuous and precise control is essential [17, 16, 13].

Chapter 3

Autonomous Parking Robot using DDPG

3.1 Introduction

3.1.1 Background

Autonomous parking represents a fundamental benchmark problem for intelligent vehicle control. In its simplest form, the task requires moving the vehicle from a given initial state to a specified target pose with both position and orientation accuracy. Unlike lane following or trajectory tracking, the parking problem demands precise maneuvering in a confined region, making it an ideal testbed for evaluating reinforcement learning algorithms in continuous control domains. As noted by Yuan *et al.* [33], automated parking in narrow spaces requires rapid and accurate generation of collision-free trajectories and handles multiple narrow-space constraints, underscoring the complexity and precision demands of the task.

In this section, the parking problem is formulated without obstacles or environmental uncertainties. The objective is to demonstrate that a reinforcement learning agent, trained with the DDPG algorithm, can learn effective control policies to drive the vehicle from a randomly initialized location to a predefined goal location. To investigate this, three levels of modeling fidelity are considered. The kinematic model provides a simplified baseline suitable for low-speed maneuvers. A dynamic model is then introduced to capture inertial

effects and actuator dynamics. Finally, the model is extended with noisy inputs to evaluate the robustness of the learned policies under more realistic conditions.

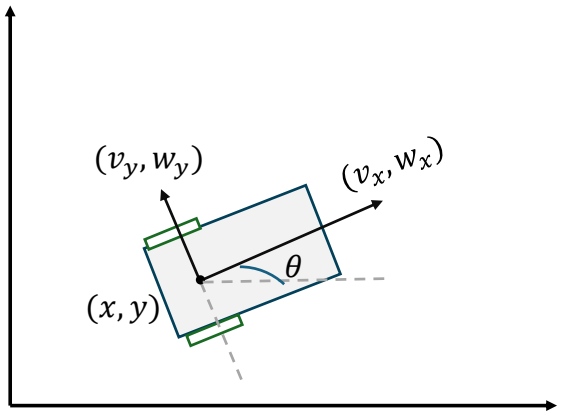


Figure 3.1: Autonomous model's states (x, y, θ) and velocity inputs.

3.1.2 Task Objective

The parking task is formulated in a two-dimensional plane with a bounded square workspace of size 20×20. The vehicle is initialized at random positions and orientations within this workspace at the beginning of each episode. The mission goal is to drive the vehicle to a designated target pose, defined by a specific position and orientation.

The episode terminates when the vehicle reaches the goal pose within a tolerance region, when the maximum step limit is exceeded, or when the vehicle leaves the workspace boundaries. This setup provides a controlled environment in which the agent must learn to generalize across different initial conditions while consistently converging to the same final target.

3.1.3 DDPG Agent Setup

The DDPG agent is designed to solve the autonomous parking problem formulated in the Simulink environment. The agent operates with continuous state and action spaces, which makes it well suited for vehicle control tasks that require smooth commands.

State and Action Spaces

For the kinematic model, the observation vector consists of five states: $(x, y, \theta, v, \omega)$, where x and y represent the position of vehicle, θ for the orientation angle, v for the linear velocity, and ω for the angular velocity. The action vector is two-dimensional, consisting of the commanded v and ω.

For the dynamic model, the observation vector is extended to seven states: $(x, y, \theta, v, \omega, u_L, u_R)$, where u_L and u_R represent the wheel-input forces applied to the left and right wheels. The action vector is also two-dimensional, consisting of the commanded wheel forces (u_L, u_R). By including both wheel inputs in the state and action spaces, the agent is provided with information about the vehicle's actuation dynamics in addition to its pose and velocities, which makes the control task more realistic and challenging.

For compatibility with the DDPG algorithm, both actions are normalized to the interval $[-1, 1]$. This normalization is required because the actor network is designed to output bounded actions, which improves numerical stability during training and prevents divergence. The normalized actions can then be rescaled to match the physical actuator limits of the vehicle model.

Actor and Critic Networks.

The actor network receives the observation vector as input and maps it to continuous control actions. It is composed of two fully connected hidden layers with 128 neurons, each followed by an output layer that generates the linear and angular velocity commands.

The critic network is designed to estimate the state-action value. It has two input channels: one for the observation vector and one for the action vector. Each channel is processed through fully connected layers before being merged into a shared pathway. This combined representation is then passed through an additional fully connected layer and a final output neuron, producing the scalar Q-value of the state-action pair.

Both the actor and the critic are implemented using the MATLAB Reinforcement Learning Toolbox and trained jointly within the DDPG framework.

Exploration Strategy.

During training, exploration noise is added to the actions to encourage diverse experiences. A Gaussian noise process with decaying variance is employed, starting with a standard deviation of 0.1 and decreasing to 0.01 over the course of training. This enables broad exploration in the early stages while promoting policy refinement later on.

Training Parameters.

The agent is trained using experience replay with a buffer size of 10^6. Mini-batches of size 512 are sampled for updates. Both actor and critic networks are trained with the Adam optimizer using a learning rate of 10^{-3}. The discount factor is set to 0.995, and target networks are updated using a smoothing factor of 10^{-3}. Training is conducted for a maximum of 10,000 episodes, each lasting up to 30 seconds of simulated time. Early stopping is triggered if the average reward exceeds 1000, and agents are saved when the episodic reward reaches 400 or higher.

Environment Reset.

At the beginning of each episode, the vehicle is initialized at a random location on a circle of radius 10 within the 20×20 workspace. Its orientation is also randomized. This ensures that the agent encounters a wide range of initial conditions, which improves the generalization capability of the learned policy.

3.2 Autonomous Parking with Kinematic Model

3.2.1 Kinematic Model

The vehicle is modeled using a unicycle kinematic representation, which provides a simplified yet effective description of motion for low-speed maneuvers such as parking. This model captures the essential nonholonomic constraint of wheeled vehicles, namely that lateral motion cannot be commanded directly.

The state vector is defined as

$$s = \begin{bmatrix} x & y & \theta & v & \omega \end{bmatrix}^T, \tag{3.1}$$

where x and y denote the position of the vehicle in the global coordinate frame, θ is the orientation (heading angle), v is the linear velocity, and ω is the angular velocity.

The control inputs to the system are the linear velocity v and the angular velocity ω. The vehicle's motion is then described by the following equations [34]:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega. \tag{3.2}$$

This formulation assumes negligible slip and dynamic effects, which is reasonable for low-speed parking maneuvers. In the reinforcement learning framework, the agent outputs continuous values of v and ω within the normalized interval $[-1, 1]$, which are then scaled to match the physical limits of the vehicle. This kinematic model serves as the baseline for evaluating the DDPG agent before extending the analysis to more realistic dynamic and noisy models in later sections.

3.2.2 Reward Function Design

The design of the reward function plays a crucial role in reinforcement learning, as it guides the agent toward the desired parking behavior. In this study, the reward is designed to encourage convergence to the target pose while penalizing inefficient or undesirable actions. The total reward R is expressed as the weighted sum of several components:

$$R = R_{\text{pos}} + R_{\theta} + P_{\text{pos}} + P_{\text{boundary}} + P_{\text{smooth}} + P_{\text{turn}}, \tag{3.3}$$

where each term is defined as follows.

Position Reward: The primary objective is to minimize the squared distance to the target position $(x_{\text{tar}}, y_{\text{tar}}) = (0, 0)$. A shaping function is used to provide higher reward as the agent

approaches the goal:

$$R_{\text{pos}} = \frac{10}{1 + 3(x - x_{\text{tar}})^2 + 3(y - y_{\text{tar}})^2}. \tag{3.4}$$

Position Penalty: To further discourage large deviations from the target, a quadratic penalty term is included:

$$P_{\text{pos}} = -0.01\big(3(x - x_{\text{tar}})^2 + 3(y - y_{\text{tar}})^2\big). \tag{3.5}$$

Orientation Reward: If the vehicle reaches sufficiently close to the target position and aligns with the target orientation $\theta_{\text{tar}} = 0$, an additional bonus is given:

$$R_\theta = \begin{cases} 5, & \text{if } \big((x, y) \approx (0, 0) \text{ and } |\theta - \theta_{\text{tar}}| < 0.5\big), \\ 0, & \text{otherwise.} \end{cases} \tag{3.6}$$

Boundary Penalty: If the agent leaves the predefined workspace, a large penalty is applied:

$$P_{\text{boundary}} = -100. \tag{3.7}$$

Action Smoothness Penalty: To encourage smooth control, large changes in velocity are penalized:

$$P_{\text{smooth}} = -0.01\,(\Delta v)^2, \tag{3.8}$$

where Δv is the change in linear velocity between successive actions.

Sharp Turning Penalty: Similarly, sharp variations in angular velocity are penalized:

$$P_{\text{turn}} = -0.01\,(\Delta \omega)^2. \tag{3.9}$$

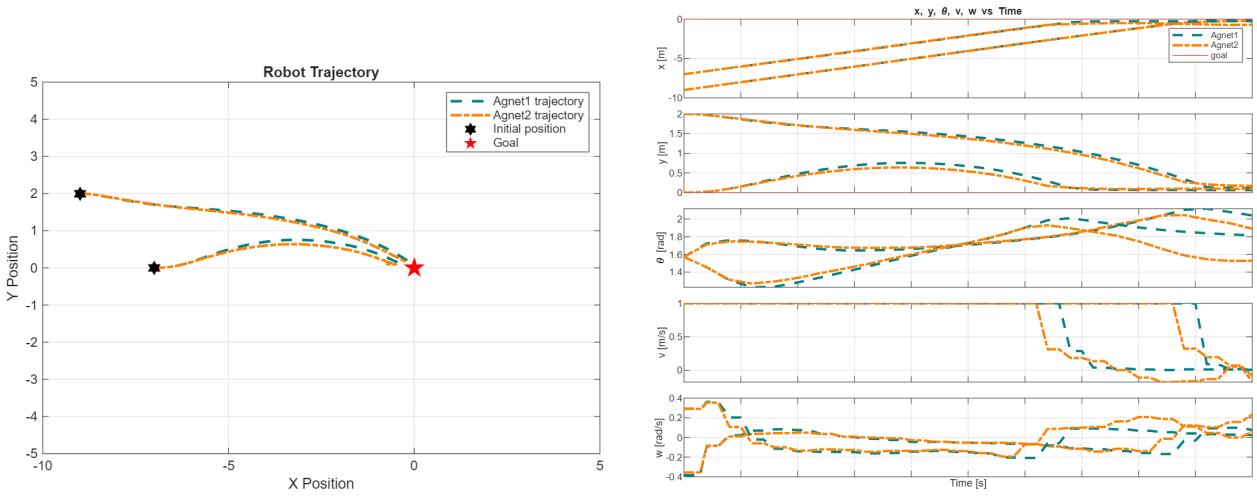
Overall Reward: The combination of these terms encourages the agent to approach the target efficiently, align with the correct orientation, and use smooth, stable control actions while avoiding boundary violations.

3.2.3 Results and Discussion

The performance of the DDPG agents under the kinematic model is illustrated in Fig. 3.2. Figure 3.2a shows the trajectories of Agent 1 and Agent 2 from different initial positions to the target pose at the origin. Both agents successfully converge to the goal region, demonstrating the ability of the learned policy to generalize across different starting conditions. The trajectories are smooth, and the final positions align well with the target, indicating that the reward function effectively guides the agent toward both position and orientation accuracy.

Figure 3.2b presents the evolution of the state variables during the parking maneuver. The position errors in both the x and y directions decrease steadily as the vehicle approaches the goal, while the orientation error converges to zero. The linear and angular velocities also stabilize, which reflects the smooth control actions encouraged by the penalties on abrupt velocity changes in the reward function.

Overall, the results confirm that the DDPG agent trained under the kinematic model can learn effective parking policies. Both agents exhibit consistent performance, suggesting that the policy is not over-fitted to a particular initial condition. This validates the kinematic model as a suitable baseline for autonomous parking tasks before extending the analysis to more complex models that incorporate dynamics and noise.



(a) Trajectory. (b) State evolution.

Figure 3.2: Robot trajectory and state evolution under the kinematic model for Agent 1 and Agent 2.

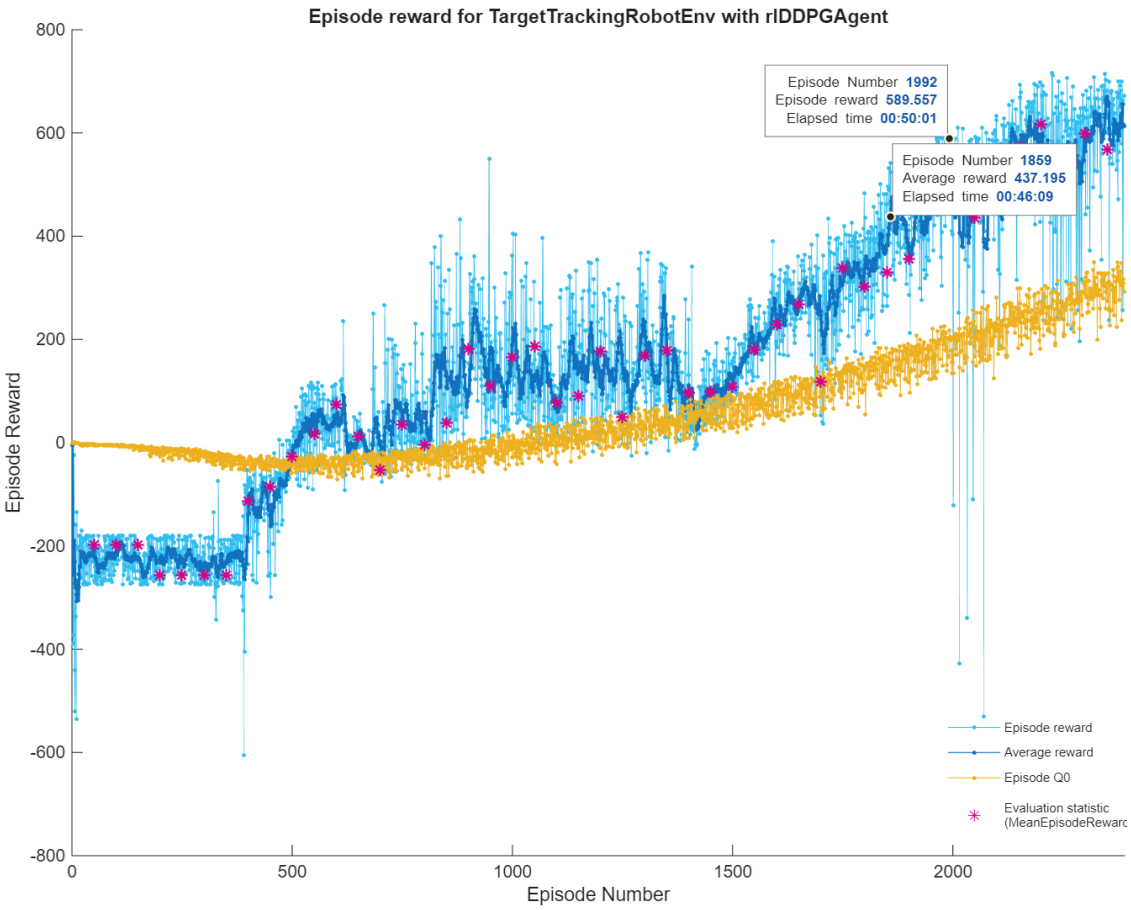


Figure 3.3: Training performance of the RL DDPG Agent in the kinematic model's environment.

3.3 Autonomous Parking with Dynamic Model

3.3.1 Dynamic Model

To account for the inertial properties of the vehicle, a dynamic model is used in place of the simplified kinematic formulation. This model incorporates mass, moment of inertia, and wheel actuation, resulting in a more realistic description of the system behavior.

The state vector is defined as

$$s = \begin{bmatrix} x & y & \theta & v & \omega & u_L & u_R \end{bmatrix}^T, \tag{3.10}$$

where x and y denote the vehicle position in the global frame, θ is the orientation angle,

v is the linear velocity, ω is the angular velocity, and u_L and u_R represent the forces applied to the left and right wheels, respectively.

The system parameters are: vehicle mass $m = 10$ kg, rotational inertia $I = 5$ kg $\cdot$ m^2, wheel radius $r = 0.1$ m, and half of the wheelbase $R_i = 0.2$ m.

The vehicle dynamics are governed by the following equations[35]:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega, \tag{3.11}$$

$$\dot{v} = \frac{1}{mr}(u_L + u_R), \qquad \dot{\omega} = \frac{2R_i}{Ir}(u_L - u_R). \tag{3.12}$$

This formulation links the wheel-level forces directly to the translational and rotational accelerations of the vehicle. By including the wheel inputs u_L and u_R in the state vector, the agent is fully aware of the actuation dynamics, making the control task more challenging compared to the kinematic model. The dynamic model provides a more realistic benchmark for testing reinforcement learning approaches to autonomous parking.

3.3.2 Reward Function Design

For the dynamic model, the reward function is constructed to balance accurate goal-reaching with smooth and physically feasible control actions. The overall reward is defined as the sum of multiple components:

$$R = R_{\text{pos}} + P_{\text{pos}} + R_\theta + P_{\text{boundary}} + P_{\text{smooth}} + P_{\text{turn}} + P_\omega + P_{\text{torque}}, \tag{3.13}$$

where each term is described below.

Position Reward and Penalty: The agent is encouraged to minimize the squared distance to the target position $(x_{\text{tar}}, y_{\text{tar}}) = (0, 0)$. A shaping reward is provided as

$$R_{\text{pos}} = \frac{10}{1 + 3(x - x_{\text{tar}})^2 + 3(y - y_{\text{tar}})^2}, \tag{3.14}$$

while a quadratic penalty discourages large deviations:

$$P_{\text{pos}} = -0.01\big(3(x - x_{\text{tar}})^2 + 3(y - y_{\text{tar}})^2\big). \tag{3.15}$$

Orientation Reward: If the vehicle is within a small neighborhood of the target and aligned with the desired orientation $\theta_{\text{tar}} = 0$, a bonus is awarded:

$$R_\theta = \begin{cases} 5, & \text{if } \|(x, y)\| < 0.5 \text{ and } |\theta - \theta_{\text{tar}}| < 0.5, \\ 0, & \text{otherwise.} \end{cases} \tag{3.16}$$

Boundary Penalty: Leaving the workspace results in a large penalty:

$$P_{\text{boundary}} = -100. \tag{3.17}$$

Action Smoothness Penalty: To promote gradual changes in linear velocity, abrupt variations are penalized:

$$P_{\text{smooth}} = -0.03(v - v_0)^2, \tag{3.18}$$

where v_0 is the linear velocity from the previous step.

Sharp Turning Penalty: Similarly, sudden changes in angular velocity are discouraged:

$$P_{\text{turn}} = -0.05(w - w_0)^2, \tag{3.19}$$

where w_0 is the angular velocity from the previous step.

Angular Velocity Penalty: Excessively large angular velocities are penalized to prevent unstable spinning:

$$P_\omega = \begin{cases} -0.01w^2, & |w| > 1.0, \\ 0, & \text{otherwise.} \end{cases} \tag{3.20}$$

Torque Penalties: Two additional penalties regulate the wheel inputs. First, a penalty on the overall magnitude of the wheel torques:

$$P_{\text{torque,total}} = -0.01(u_L^2 + u_R^2), \tag{3.21}$$

and second, a penalty on large differences between the left and right wheels:

$$P_{\text{torque,diff}} = -0.01(u_L - u_R)^2. \tag{3.22}$$

Overall Rewards: The combination of these terms encourages the agent to reach the parking goal accurately, maintain smooth velocity and turning profiles, and avoid unrealistic actuation. Compared to the kinematic case, the dynamic reward design places greater emphasis on smoothness and torque regulation, which reflects the added complexity of controlling a vehicle with inertia and input constraints.

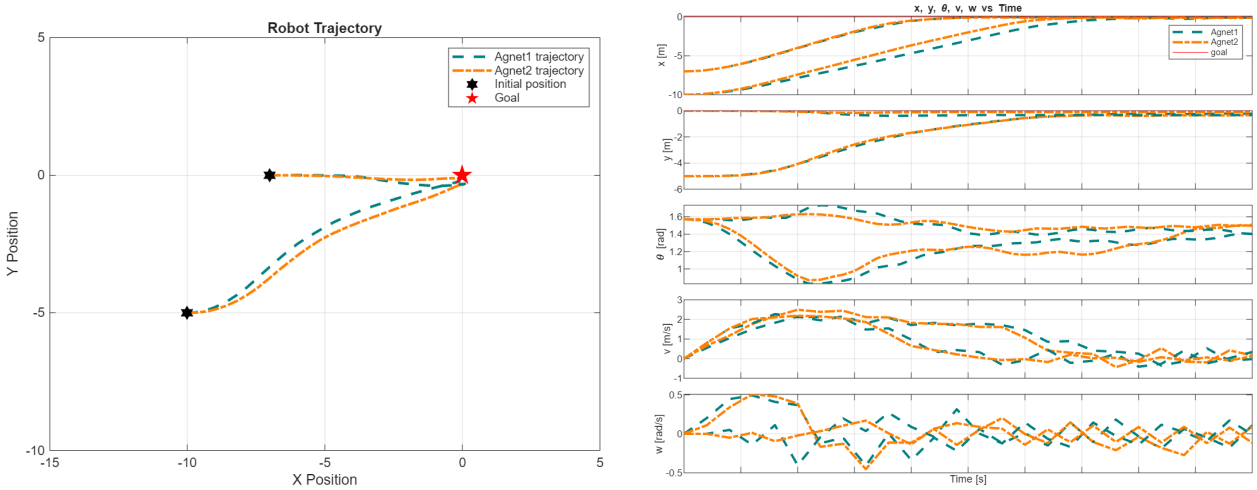
3.3.3 Results and Discussion

The performance of the DDPG agents under the dynamic model is illustrated in Fig. 3.4. Figure 3.4a shows the trajectories of Agent 1 and Agent 2 from different initial positions to the target pose at the origin. Both agents are able to reach the goal successfully, confirming that the dynamic model policies converge despite the additional complexity of inertia and torque-driven actuation. Compared to the kinematic model, the trajectories appear less direct and contain small oscillations, which reflects the increased difficulty of regulating acceleration and wheel inputs.

Figure 3.4b presents the evolution of the vehicle states. The position variables x and y converge smoothly to the target values, while the orientation angle θ also stabilizes near the goal pose. However, compared with the kinematic case, the convergence of both v and ω is less stable, exhibiting oscillations before settling. This behavior can be attributed to the dynamic effects: the agent must learn to compensate for inertia, torque balance, and actuation delays, which makes precise control more challenging.

Overall, the results demonstrate that the DDPG agent can still learn effective parking

policies under the dynamic model. While the kinematic model produced smoother and more direct trajectories, the dynamic model captures more realistic vehicle behavior, including overshoot and oscillations in velocity and angular velocity. These results highlight the trade-off between simplicity and realism: the kinematic model provides a clean baseline, while the dynamic model exposes the agent to challenges closer to real-world conditions.



(a) Trajectory.

(b) State evolution.

Figure 3.4: Robot trajectory and state evolution under the dynamic model for Agent 1 and Agent 2.

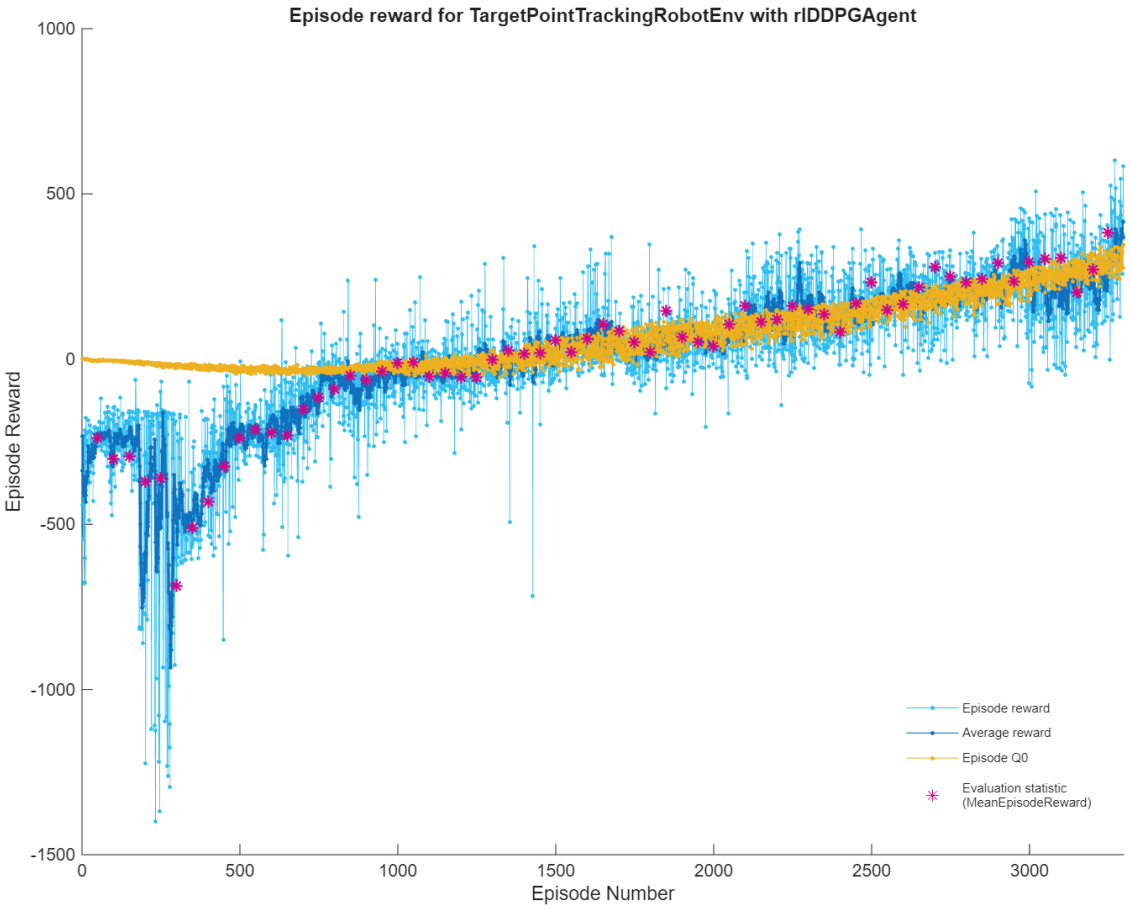


Figure 3.5: Training performance of the RL DDPG Agent in the dynamic model's environment.

3.4 Autonomous Parking with Dynamic Model under Noisy Inputs

3.4.1 Noise Modeling

To assess the robustness of the learned policy, band-limited white noise was added to the wheel input forces u_L and u_R in the Simulink environment. Band-limited white noise is a random signal with zero mean and constant power spectral density within a specified frequency range. Unlike ideal white noise, which has infinite bandwidth and is physically unrealizable, the band-limited version constrains the frequency content, making the disturbances bounded and more representative of actuator uncertainties in real systems [36].

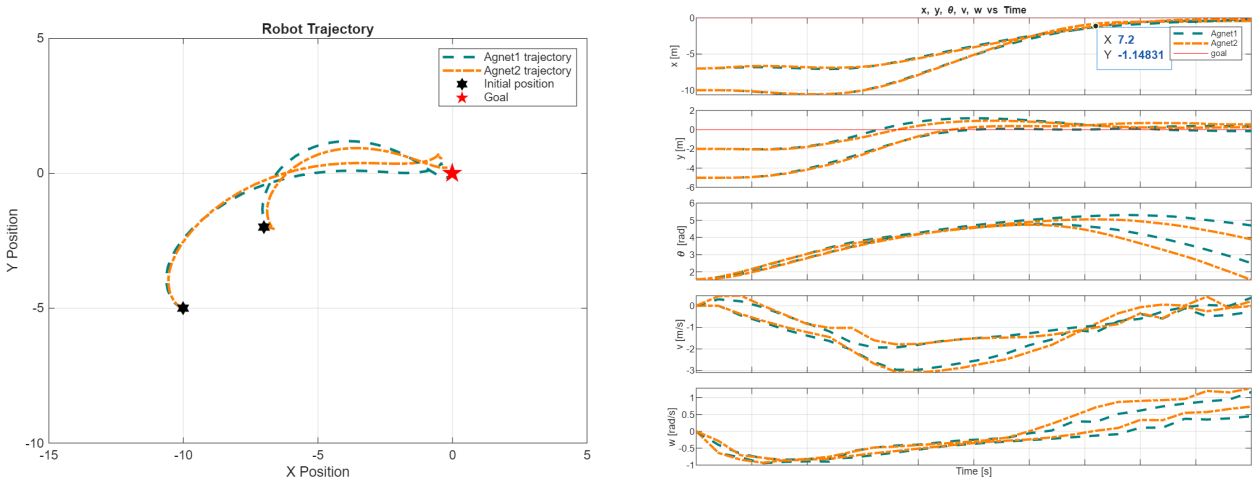
The reward function design remains identical to the Section 3.3. By keeping the same reward structure, the difference in performance can be attributed directly to the effect of noisy inputs rather than to changes in the learning objective. This allows for a clear comparison between the noise-free and noisy dynamic environments, highlighting the policy's robustness under actuation disturbances.

3.4.2 Results and Discussion

The results of the DDPG agents under the dynamic model with noisy wheel inputs are shown in Fig. 3.6. Figure 3.6a illustrates the trajectories of Agent 1 and Agent 2 when band-limited white noise is injected into the left and right wheel torques. Despite the disturbances, both agents are still able to reach the target parking pose successfully. Compared with the noise-free dynamic model, the trajectories exhibit larger deviations and less smooth curvature, indicating the effect of input uncertainty on path execution.

Figure 3.6b presents the evolution of the state variables. The position and orientation states converge to the goal, but the velocity profiles (v and ω) display stronger oscillations than in the deterministic case. These fluctuations reflect the agent's attempts to correct for noisy actuation, particularly in angular velocity control. Nevertheless, the added penalties in the reward function for large torque magnitudes and differences help maintain stability, preventing divergence or instability in the trajectories.

Overall, the comparison shows that the learned policy retains robustness against actuator-level noise. While the dynamic model produced smoother results in the absence of disturbances, the noisy case highlights realistic challenges that reinforcement learning controllers must overcome. The ability of both agents to park reliably under noise demonstrates the potential of DDPG to handle uncertain inputs, although at the cost of increased control effort and less stable velocity responses.



(a) Trajectory.

(b) State evolution.

Figure 3.6: Robot trajectory and state evolution under the dynamic model with noise input for Agent 1 and Agent 2.

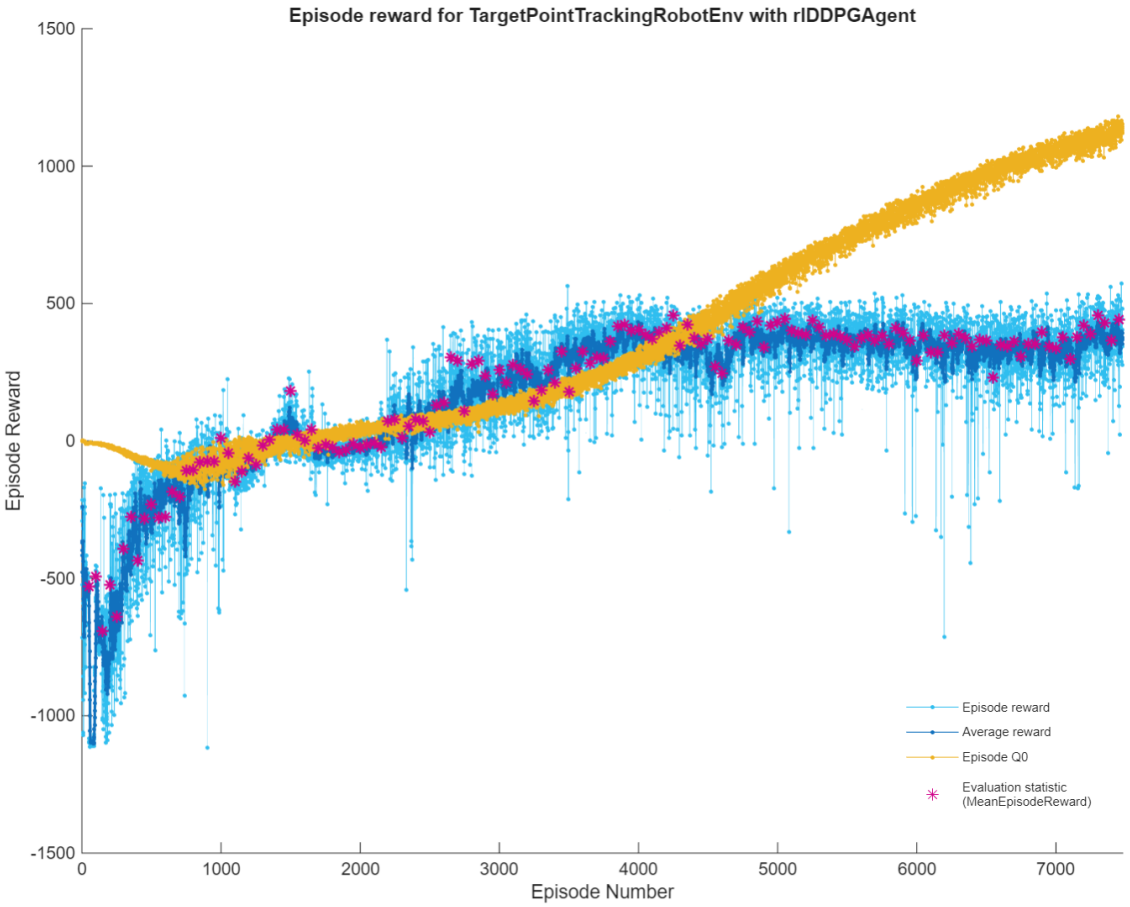


Figure 3.7: Training performance of the RL DDPG Agent in the dynamic noise model's environment.

Chapter 4

Safety in Autonomous Control

4.1 Introduction

4.1.1 Background

Safety is a central requirement in autonomous systems, especially in safety-critical applications such as autonomous driving, robotic navigation, and aerial vehicles. Even small violations of safety constraints, such as collisions or entering restricted regions, can result in system failure or hazardous outcomes. RL has shown strong adaptability in learning complex behaviors from interaction, but its trial-and-error nature provides no guarantee of safety during training or execution. This motivates the need for additional mechanisms that can enforce safety constraints.

Safety-critical control methods such as CBF and CLF enforce system safety by guaranteeing forward invariance of a safe set under appropriate control inputs [37]. CBF provides a model-based mathematical framework for guaranteeing safety. By constructing a barrier function that defines a safe set, the method ensures that system states remain within this set for all time. CBF achieve this by imposing inequality constraints on admissible control inputs, effectively filtering unsafe actions while still allowing flexibility within the safe region [38, 39].

When compared directly, RL and CBF represent two complementary paradigms. RL excels in handling high-dimensional, uncertain, and nonlinear systems but lacks inherent

safety guarantees. CBFs, on the other hand, provide provable safety but rely on simplified system models and may be less effective in highly uncertain environments[40]. Integrating RL with CBF offers the potential to combine the adaptability of data-driven learning with the rigorous safety guarantees of barrier functions.

4.1.2 Obstacle Environment Setup and Goal

To evaluate safety mechanisms, a uniform circular obstacle in the environment is considered. The agent is tasked with reaching a specified goal position while avoiding collisions with the obstacle. This scenario captures one of the most fundamental safety challenges in autonomous navigation—maintaining progress toward the objective while respecting hard safety constraints. By using a simple yet representative obstacle setup, the problem becomes tractable for analysis while still highlighting the strengths and limitations of different approaches. This environment therefore enables a direct comparison between a purely RL-based obstacle avoidance strategy and a CBF-based method, providing valuable insights into how learning-based and model-based safety frameworks perform under identical conditions.

4.2 Obstacle Avoidance using Reinforcement Learning Only

4.2.1 Reward Function Design

In this section, the focus is on modifying the reward function to enable the agent to perform the obstacle avoidance task. The updated reward function builds upon the formulation used in the dynamic model (Eq. 3.2) by adding a penalty term that accounts for the presence of obstacles. This ensures that the agent learns not only to reach the goal pose but also to avoid unsafe regions around obstacles.

The distance between the robot position (x, y) and the obstacle center $(x_{\text{obs}}, y_{\text{obs}})$ is defined as

$$d_{\text{obs}} = \sqrt{(x - x_{\text{obs}})^2 + (y - y_{\text{obs}})^2}. \tag{4.1}$$

If the robot approaches the obstacle boundary—defined by the obstacle radius r_{obs} plus a safety margin Δr—a penalty is applied as

$$P_{\text{obs}} = -\frac{10}{1 + 5\,(d_{\text{obs}} - r_{\text{obs}} - \Delta r)^2}. \tag{4.2}$$

This penalty increases rapidly as the robot enters the unsafe region surrounding the obstacle, effectively discouraging trajectories that approach or collide with it.

The overall reward for the obstacle-avoidance task is therefore

$$R = R_{\text{pos}} + P_{\text{pos}} + R_\theta + P_{\text{boundary}} + P_{\text{smooth}} + P_{\text{turn}} + P_\omega + P_{\text{torque}} + P_{\text{obs}}, \tag{4.3}$$

where the terms $R_{\text{pos}}, P_{\text{pos}}, R_\theta, P_{\text{boundary}}, P_{\text{smooth}}, P_{\text{turn}}, P_\omega$, and P_{torque} follow the definitions in Eq. 3.2.

By incorporating the obstacle penalty P_{obs}, the agent is explicitly guided to maintain a safe distance from obstacles while still achieving the navigation goal. This modification allows reinforcement learning to handle safety constraints through a learned, reward-based approach.

4.2.2 Results and Discussion

Figure 4.1 illustrates the trajectories of Agent 1 and Agent 2 navigating an environment containing a circular obstacle. Both agents start from different initial positions and successfully reach the designated goal pose. The learned policies generate smooth, continuous motions that generally curve around the obstacle region while maintaining stable convergence to the goal for most of the locations.

Although both agents exhibit obstacle avoidance behavior for most of the locations, safety violations can still be observed in some locations. In certain areas, the trajectories collide with the obstacle's safety margin, indicating that the reward-based penalty encourages avoidance but cannot fully guarantee constraint satisfaction. This phenomenon reflects the inherent limitation of soft safety in reinforcement learning, where constraint enforcement depends on how strongly the penalty terms influence policy optimization.

Overall, the results demonstrate that reinforcement learning, when guided by a carefully designed reward function, can achieve adaptive and largely safe navigation in environments with static obstacles. However, since safety is enforced only through reward shaping, the learned policy still compromises strict safety in favor of smoother convergence or shorter travel time. To obtain hard safety guarantees, integrating a model-based safety mechanism such as a CBF can explicitly enforce state constraints during policy execution, ensuring provable safety even in uncertain or high-risk regions.

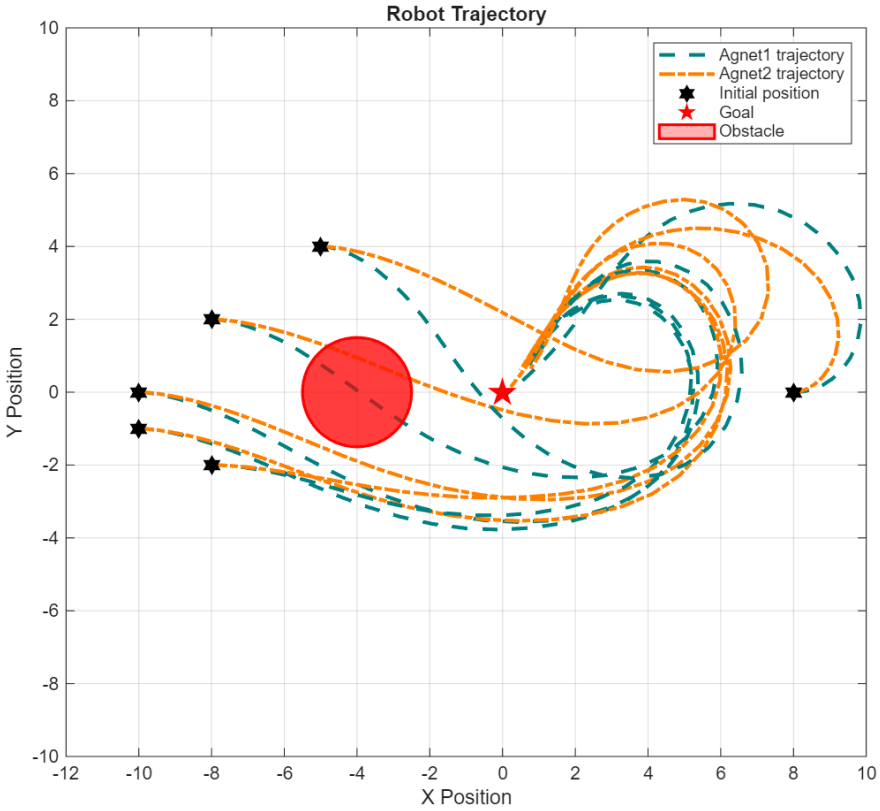


Figure 4.1: Robot trajectory with obstacle avoidance.

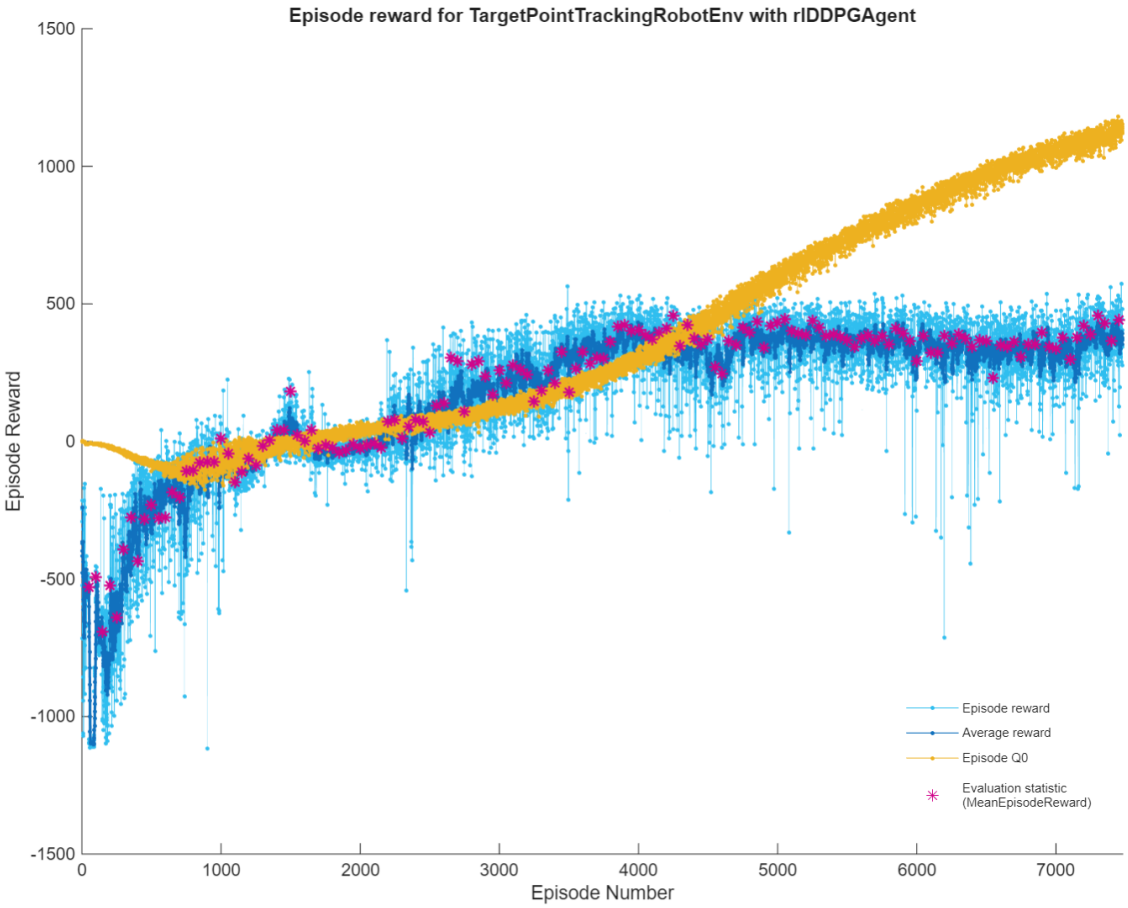


Figure 4.2: Training performance of the RL DDPG agent in the dynamic model environment with an obstacle.

4.3 Obstacle Avoidance using Control Barrier Functions

Motivation for a Safety Filter: While the reward-based approach enables adaptive obstacle avoidance, it cannot guarantee strict constraint satisfaction in all cases. Therefore, to provide mathematically provable safety, the following section introduces CBF as a real-time safety filter that modifies the RL agent's control commands only when necessary. In this section, the CBF is integrated into the RL control loop, as illustrated in Figure 4.3, to complete the obstacle avoidance task.

4.3.1 Control Barrier Functions for Kinematic Model

CBF provide a model-based framework to enforce safety constraints in control systems. Unlike reinforcement learning, which encourages safety indirectly through reward shaping, CBF guarantee forward invariance of a safe set by imposing inequality constraints on the control input.

For the unicycle kinematic model, the state vector is defined as

$$s = \begin{bmatrix} x & y & \theta \end{bmatrix}^T,$$

with dynamics,

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega,$$

where v and ω denote linear and angular velocities, respectively.

Consider a circular obstacle centered on $(x_{\text{obs}}, y_{\text{obs}})$ with radius r_{obs} and safety margin Δr. The safe set is defined as

$$\mathcal{C} = \Big\{ (x,y) \in \mathbb{R}^2 \;\Big|\; h(x,y) = (x - x_{\text{obs}})^2 + (y - y_{\text{obs}})^2 - (r_{\text{obs}} + \Delta r)^2 \geq 0 \Big\}. \tag{4.4}$$

The barrier function $h(x,y)$ ensures that the robot remains outside the unsafe region. The CBF condition

$$\dot{h}(x,y) + \alpha(h(x,y)) \geq 0 \tag{4.5}$$

preserves the forward invariance of the safe set, which $\alpha(\cdot)$ is an extended class$\mathcal{K}$ function (commonly $\alpha(h) = \gamma h$ with $\gamma > 0$).

At each time step, the safety filter solves the following quadratic program (QP):

$$u^*(x,y) = \arg\min_{u \in \mathcal{U}} \; \|u - u_{\text{des}}(x,y)\|^2 \tag{4.6}$$

subject to

$$L_f h(x,y) + L_g h(x,y) u \geq -\alpha(h(x,y)), \tag{4.7}$$

where $u_{\text{des}} = [v_{\text{des}}, \omega_{\text{des}}]^T$ the nominal control input is generated by the RL policy.

A closed-form solution to QP [38] is

$$u^*(x,y) = u_{\text{des}}(x,y) + u_{\text{safe}}(x,y), \tag{4.8}$$

with

$$u_{\text{safe}}(x,y) = \begin{cases} -\dfrac{L_g h(x,y)^\top}{L_g h(x,y) L_g h(x,y)^\top}\,\Psi(x;u_{\text{des}}), & \text{if } \Psi(x,y) < 0, \\ 0, & \text{if } \Psi(x,y) \geq 0, \end{cases} \tag{4.9}$$

and

$$\Psi(x,y) = L_f h(x,y) + L_g h(x,y) u_{\text{des}}(x,y) + \alpha h(x,y) \tag{4.10}$$

where $\alpha > 1$ is a design constant that determines the strictness of the safety constraint. For a uniform circle obstacle, the Lie derivative is given by:

$$L_f h(x,y) = 0 \tag{4.11}$$

$$L_g h(x,y) = \begin{bmatrix} 2(x - x_{\text{obs}})\cos\theta + 2(y - y_{\text{obs}})\sin\theta & 0 \end{bmatrix} = \begin{bmatrix} a & 0 \end{bmatrix} \tag{4.12}$$

The safety condition can therefore be expressed as

$$\Psi(x,y) = a u_{\text{des}} + \alpha h(x,y). \tag{4.13}$$

Accordingly, the safe control input is defined as

$$u_{\text{safe}}(x,y) = \begin{cases} -u_{des} - \dfrac{\alpha}{a} h(x,y), & \text{if } \Psi(x,y) < 0, \\ 0, & \text{if } \Psi(x,y) \geq 0, \end{cases} \tag{4.14}$$

and the resulting control correction applied to the RL command is

$$u^*(x,y) = \begin{cases} -\dfrac{\alpha}{a} h(x,y), & \text{if } \Psi(x,y) < 0, \\ 0, & \text{if } \Psi(x,y) \geq 0, \end{cases} \tag{4.15}$$

This formulation ensures that the control input remains as close as possible to the nominal RL command while enforcing safety. In practice, the CBF acts as a real-time safety filter, overriding unsafe commands only when the system approaches the obstacle boundary [38].

4.3.2 Model Setup

The overall obstacle-avoidance controller combines the RL policy with a CBF safety filter, as shown in Fig. 4.3. The RL agent produces a nominal control u_{des} based on observed states, while the CBF layer evaluates safety in real time and applies a corrective term u_{safe}. The applied control input is therefore

$$u^* = u_{\text{des}} + u_{\text{safe}}. \tag{4.16}$$

When the robot is far from the obstacle, the safety filter remains inactive ($u_{\text{safe}} = 0$), allowing the RL policy to operate freely. As the robot approaches the unsafe region, the CBF modifies the command to ensure the safety constraint as Eq.4.7 is satisfied at all times. This framework preserves the adaptability of reinforcement learning while introducing mathematically guaranteed safety.

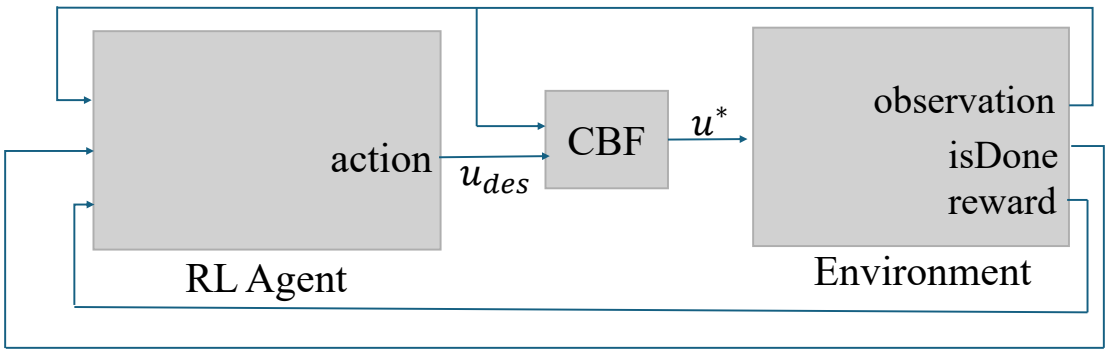


Figure 4.3: Integration of the reinforcement learning controller with the CBF safety filter.

4.3.3 Results and Discussion

Baseline Agent without CBF

Figure 4.4 shows the trajectory of the baseline RL agent uses the policy obtained from the training process described in Section 3.2, which was trained under the kinematic model without any safety constraints. The agent moves directly toward the goal, passing through the obstacle region, indicating that it lacks safety awareness and focuses solely on minimizing the distance to the goal.

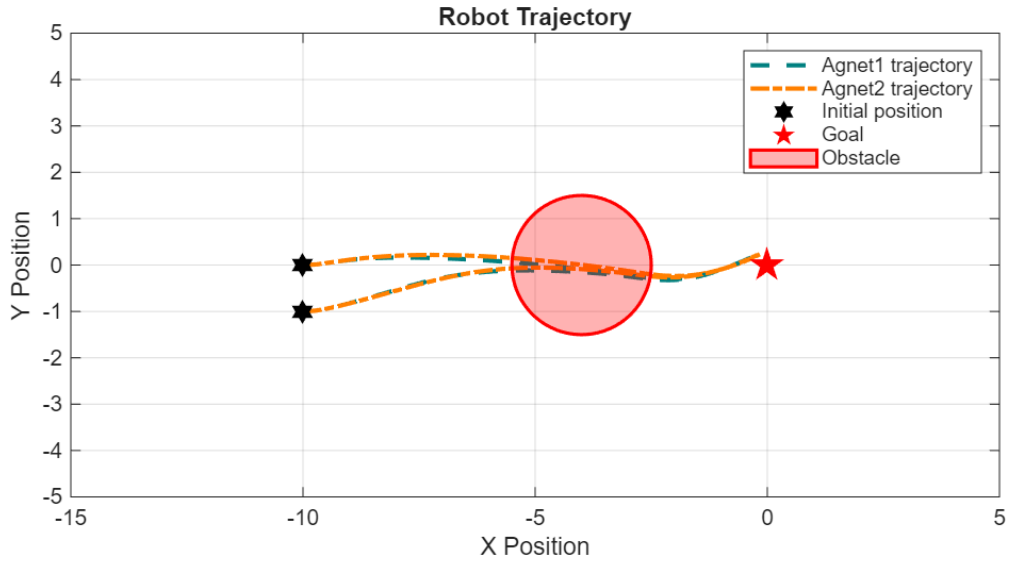


Figure 4.4: Trajectory of the baseline RL agent trained without CBF. The agent moves directly toward the goal without obstacle-avoidance capability.

Baseline Agent with CBF Filter

In this case, the baseline RL agent from Section 3.2 has no awareness of the obstacle information; however, a CBF-based safety filter is applied, as illustrated in Figure 4.3. As shown in Figure 4.5, the baseline pre-trained agent demonstrates a slight tendency to deviate from the obstacle's region, suggesting some incidental avoidance behavior learned during training. However, because the agent was never explicitly exposed to obstacle-related penalties or safety constraints, it fails to reach the target position and exhibits unstable and erratic motions near the obstacle boundary. The trajectory becomes disordered as the CBF repeatedly overrides unsafe control commands from the RL policy, leading to oscillations and loss of smoothness. Overall, although a CBF filter is applied during testing, the policy itself was not trained with safety awareness, resulting in abrupt and unstable corrections when approaching the obstacle.

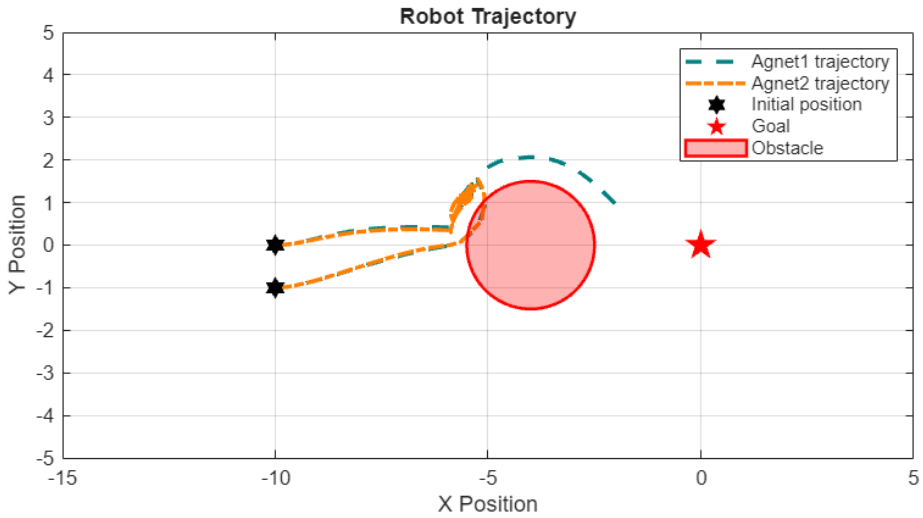


Figure 4.5: Trajectory of the pre-trained RL agent tested with a CBF filter. The agent tends toward the goal but shows irregular motion near the obstacle.

Re-trained Agent

In this case, the RL agent retrained with the CBF filter as illustrated in Figure 4.3. The reward functions, which are the same as Section 3.2 have no awareness of the obstacle. Figure 4.6 shows that the re-trained agent achieves smooth, consistent trajectories from multiple initial positions, maintaining a safe distance around the obstacle while reaching the goal. Compared with the pre-trained agent, the re-trained model exhibits better coordination with the safety filter, demonstrating stable, cooperative behavior between data-driven learning and model-based constraint enforcement.

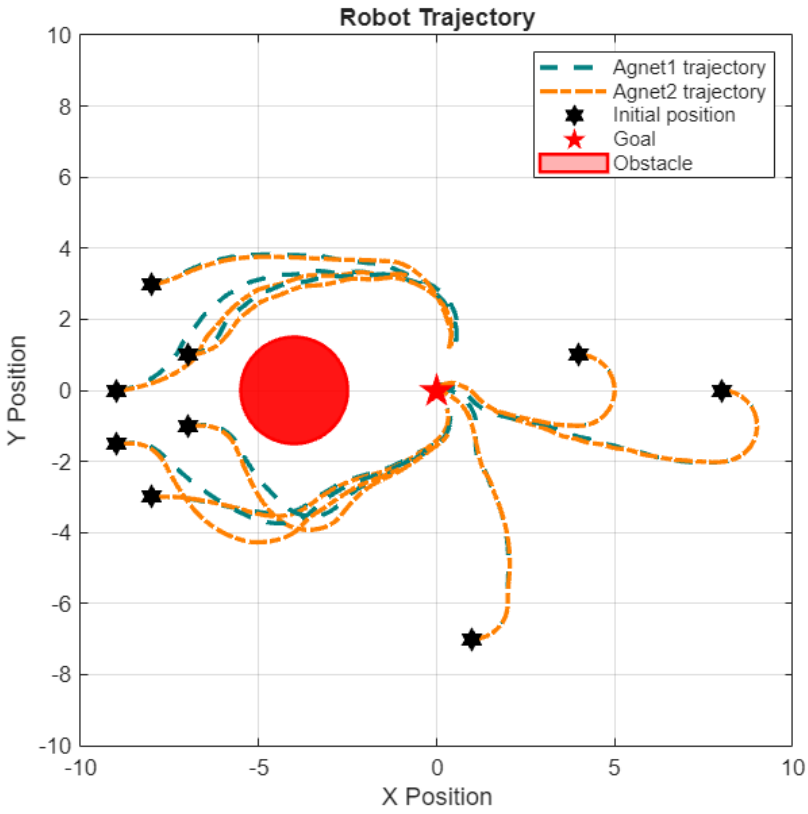


Figure 4.6: Trajectories of the re-trained RL agent with CBF filter. The agent maintains a safe distance from the obstacle while converging smoothly to the goal.

4.4 Comparative Discussion

The four cases—RL only, baseline agent only, baseline agent with CBF, and retrained agent with CBF add-on demonstrate a clear progression in safety performance. The RL-only agent demonstrates successful task completion in most trials; however, it fails to maintain safety in certain trials. The baseline RL agent ignores obstacles entirely. Adding a reward penalty introduces soft safety, producing avoidance behavior dependent on reward weighting. Applying a CBF filter during testing improves safety but causes abrupt control corrections since the policy was not trained with it. Finally, retraining the RL agent in a CBF-augmented environment results in smooth, reliable avoidance and formal safety guarantees. This progression confirms that integrating model-based CBF safety into the learning process achieves both adaptability and provable constraint satisfaction.

Chapter 5

Conclusions & Future Improvement

5.1 Conclusions

This thesis explored the application of RL and CBF in autonomous vehicle control, with a particular focus on parking and obstacle-avoidance scenarios. Through progressive modeling and simulation, the research demonstrated how data-driven and model-based methods can be integrated to achieve both adaptability and safety in control systems.

The study began by identifying the fundamental limitations of classical control approaches, such as their reliance on precise mathematical models and limited adaptability to nonlinearities, parameter variations, and unmodeled dynamics. RL was then introduced as a data-driven alternative that learns control policies through interaction, capable of handling high-dimensional and nonlinear systems without requiring explicit models. The DDPG algorithm was selected for its suitability in continuous control problems.

The autonomous parking task served as the primary benchmark for evaluating RL performance. Starting with a simplified kinematic model, the DDPG agent successfully learned to park the robot with smooth trajectories and stable convergence. The study then extended to a dynamic model incorporating vehicle inertia and torque-based actuation, confirming that the RL agent could adapt to more complex and realistic system dynamics. When band-limited noise was introduced to simulate actuator uncertainties, the trained policy maintained robustness and achieved reliable convergence, demonstrating the resilience of the learned controller against disturbances.

To address safety in autonomous control, the study examined obstacle-avoidance scenarios using both reinforcement learning and control barrier functions. The RL-based approach achieved "soft safety" by incorporating an obstacle penalty term in the reward function, which guided the agent to avoid collisions while pursuing the goal. However, the approach lacked strict safety guarantees, as avoidance performance depended on the weighting of the penalty. To overcome this limitation, CBF were introduced as a safety filter capable of ensuring forward invariance of a safe set. The integration of RL with CBF allowed the system to maintain adaptability while providing provable safety. Experimental results confirmed that an RL agent retrained in a CBF-augmented environment achieved smooth, reliable obstacle avoidance while maintaining formal safety guarantees.

Overall, this research demonstrated that reinforcement learning can effectively learn autonomous control policies for complex, nonlinear systems, and that combining RL with CBF enables a balance between adaptability, robustness, and guaranteed safety.

5.2 Future Improvement

While the presented framework provides a strong foundation for safe and adaptive autonomous control, several areas offer potential for future improvement and expansion:

1. **Sim-to-Real Transfer:**

   The current work is limited to simulation environments. Future research should focus on transferring the trained RL-CBF framework to physical robots or vehicles, addressing real-world issues such as sensor noise, actuator delay, and modeling mismatch.

2. **Multi-Agent and Dynamic Obstacles:**

   The experiments considered a single robot navigating around a static obstacle. Extending the approach to multi-agent settings or dynamic obstacles would introduce challenges in real-time coordination and prediction, further testing the scalability of the RL–CBF architecture.

3. **Adaptive Safety Layers:**

   While the CBF provides static safety constraints, future studies could develop adap-

tive or learning-based barrier functions that adjust safety limits based on uncertainty estimation or changes in the online environment.

4. **Hybrid Control and Optimization:**
   Combining RL with MPC or other optimization-based methods could improve both performance and interpretability, particularly in scenarios requiring explicit trajectory planning under constraints.

5. **Energy Efficiency and Control Smoothness:**
   Future work could refine the classification of rewards to include energy or minimization of control effort, enabling more efficient and smoother control suitable for real-world applications such as autonomous parking, drone navigation, and mobile robotics.

In summary, this thesis establishes a foundation for integrating data-driven reinforcement learning with model-based safety assurance methods. The continued development of such hybrid systems promises significant advances in the design of safe, intelligent, and autonomous control architectures capable of operating reliably in uncertain and dynamic environments.

Appendix A

Appendix

A.1 Source Code Repository

All MATLAB and Simulink source codes, trained agents, and simulation environments used in this thesis are available at the following GitHub repository:

`https://github.com/Junquan-Wu/RL_Practice`

The repository includes:

- Foundational RL practice code.
- Environment setup files for dynamic and kinematic models.
- Reward function, training and simulation configuration scripts.
- DDPG and CBF integration examples.
- Saved agents and trajectory data for reproducibility.

Bibliography

[1] K. Ogata, *Modern Control Engineering*. Upper Saddle River, NJ: Prentice Hall, 5th ed., 2010.

[2] F. L. Lewis and D. Vrabie, *Reinforcement learning and adaptive dynamic programming for feedback control*. IEEE, 2009.

[3] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Belmont, MA: Athena Scientific, 2019.

[4] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[5] C. V. Hollot, D. P. Looze, and A. C. Bartlett, "Parametric uncertainty and unmodeled dynamics: Analysis via parameter space methods," *Automatica*, vol. 26, no. 2, pp. 269–282, 1990.

[6] M. Krstić, J. Sun, and P. V. Kokotović, "Robust control of nonlinear systems with input unmodeled dynamics," *IEEE Transactions on Automatic Control*, vol. 41, no. 6, pp. 913–920, 1996.

[7] J.-P. Richard, "Time-delay systems: an overview of some recent advances and open problems," *Automatica*, vol. 39, no. 10, pp. 1667–1694, 2003.

[8] L. Liu, S. Tian, D. Xue, T. Zhang, Y. Chen, and S. Zhang, "A review of industrial mimo decoupling control," *International Journal of Control, Automation and Systems*, vol. 17, 04 2019.

[9] K. J. Åström and L. Rundqwist, "Integrator windup and how to avoid it," in *1989 American Control Conference*, pp. 1693–1698, 1989.

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

[11] B. Wang and A. Loria, "Leader-follower formation and tracking control of underactuated surface vessels," *arXiv preprint arXiv:2407.18844*, July 2024.

[12] S. Zhao, *Mathematical Foundations of Reinforcement Learning*. Springer Nature Singapore, 2025.

[13] I. Mandralis, R. M. Murray, and M. Gharib, "Quadrotor morpho-transition: Learning vs model-based control strategies," *arXiv preprint*, vol. arXiv:2506.14039, 2025.

[14] C.-H. Pi, W.-Y. Ye, and S. Cheng, "Robust quadrotor control through reinforcement learning with disturbance compensation," *Applied Sciences*, vol. 11, no. 7, p. 3257, 2021.

[15] D. Kim, J. D. Lee, H. Bang, and J. Bae, "Reinforcement learning-based fault-tolerant control for quadrotor with online transformer adaptation," *arXiv preprint arXiv:2505.08223*, 2025.

[16] T. Y. Trad, K. Choutri, M. Lagha, S. Meshoul, F. Khenfri, R. Fareh, and H. Shaiba, "Real-time implementation of quadrotor uav control system based on a deep reinforcement learning approach," *Computers, Materials and Continua*, 2024.

[17] Y. Cao, K. Ni, X. Jiang, T. Kuroiwa, H. Zhang, T. Kawaguchi, S. Hashimoto, and W. Jiang, "Path following for autonomous ground vehicle using ddpg algorithm: A reinforcement learning approach," *Applied Sciences*, 2023.

[18] P. Wenzel, T. Schön, L. Leal-Taixé, and D. Cremers, "Vision-based mobile robotics obstacle avoidance with deep reinforcement learning," *arXiv preprint arXiv:2103.04727*, 2021.

[19] G. Zhu, C. Pei, J. Ding, and J. Shi, "Deep deterministic policy gradient algorithm based lateral and longitudinal control for autonomous driving," *IEEE Transactions on Vehicular Technology*, pp. 740–745, 2020.

[20] G. Jin, Z. Li, B. Leng, R. Yu, X. Lu, and C. Sun, "Multi-timescale hierarchical reinforcement learning for unified behavior and control of autonomous driving," *arXiv preprint arXiv:2506.23771*, 2025.

[21] Y. S. Shao, C. Chen, S. Kousik, and R. Vasudevan, "Reachability-based trajectory safeguard (rts): A safe and fast reinforcement learning safety layer for continuous control," *arXiv preprint arXiv:2011.08421*, 2021.

[22] W. Zhao, T. He, and C. Liu, "Probabilistic safeguard for reinforcement learning using safety index guided gaussian process models," *arXiv preprint arXiv:2210.01041*, 2022.

[23] Y. Cheng, P. Zhao, and N. Hovakimyan, "Safe and efficient reinforcement learning using disturbance-observer-based control barrier functions," *arXiv preprint arXiv:2211.17250*, 2022.

[24] L. Zhao, K. Gatsis, and A. Papachristodoulou, "Stable and safe reinforcement learning via a barrier-lyapunov actor-critic approach," *arXiv preprint arXiv:2304.04066*, 2023.

[25] A. Al-Mahasneha, M. A. Mallouh, M. A. Al-Khawaldeh, B. Jouda, O. Shehata, and M. Baniyounis, "Online reinforcement learning control of robotic arm in presence of high variation in friction forces," *Systems Science & Control Engineering*, vol. 11, 2023.

[26] E. Ginzburg-Ganz, I. Segev, A. Balabanov, E. Segev, S. K. Naveh, R. Machlev, J. Belikov, L. Katzir, S. Keren, and Y. Levron, "Reinforcement learning model-based and model-free paradigms for optimal control problems in power systems: Comprehensive review and future directions," *Energies*, 2024.

[27] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[28] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[29] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pp. 1928–1937, 2016.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, T. E. Nicolas Heess, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2016.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, and A. A. R. et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[32] R. Dong, H. Zhang, and S. Liu, "An enhanced deep deterministic policy gradient algorithm for intelligent control of robotic arms," *Frontiers in Neurorobotics*, 2023.

[33] Z. Yuan, Z. Wang, X. Li, L. Li, and L. Zhang, "Hierarchical trajectory planning for narrow-space automated parking with deep reinforcement learning: A federated learning scheme," *Sensors*, vol. 23, no. 8, 2023.

[34] C. Canudas de Wit, H. Khennouf, C. Samson, and O. J. Sordalen, "Nonlinear control design for mobile robots," in *Recent Trends in Mobile Robots*, pp. 121–156, World Scientific, 1994.

[35] E. Nuño, A. Loría, Á. I. Paredes, and T. Hernández, "Consensus-based formation control of multiple nonholonomic vehicles under input constraints," *IEEE Control Systems Letters*, vol. 6, pp. 2767–2772, 2022.

[36] MathWorks, *Band-Limited White Noise*. The MathWorks, Inc., 2023. Accessed: 2025-09-15.

[37] T. Han and B. Wang, "Safety-critical stabilization of force-controlled nonholonomic mobile robots," *IEEE Control Systems Letters*, vol. 8, pp. 2469–2474, 2024.

[38] X. Xu, P. Tabuada, J. W. Grizzle, and A. D. Ames, "Robustness of control barrier functions for safety critical control," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 54–61, 2015. Analysis and Design of Hybrid Systems ADHS.

[39] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3387–3395, 2019.

[40] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2017.