Baseline-Assisted Adaptive Dynamic Programming Control for Mobile Robots and Surface Vessels

Thesis

Submitted in partial fulfillment of the requirements for the degree

Master of Engineering, Mechanical

At

The City College of the City University of New York

By

Yubai Liang

May 2026

Approved:

Prof. Bo Wang, Thesis Advisor

Prof. Feridun Delale, Chair Dept. of Mechanical Engineering

Baseline-Assisted Adaptive Dynamic Programming Control for Mobile Robots and Surface Vessels

Yubai Liang
Dept. of Mechanical Engineering
Thesis Advisor: Prof. Bo Wang

Abstract

This thesis presents a simulation-based study of baseline-assisted Adaptive Dynamic Programming (ADP) control for nonlinear vehicle systems, including a unicycle mobile robot, an underactuated surface vessel, and a preliminary multiple-vessel leader-follower system. The objective is to evaluate how an approximate value-function-based correction can be combined with conventional stabilizing controllers for parking and trajectory-tracking tasks.

First, a linearized ADP controller is developed for a local tracking-error model of the unicycle robot. The nonlinear body-frame tracking-error dynamics are linearized around a moving reference trajectory, and a quadratic value function is used to obtain a feedback gain for the transformed input. Simulation results show that this linearized controller can achieve local trajectory tracking, but its performance is limited for parking tasks and large initial errors because of the local nature of the approximation.

The main part of the thesis investigates a mixed baseline-ADP control structure for nonlinear vehicle systems. In this structure, a baseline controller provides the primary stabilizing action, while an ADP correction term is generated from an approximate value function. A weighting parameter λ is introduced to balance the baseline input and the ADP correction. The method is evaluated through simulations of unicycle parking and tracking, surface-vessel parking and tracking, and a preliminary leader-history-following task for multiple vessels. Random-initial-condition tests are also conducted to examine the influence of λ.

The results indicate that the contribution of the ADP correction is task-dependent. Parking tasks rely strongly on the baseline controller, whereas the surface-vessel tracking case can benefit from an intermediate value of λ. These findings suggest that, for the systems studied in this thesis, the ADP correction is more suitable as a supplementary adjustment to a stabilizing baseline controller than as a standalone controller. A preliminary DDPG implementation is included in the appendix as supplementary reinforcement-learning material.

Acknowledgments

I would like to express my sincere gratitude to my thesis advisor, Prof. Bo Wang, for his guidance, support, and valuable feedback throughout this project. The encouragement helped me build confidence and continue moving forward during the research process.

I would also like to thank Mon, my friends, and everyone who has helped me for their companionship, support, and encouragement throughout this journey.

Finally, I am grateful for the patience that allowed me to complete this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Mobile robots and nonlinear vehicle systems are central to many automation tasks in which motion must be planned and controlled under kinematic constraints, dynamic coupling, and environmental disturbances [1]. Examples include autonomous parking in confined urban spaces, automated transport vehicles in warehouses, and unmanned surface vessels performing docking or navigation in busy ports [2, 3]. In these applications, the controller must not only drive the vehicle toward a desired objective, but also maintain accuracy, robustness, and safety when the operating environment is uncertain.

Two important control problems are considered in these applications: point stabilization and trajectory tracking [4, 5]. Point stabilization requires the vehicle to reach and remain at a prescribed target pose. A typical example is assisted parking, where the vehicle must arrive at a target position and stop with a specified orientation [2]. Trajectory tracking instead requires the vehicle to follow a reference path while keeping the tracking error small [6]. This problem appears in autonomous driving, path tracking, and navigation tasks where both spatial accuracy and timing are relevant. Although these problems are closely related, they are not exactly the same, and each one may require a different controller design [4, 7].

For wheeled mobile robots, the unicycle-type kinematic model is a common starting point [4]. The robot configuration is described by its planar position and heading angle,

$$z = [x,\, y,\, \theta]^T,$$

where x and y denote the position of the robot in the inertial frame, and θ denotes its

heading angle. The control inputs are the translational velocity v and the angular velocity ω. The corresponding kinematic equations are

$$\dot{x} = v\cos\theta, \qquad \dot{y} = v\sin\theta, \qquad \dot{\theta} = \omega. \tag{1.1}$$

Although the model is simple, it captures a key feature of wheeled mobile robots: the nonholonomic motion constraint [4]. Since the robot cannot generate lateral motion directly, position errors cannot be corrected by simply moving sideways. The robot must instead coordinate its heading angle, forward velocity, and angular velocity to approach the desired pose or path.

Surface vessels introduce a more complex nonlinear control problem. Unlike the unicycle model, a surface vessel is usually described by both kinematic and dynamic equations. The state may include the inertial-frame position, heading angle, surge velocity, sway velocity, and yaw rate, while the available control inputs are typically the surge force and yaw moment. Because there is no direct control input in the sway direction, the vessel is underactuated. As a result, lateral motion must be produced indirectly through the coupling among heading, surge motion, sway dynamics, and yaw motion [8, 9].

Classical control methods, including PID control, LQR control, feedback linearization, and nonlinear feedback control, provide clear mathematical structures and can perform well when the model is accurate and the operating conditions are known [10]. However, nonlinear vehicle systems often operate under uncertainty [5]. Wheeled mobile robots may experience wheel slip, actuator saturation, or parameter mismatch, while surface vessels may be affected by water resistance, ocean currents, wind, and unmodeled hydrodynamic forces [8]. In these cases, a controller designed for an ideal nominal model may require repeated tuning or may fail to maintain the desired performance across different operating conditions.

Adaptive Dynamic Programming (ADP) provides a useful framework [11]. ADP connects optimal control with reinforcement learning by seeking a control policy that minimizes a long-term cost function while avoiding the need to solve the Hamilton–Jacobi–Bellman equation exactly [12, 13]. For nonlinear systems, an exact analytical solution of this equation is generally difficult or unavailable. ADP addresses this difficulty by approximating the value function and updating the control policy through an iterative learning process [14]. In the linear quadratic setting, this process can be interpreted through policy evaluation and policy improvement [15]. Policy evaluation estimates the cost associated with the current policy [16], while policy improvement updates the controller using the estimated value function [12, 13].

This thesis uses ADP as the main control framework for nonlinear vehicle-control simulations. A linear second-order ADP Simulink model is first considered as a reference implementation to provide a simpler setting for understanding the basic ADP structure [12]. The linear and nonlinear ADP implementations are not mathematically identical. The linear second-order model is used mainly to clarify the ADP mechanism in a system where the dynamics can be written in linear form, and the value function is associated with a quadratic cost [6, 12]. The main focus of the thesis is the nonlinear ADP simulation study for unicycle robots, single surface vessels, and multiple-vessel systems.

The unicycle model is used as the first nonlinear test case because it has a relatively simple structure while still containing the essential nonholonomic constraint [4]. The surface-vessel model is then studied as a more challenging system due to its inertia, damping, underactuation, and dynamic coupling [17]. The vessel simulation is further extended to a multiple-vessel case, where a leader–follower structure is used to examine coordinated motion [18]. Across these nonlinear simulations, different values of the weighting parameter λ are tested to study the balance between the baseline controller and the ADP correction term.

A Deep Deterministic Policy Gradient (DDPG) reinforcement-learning controller was also implemented in Simulink during the early stage of this project [19]. The DDPG simulations included both kinematic and dynamic vehicle models and provided useful experience in reinforcement-learning-based control [20], particularly in reward design, observation, and action selection, and Simulink-based training [10, 21]. However, the final research direction shifted from general deep reinforcement learning to ADP-based control. For this reason, the DDPG results are not treated as the main contribution of the thesis and are included only as supplementary baseline results in the appendix. The main contribution of this thesis remains the ADP-based nonlinear control study for unicycle, single-vessel, and multiple-vessel systems.

1.2 Motivation

Nonlinear vehicle control is challenging due to motion constraints, dynamic coupling, and limited actuation. These issues appear in both mobile robots and surface vessels. A unicycle mobile robot cannot generate lateral motion directly under its nonholonomic constraint [4]. An underactuated surface vessel has a similar limitation, since it does not have a direct control input in the sway direction [17]. As a result, position and heading errors must be reduced through the available inputs and the natural coupling in the system dynamics [9].

Classical model-based control methods are useful due to their clear mathematical structure and interpretable control laws [5]. However, these methods usually depend on an accurate model and carefully selected control gains. In practical applications, the nominal model may not fully describe the real system. For example, surface vessels may be affected by external disturbances, parameter uncertainty, and unmodeled hydrodynamic effects [8]. These effects may cause tracking error, oscillation, or slow convergence if the controller is not designed with sufficient robustness.

Reinforcement learning provides another possible approach for nonlinear control problems [20]. In the early stage of this project, a DDPG controller was implemented in Simulink for kinematic and dynamic vehicle models [10]. This work helped develop experience with observation design, action design, reward function design, and simulation-based training [21]. However, DDPG also has practical limitations. The training process may require many episodes, the reward function requires careful tuning, and the learned policy is not always easy to interpret from a control-theoretic point of view [22]. Therefore, the main direction of this thesis was shifted toward Adaptive Dynamic Programming.

Adaptive Dynamic Programming is selected as the main framework because it has a direct connection with optimal control [11]. ADP uses a value function to evaluate the long-term cost of a control policy, and the controller is improved based on this value function [12, 13]. The linear quadratic formulation provides an important reference, since it connects ADP with the Riccati equation and classical LQR control [12]. However, the nonlinear vehicle systems studied in this thesis cannot be fully represented by constant system matrices. Therefore, this thesis first studies a linearized ADP controller for the unicycle tracking-error model and then extends the control design to nonlinear vehicle systems.

Another motivation of this thesis is to examine the balance between a baseline controller and an ADP correction term. For nonlinear systems, a learned correction term alone may not always produce stable or smooth behavior, especially when the system is underactuated or strongly coupled. A baseline controller can provide the main stabilizing behavior, while the ADP correction term can adjust the input based on the approximate value function. For this purpose, this thesis uses a mixed control law with a weighting parameter λ. Different values of λ are tested to study how the controller behavior changes when the baseline controller or the ADP correction term has more influence.

1.3 Problem Statement

This thesis investigates the design and simulation-based evaluation of baseline-assisted ADP control strategies for nonlinear vehicle systems. The systems considered in this study include:

1) a unicycle-type mobile robot,

2) a single underactuated surface vessel,

3) a multiple-vessel leader-follower system.

The control objective is to reduce tracking or stabilization errors while maintaining the control inputs within reasonable bounds. For the unicycle robot, the controller must account for the nonholonomic motion constraint. For the surface vessel, the controller must further address underactuated dynamics and the coupling among surge, sway, and yaw motions [9, 17].

The thesis considers two ADP-based control levels. First, a linearized ADP controller is designed for a local linearized tracking-error model of the unicycle robot. This part connects the linear ADP theory with a nonlinear vehicle model through local linearization. Second, a nonlinear ADP-based control structure is used for nonlinear vehicle systems. This structure combines a baseline controller with an ADP correction term:

$$u = \lambda u_{\text{Baseline}} + (1 - \lambda) u_{\text{ADP}}, \tag{1.2}$$

where u_{Baseline} is the baseline control input, u_{ADP} is the ADP correction input, and $\lambda \in [0, 1]$ is a weighting parameter.

When $\lambda = 1$, the control law reduces to the baseline controller. When $\lambda = 0$, the control law relies only on the ADP correction term. Intermediate values of λ provide a tradeoff between the baseline control behavior and the ADP correction. One important goal of this thesis is to compare different values of λ and analyze their influence on the nonlinear control performance.

1.4 Research Objectives

The main objective of this thesis is to develop and evaluate an Adaptive Dynamic Programming-based control framework for nonlinear vehicle systems [11, 12]. The study starts from a

linearized ADP design for a unicycle tracking-error model and then extends the method to nonlinear systems, including a unicycle-type mobile robot, a single underactuated surface vessel, and a multiple-vessel leader-follower system.

The first objective is to connect the linear ADP theory with a physically meaningful vehicle model. Instead of applying linear ADP to an arbitrary linear system, the unicycle tracking-error dynamics are linearized around the reference trajectory. The resulting local linear model is then used to design and evaluate a linearized ADP controller.

The second objective is to apply a nonlinear ADP-based control structure to vehicle systems with nonlinear motion, nonholonomic constraints, underactuation, and dynamic coupling. In this structure, a baseline controller provides the main stabilizing behavior, while the ADP correction term modifies the input based on an approximate value function.

The third objective is to study the mixed control law that combines the baseline controller and the ADP correction term. Different values of the weighting parameter λ are tested to analyze how the balance between the two terms affects tracking error, convergence, control smoothness, and input response.

1.5 Contributions

The main contribution of this thesis is the implementation and simulation-based evaluation of a mixed baseline-ADP control structure for nonlinear vehicle systems.

First, a linearized ADP controller is implemented for a unicycle tracking-error model. Instead of applying linear ADP to an arbitrary linear system, the nonlinear tracking-error dynamics are locally linearized around the reference trajectory. The resulting linearized model provides a physical connection between the linear ADP formulation and the nonlinear unicycle control problem.

Second, a mixed baseline-ADP control structure is implemented for nonlinear unicycle and surface vessel models. In this structure, the baseline controller provides the main stabilizing behavior, while the ADP correction term is generated from a quadratic approximate value function. The unicycle model is used to study parking and trajectory tracking under the nonholonomic constraint [4]. The surface vessel model is used as a more complex nonlinear system with inertia, damping, underactuation, and coupling among surge, sway, and yaw motion [8, 9, 17].

Third, the influence of the weighting parameter λ is evaluated through fixed-initial-

condition simulations and random-initial-condition tests. Different values of λ are tested to study the balance between the baseline controller and the ADP correction term. The results show that the effect of λ is task-dependent. Parking tasks rely strongly on the baseline controller, while the vessel tracking task can benefit from an intermediate value of λ.

Fourth, a preliminary leader-history following simulation is developed for a multiple-vessel system [18]. In this case, the follower vessels use delayed states from the leader trajectory history as their references. This part extends the single-vessel tracking setup to a simple coordinated cruising scenario, but it is treated as a preliminary extension rather than a complete formation control design.

Finally, the earlier DDPG implementation is reorganized as supplementary baseline material in the appendix. This keeps the main thesis focused on ADP-based control while still preserving the reinforcement-learning results developed during the early stage of the project.

Chapter 2

Literature Review and Theoretical Background

2.1 Nonlinear Vehicle Control Problems

Nonlinear vehicle systems often include kinematic constraints, nonlinear coupling, and limited actuation. These properties make the control problem different from a standard fully actuated linear system. In this thesis, the vehicle systems are the unicycle robot and the underactuated surface vessel. The unicycle robot is used as a relatively simple nonlinear mobile robot model, while the surface vessel represents a more complex dynamic system with inertia, damping, coupling, and underactuation.

2.1.1 Point Stabilization and Trajectory Tracking

Two common control objectives for vehicle systems are point stabilization and trajectory tracking [4]. Point stabilization requires the vehicle to reach a fixed target pose and remain close to it. In this thesis, this objective is related to parking-type behavior, where the vehicle moves toward a desired position and orientation. Trajectory tracking requires the vehicle to follow a time-varying reference trajectory. The desired trajectory is usually written as

$$x_d(t), \quad y_d(t), \quad \theta_d(t),$$

where $x_d(t)$ and $y_d(t)$ define the desired position, and $\theta_d(t)$ defines the desired heading angle. The controller should reduce the tracking error while the reference changes with time.

For nonlinear and underactuated vehicle systems, point stabilization and trajectory tracking are not identical problems. In a tracking problem, the reference may have a nonzero velocity, and the vehicle can follow the moving reference locally. In a stabilization or parking problem, the final reference velocity may become zero [5].

2.1.2 Nonholonomic Mobile Robot Control

A wheeled mobile robot is commonly represented by the unicycle-type kinematic model. The state consists of the planar position and heading angle, and the control inputs are the translational velocity and angular velocity. The model contains the nonlinear terms $\cos\theta$ and $\sin\theta$, which show that the robot's motion depends on its heading angle.

2.1.3 Underactuated Surface Vessel Control

Surface vessel control is more complex than unicycle robot control because the vessel model includes both kinematics and dynamics. The motion is usually described by the earth-fixed position and heading, together with body-fixed velocities such as surge velocity, sway velocity, and yaw rate. The control inputs are typically surge force and yaw moment rather than direct velocity commands.

A common difficulty in surface vessel control is underactuation. In many vessel models, there is no direct control input in the sway direction [8]. Therefore, lateral motion must be generated indirectly through the coupling among surge, sway, and yaw motion [9, 17]. In addition, inertia and damping make the vessel's response slower and more complex than the response of a kinematic unicycle robot.

These properties make the surface vessel a useful test case for nonlinear ADP-based control. The controller must handle nonlinear coupling and underactuation while keeping the tracking or parking error within a reasonable range.

2.2 Optimal Control and the HJB Equation

Optimal control provides a mathematical framework for designing a control policy that minimizes a long-term cost function. For a nonlinear control-affine system,

$$\dot{x} = f(x) + g(x)u,$$

where x is the system state and u is the control input, an infinite-horizon cost function can be written as

$$J(x_0; u) = \int_0^\infty r(x(t), u(t))dt,$$

where x_0 is the initial state and $r(x, u)$ is the instantaneous cost.

A commonly used cost structure is

$$r(x, u) = q(x) + u^T Ru,$$

where $q(x)$ penalizes the state error and $u^T Ru$ penalizes the control effort. The value function describes the accumulated cost starting from a given state. Under the optimal policy, the optimal value function is defined as

$$V^*(x) = \min_u \int_0^\infty r(x(t), u(t))dt.$$

For continuous-time nonlinear systems, the optimal value function satisfies the Hamilton–Jacobi–Bellman equation:

$$0 = \min_u \left[r(x, u) + \nabla V^T(x)(f(x) + g(x)u) \right].$$

When the cost is quadratic in the control input, the corresponding optimal control law can be written as

$$u^*(x) = -\frac{1}{2} R^{-1} g^T(x) \nabla V^*(x).$$

This expression shows that the optimal control input depends on the gradient of the value function. However, solving the HJB equation exactly is difficult for most nonlinear systems. This difficulty motivates approximate methods such as Adaptive Dynamic Programming, where the value function or its gradient is approximated and then used to improve the control policy [11–13].

2.3 Adaptive Dynamic Programming

Adaptive Dynamic Programming is an approximate optimal control method based on value functions and policy improvement [11, 12]. It is useful when the optimal value function is difficult to obtain analytically. Instead of solving the Hamilton–Jacobi–Bellman equation directly, ADP estimates the value function from system information or data and then updates

the control policy based on the estimated value function.

The ADP procedure can be described by two main steps: policy evaluation and policy improvement [15]. In policy evaluation, the value function associated with the current control policy is estimated. In policy improvement, the control policy is updated using the evaluated value function. These two steps are repeated until the policy or the value function converges.

2.3.1 Policy Evaluation and Policy Improvement

For a given control policy, the value function represents the long-term cost produced by that policy. Policy evaluation estimates this cost. After the value function is obtained, policy improvement uses it to generate a new control policy with a lower cost.

In continuous-time optimal control, this idea is closely related to the Hamilton–Jacobi–Bellman equation. The value function provides information about how the cost changes with respect to the state. Therefore, the gradient of the value function can be used to update the control input. This is the main reason why the value function plays a central role in ADP-based control.

In this thesis, the ADP idea is used in two different ways. Chapter 3 uses a linear ADP formulation for a local linearized unicycle error model. Chapter 4 uses a nonlinear ADP-based correction term together with a baseline controller for nonlinear vehicle systems.

2.3.2 Linear ADP and LQR Connection

For a continuous-time linear system,

$$\dot{x} = Ax + Bu,$$

with a quadratic cost function

$$J(x_0; u) = \int_0^\infty \left(x^T Q x + u^T R u\right) dt,$$

the value function can be written in the quadratic form

$$V(x) = x^T P x,$$

where $P = P^T > 0$ is the value matrix.

If the control policy is

$$u = -Kx,$$

then the closed-loop system is

$$\dot{x} = (A - BK)x.$$

For a given feedback gain K_k, the policy evaluation step estimates the corresponding value matrix P_k. In the linear quadratic case, P_k satisfies

$$(A - BK_k)^T P_k + P_k(A - BK_k) + Q + K_k^T R K_k = 0.$$

The policy improvement step then updates the feedback gain as

$$K_{k+1} = R^{-1} B^T P_k.$$

This structure is closely related to the Linear Quadratic Regulator. The main difference is that ADP can estimate the value-related matrix using data, while the classical LQR solution is obtained from a model-based Riccati equation. In Chapter 3, this linear ADP structure is applied to a local linearized tracking-error model of the unicycle robot.

2.3.3 Nonlinear ADP and Value Function Approximation

For nonlinear systems, the dynamics are usually written as

$$\dot{x} = f(x) + g(x)u.$$

Unlike the linear case, the system generally cannot be represented by constant matrices A and B, and the exact optimal value function is difficult to obtain. Therefore, nonlinear ADP often uses an approximate value function.

In nonlinear ADP, the value function is usually approximated by a selected parameterized structure. In this thesis, the nonlinear simulations use a feature-based quadratic approximation,

$$V(e) = \phi^T(e) P \phi(e),$$

where e is the tracking error, $\phi(e)$ is a selected feature vector, and P is the approximate value matrix. The feature vector may include both error states and nonlinear terms, such as $\cos(e_\theta)$ and $\sin(e_\theta)$, to better represent heading-dependent behavior.

The gradient of the approximate value function is

$$\nabla V(e) = 2J^T(e)P\phi(e),$$

where

$$J(e) = \frac{\partial \phi(e)}{\partial e}.$$

Using the value-function-based control structure, the ADP correction term can be written as

$$u_{\text{ADP}} = -\frac{1}{2}R^{-1}G^T\nabla V(e),$$

where G is the input distribution matrix associated with the selected error dynamics or input channel.

For nonlinear vehicle systems, the ADP correction term alone may not always provide reliable behavior, especially when the initial error is large or the system is underactuated. Therefore, this thesis combines the ADP correction with a baseline controller:

$$u = \lambda u_{\text{Baseline}} + (1-\lambda)u_{\text{ADP}},$$

where $\lambda \in [0, 1]$ controls the balance between the baseline controller and the ADP correction term. This mixed structure is used in Chapter 4 for the nonlinear vehicle-control simulations.

2.4 Reinforcement Learning and DDPG Baseline

Reinforcement learning is another learning-based framework for control problems. In reinforcement learning, an agent interacts with an environment by observing the system state, selecting an action, and receiving a reward [20, 23]. The objective is to learn a policy that gives good long-term performance through repeated interaction with the environment.

Deep Deterministic Policy Gradient (DDPG) is an actor-critic reinforcement learning method for continuous action spaces [19]. The actor network generates continuous control actions, while the critic network estimates the action-value function. This structure makes DDPG suitable for control problems with continuous inputs, such as velocity commands, force inputs, and moment inputs.

In the early stage of this project, a DDPG controller was implemented in Simulink for kinematic and dynamic vehicle models. This work helped develop experience with observation design, action design, reward function design, and simulation-based training. However,

DDPG also has practical limitations. The training process may require many episodes, the reward function needs careful tuning, and the learned policy is not always easy to interpret from a control-theoretic perspective [22].

For this reason, DDPG is not treated as the main method in this thesis. It is included only as supplementary baseline material in the appendix. The main focus of this thesis is the ADP-based control framework, which has a clearer connection with value functions, policy improvement, and optimal control theory.

2.5 Summary

This chapter reviewed the main theoretical background used in this thesis. Nonlinear vehicle control problems were first introduced through point stabilization, trajectory tracking, nonholonomic mobile robot control, and underactuated surface vessel control. These topics provide the modeling background for the unicycle robot and surface vessel systems studied in the later chapters.

The chapter then reviewed optimal control and the Hamilton–Jacobi–Bellman equation. This discussion showed that the optimal control law depends on the gradient of the value function, but solving the HJB equation exactly is difficult for nonlinear systems. Adaptive Dynamic Programming was introduced as an approximate optimal control framework based on value function approximation, policy evaluation, and policy improvement.

The linear ADP formulation provides the theoretical basis for the linearized unicycle controller in Chapter 3. The nonlinear ADP formulation and the mixed baseline-ADP control law provide the basis for the nonlinear vehicle control simulations in Chapter 4. DDPG was also reviewed briefly as a reinforcement learning baseline, but it is treated as supplementary material rather than the main control method.

Chapter 3

Linearized ADP Control for Local Unicycle Tracking

3.1 Chapter Overview

This chapter introduces the linear ADP method as the first step of the control design. The linear case has a simpler structure than the nonlinear case and provides a clear setting for explaining value function approximation, policy evaluation, and policy improvement. For a linear system, the value function can be written in the quadratic form $V(x) = x^T P x$, and the feedback gain can be obtained from the learned value matrix P. This makes the linear case useful for showing how ADP updates the value matrix and improves the control policy.

In the linear ADP formulation, the system is written in the standard form

$$\dot{x} = Ax + Bu.$$

The linear model used in this chapter is obtained from a local linear approximation of the unicycle tracking-error dynamics. The original unicycle model is nonlinear, but its tracking-error dynamics can be approximated by a linear model near the desired reference trajectory. In this way, the linear ADP method is applied to a physically meaningful system while the controller design remains within the linear framework.

The tracking task is first rewritten as an error stabilization problem. The original objective is to make the robot states $x(t)$, $y(t)$, and $\theta(t)$ follow the reference trajectory $x_r(t)$, $y_r(t)$, and $\theta_r(t)$. This objective can be expressed by defining a tracking error vector e. When the tracking error approaches zero, the robot follows the desired reference trajectory.

After the tracking error is defined, the system dynamics are expressed in terms of the error variables. The tracking-error dynamics are then linearized around the desired operating point, which corresponds to zero tracking error and zero transformed input. Under this local approximation, small-angle and first-order assumptions are used. Higher-order products, such as $u_2 e_1$ and $u_2 e_2$, are neglected near the equilibrium point. The resulting local linear model is

$$\dot{e} = Ae + B\tilde{u},$$

where e is the tracking error and $\tilde{u}$ is the transformed control input.

The controller designed in this chapter is a linear ADP controller for this local linearized error system. The value function is approximated as $V(e) = e^T P e$, and the feedback gain is obtained from the learned matrix P. The connection to the nonlinear unicycle model gives the linear system a physical meaning, while the main focus of this chapter remains the linear ADP design.

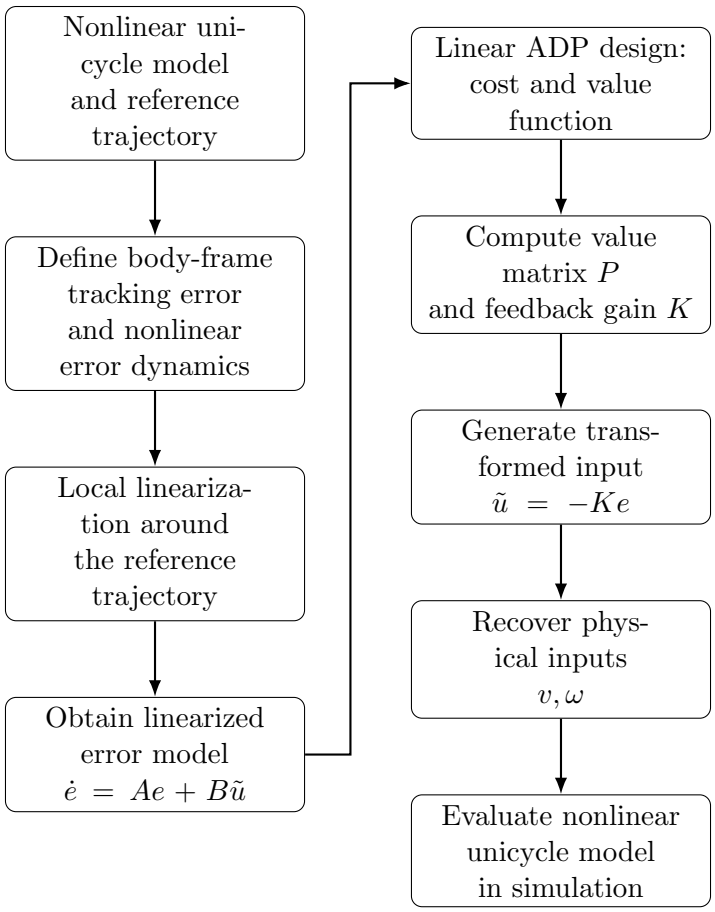

Figure 3.1: Flowchart of the linearized ADP control design for the unicycle tracking-error system.

3.2 Unicycle Tracking-Error Linearization

The unicycle robot is described by the kinematic model

$$\dot{x} = v\cos\theta,$$

$$\dot{y} = v\sin\theta,$$

$$\dot{\theta} = \omega,$$

where x and y are the position of the robot in the global frame, θ is the heading angle, v is the linear velocity, and ω is the angular velocity. The desired reference trajectory is denoted by $x_r(t)$, $y_r(t)$, and $\theta_r(t)$, and is generated by the reference inputs $v_r(t)$ and $\omega_r(t)$.

Since the unicycle model contains the nonlinear terms $\cos\theta$ and $\sin\theta$, it cannot be directly written in the standard linear form $\dot{x} = Ax + Bu$. Therefore, the tracking problem is first reformulated as an error stabilization problem. The tracking error is expressed in the robot body frame as

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix}.$$

Here, e_1 is the longitudinal position error, e_2 is the lateral position error, and e_3 is the heading error. If $e \to 0$, then the robot follows the desired reference trajectory.

By differentiating the body-frame error definition and substituting the unicycle kinematics, the position-error dynamics and heading-error dynamics can be written as

$$\dot{e}_1 = \omega e_2 + v_r\cos(e_3) - v,$$

$$\dot{e}_2 = -\omega e_1 + v_r\sin(e_3),$$

$$\dot{e}_3 = \omega_r - \omega.$$

To write the error system in a form suitable for linearization, the transformed inputs are defined as

$$u_1 = -v + v_r\cos(e_3),$$

$$u_2 = \omega_r - \omega.$$

Therefore, the actual robot inputs can be recovered from

$$v = v_r \cos(e_3) - u_1,$$

$$\omega = \omega_r - u_2.$$

Using these transformed inputs, the nonlinear tracking-error dynamics become

$$\dot{e}_1 = (\omega_r - u_2)e_2 + u_1,$$

$$\dot{e}_2 = -(\omega_r - u_2)e_1 + v_r \sin(e_3),$$

$$\dot{e}_3 = u_2.$$

The system is then linearized around the zero-error condition and the nominal transformed input. Near this operating point, the small-angle approximation

$$\sin(e_3) \approx e_3$$

is used. The products $u_2 e_1$ and $u_2 e_2$ are neglected because they are second-order small terms near the equilibrium point. Therefore, the nonlinear tracking-error dynamics are approximated by

$$\dot{e}_1 \approx \omega_r e_2 + u_1,$$

$$\dot{e}_2 \approx -\omega_r e_1 + v_r e_3,$$

$$\dot{e}_3 = u_2.$$

Thus, the local linearized tracking-error model is obtained as

$$\dot{e} = Ae + B\tilde{u},$$

where

$$A = \begin{bmatrix} 0 & \omega_r & 0 \\ -\omega_r & 0 & v_r \\ 0 & 0 & 0 \end{bmatrix},$$

and

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

The matrices A and B are not selected arbitrarily. They are derived from the local linearization of the nonlinear unicycle tracking-error dynamics. The linearized model gives the linear ADP controller a direct connection to the original nonlinear robot model.

3.3 Linear ADP Design for the Linearized Error System

After linearization, the tracking-error system is written as

$$\dot{e} = Ae + B\tilde{u},$$

where $e \in \mathbb{R}^3$ is the tracking error and $\tilde{u} = [u_1, u_2]^T$ is the transformed control input. Since this system has the same structure as a continuous-time linear system, the linear ADP method can be used to learn a feedback control law for the error dynamics.

The control objective is to reduce the tracking error while limiting the control effort. The infinite-horizon quadratic cost function is defined as

$$J(e_0; \tilde{u}) = \int_0^\infty \left(e^T Q e + \tilde{u}^T R \tilde{u}\right) dt,$$

where Q penalizes the tracking error and R penalizes the transformed control input. Larger weights in Q place more emphasis on error reduction, while larger weights in R lead to smaller control inputs.

For the linearized error system, the value function is approximated by the quadratic form

$$V(e) = e^T P e,$$

where $P = P^T > 0$ is the value matrix. This matrix represents the cost associated with the current feedback policy. Once P is known, the feedback gain can be computed as

$$K = R^{-1} B^T P.$$

The transformed control input is then given by

$$\tilde{u} = -Ke.$$

In the ADP implementation, P is estimated from measured state and input data. During the learning stage, an initial stabilizing feedback gain is used to generate the control input. A small exploration signal is also added so that the collected data contain sufficient information for estimating the value matrix. The collected data are used in the integral Bellman equation to solve for the unknown entries of P. After P_{ADP} is obtained, the corresponding feedback gain is calculated as

$$K_{\mathrm{ADP}} = R^{-1}B^T P_{\mathrm{ADP}}.$$

After the learning stage, the linear ADP controller is evaluated on the nonlinear tracking-error dynamics. At each simulation step, the controller first computes the transformed input

$$\tilde{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = -K_{\mathrm{ADP}} e.$$

The actual robot commands are then recovered from the inverse input transformation:

$$v = v_r \cos(e_3) - u_1,$$

$$\omega = \omega_r - u_2.$$

Thus, the controller is designed using the local linearized model, but its performance is tested on the nonlinear unicycle error dynamics. This allows the linear ADP design to be evaluated as a local control method for the nonlinear tracking problem.

3.4 Simulation Studies

The linearized ADP controller is evaluated in two scenarios: trajectory tracking and parking. The tracking case uses a circular reference trajectory generated by the reference linear velocity and angular velocity

$$v_r = 0.5, \quad \omega_r = 0.2.$$

The corresponding reference radius is

$$R_c = \frac{v_r}{\omega_r} = 2.5 \text{ m}.$$

The weighting matrices are selected as

$$Q = \mathrm{diag}(8, 8, 3),$$

and

$$R = \mathrm{diag}(0.5, 0.5).$$

The matrix Q determines the relative importance of the tracking errors. In this work, larger weights are assigned to the position errors e_1 and e_2, while the heading error e_3 is also penalized with a smaller weight. This choice makes the controller focus mainly on reducing the position tracking error while still regulating the heading error. The matrix R penalizes the transformed control inputs u_1 and u_2. Since the two diagonal entries of R are equal, the two transformed inputs are weighted equally in the cost function. The selected values of Q and R were chosen empirically to obtain a reasonable balance between tracking performance and control effort. Different choices of Q and R would change the closed-loop behavior. Increasing the entries of Q generally makes the controller reduce the tracking error more aggressively, but it may also require larger control inputs. Increasing the entries of R reduces the control effort and usually gives smoother inputs, but it may slow down the convergence of the tracking error.

The simulation step size is

$$\Delta t = 0.01 \text{ s}.$$

The learning time is set to 35 s, and the nonlinear evaluation time is set to 40 s. The initial robot state is

$$x_0 = \begin{bmatrix} 2.0 \\ -1.5 \\ 1.2 \end{bmatrix}.$$

During the learning stage, a small exploration signal is added to the feedback input. This signal helps the system generate sufficiently rich state and input data for estimating the value matrix P. The control input is also saturated so that the generated commands remain within a reasonable range. For the parking case, the desired reference is a fixed target pose. In this case, the reference velocities are zero. However, directly linearizing the unicycle model at zero reference velocity may lead to a weakly controllable local linear model. Therefore, the ADP learning stage uses a small moving nominal condition to obtain a useful feedback gain. The learned gain is then evaluated on the nonlinear parking task with a fixed target pose.

3.4.1 Circular Tracking Evaluation

Figure 3.2 shows the trajectory tracking result using the linearized ADP controller. The first subplot shows the robot path and the circular reference path. The remaining subplots show

the x-trajectory, y-trajectory, x-error, y-error, and angle error. The robot follows the circular reference trajectory generated by $v_r = 0.5$ and $\omega_r = 0.2$. The tracking errors decrease quickly and remain bounded during the simulation, which indicates that the learned feedback gain stabilizes the local tracking-error dynamics.

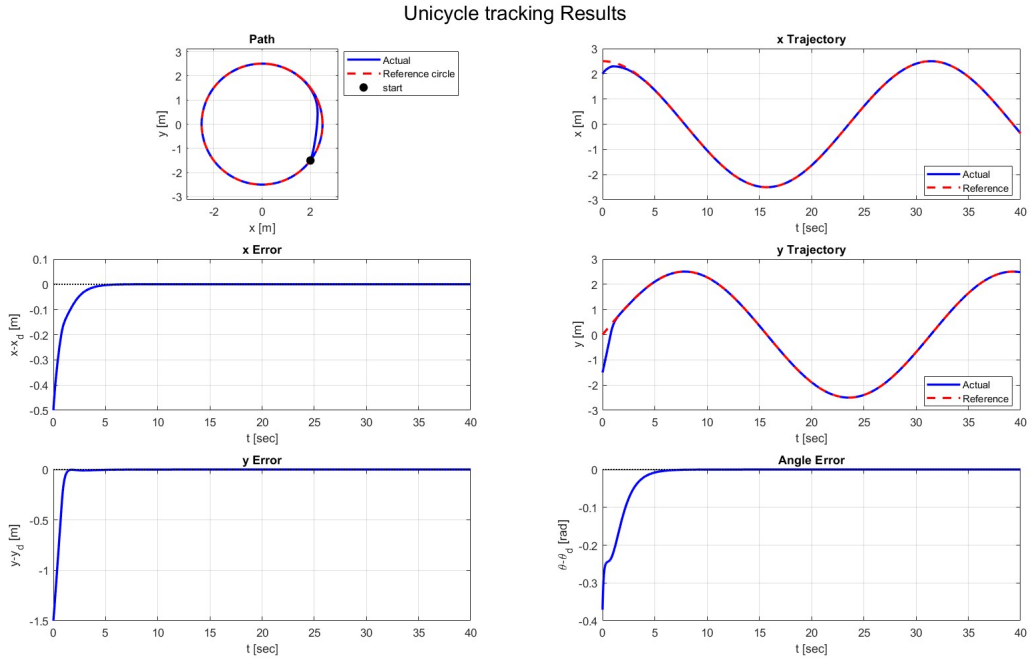

Figure 3.2: Tracking result using the linearized ADP controller.

The result shows that the linearized ADP controller can guide the robot along the desired trajectory when the tracking error remains in the region where the local linear approximation is valid. Since the controller is designed for the transformed input $\tilde{u}$, the actual robot inputs v and ω are recovered through the inverse input transformation during simulation.

3.4.2 Parking Test and Limitation

The parking task is also tested with the linearized ADP controller. In this task, the reference is a fixed target pose, which is different from the tracking case with a moving reference trajectory. Since parking involves larger changes in position and heading, it is more sensitive to the local linearization assumption.

Figure 3.3 shows the parking response using the linearized ADP controller. The robot does not fully converge to the fixed target pose. Although the trajectory changes during the initial transient, the position errors remain nonzero after the transient response. This

indicates that the local linearized controller is not sufficient for the parking task when the initial error is relatively large.

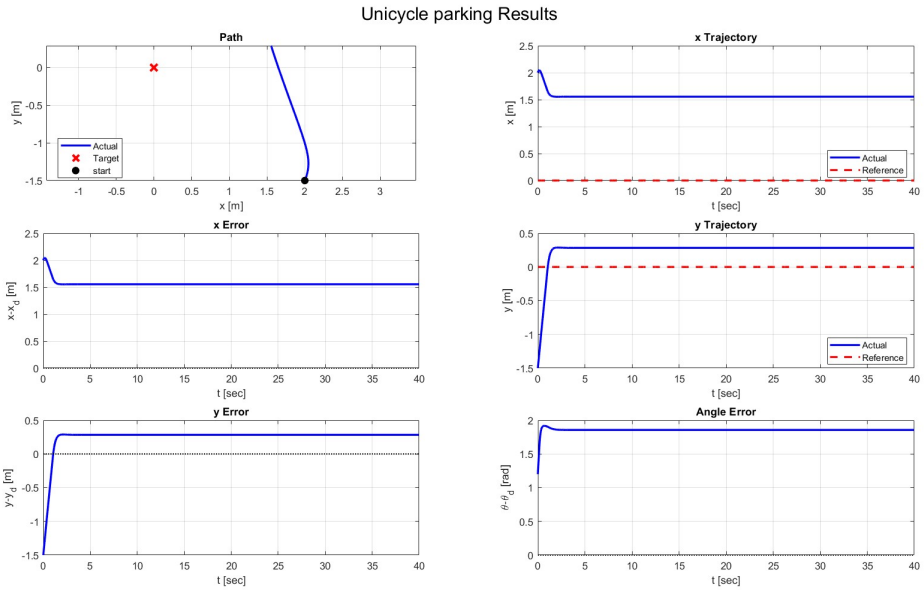

Figure 3.3: Parking response using the linearized ADP controller.

The limited parking performance is expected because the controller is designed from a local linearized model. The unicycle robot cannot directly move in the lateral direction, and the parking task depends strongly on nonlinear motion, such as turning and forward movement. The local linearized model does not fully capture these nonlinear effects.

3.5 Discussion

This chapter applied linear ADP to a linearized model of the nonlinear unicycle robot. Unlike an arbitrary linear system, the model used in this chapter has a clear physical meaning because the matrices A and B are derived from the nonlinear tracking-error dynamics. This provides a direct connection between the linear ADP theory and the nonlinear vehicle control problem.

The simulation results show that the learned ADP controller can stabilize the linearized error system in the tracking case. However, the linearized ADP method has clear limitations. The controller is designed from a local approximation around the reference trajectory, so its performance depends on the assumption that the tracking error remains small. When the robot starts far from the reference, has a large heading error, or performs a parking task,

the neglected nonlinear terms can have a stronger effect on the closed-loop response.

Another limitation is that the ADP controller is designed for the transformed input $\tilde{u}$, while the actual robot commands are v and ω. Although the input transformation makes it possible to apply the linear ADP method, the controller still depends on the local linearized error model. Therefore, it cannot fully represent the nonlinear motion required in tasks such as parking.

Chapter 4

Baseline-Assisted ADP Correction for Nonlinear Vehicle Systems

4.1 Chapter Overview

The previous chapter applied linear ADP to a local linearized tracking-error model of the unicycle robot. The results showed that the linearized method is useful for local tracking, but its performance is limited when the robot performs parking or starts with a relatively large initial error. This limitation mainly comes from the local approximation used in the linearized model.

This chapter extends the control design to nonlinear vehicle systems, including the unicycle robot, the underactuated surface vessel, and the multiple-vessel leader-follower system. The same nonlinear ADP-based control framework is applied to these systems, but the model, input structure, and control objective are different in each case. The general nonlinear ADP control structure includes the nonlinear system formulation, the approximate value function, the ADP correction term, and the mixed baseline-ADP control law. The method is first applied to the nonlinear unicycle robot, which allows comparison with the linearized ADP method in Chapter 3. The surface vessel and multiple-vessel cases are then considered as extensions to more complex vehicle systems.

Figure 4.1 summarizes the nonlinear ADP-based control structure used in this chapter.

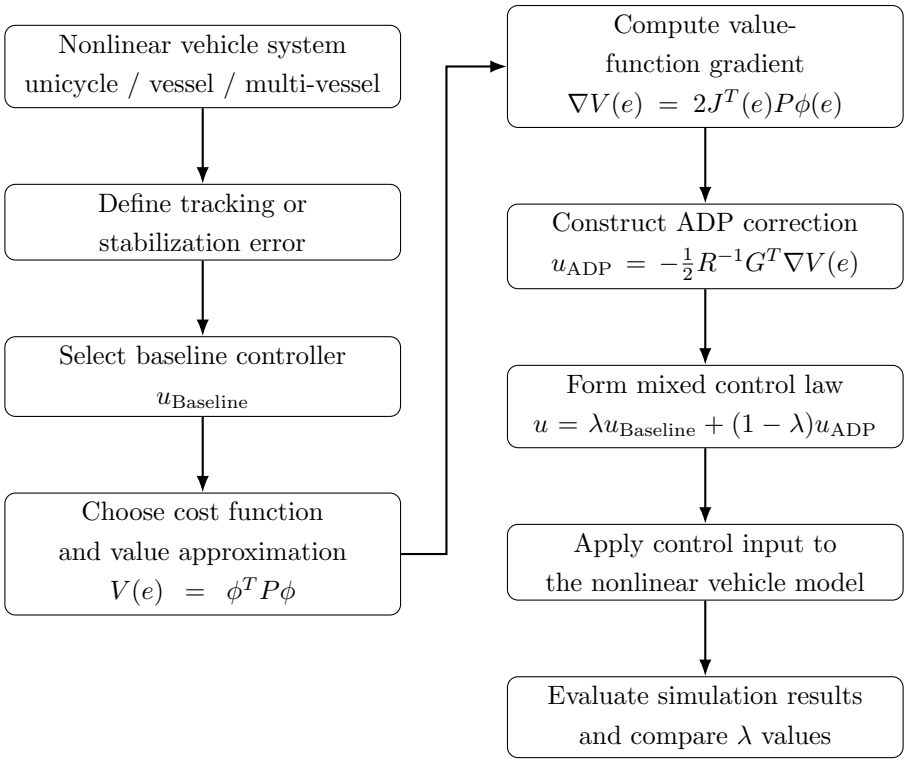

Figure 4.1: Flowchart of the nonlinear ADP-based control structure used for vehicle systems.

4.2 Baseline-Assisted ADP Correction Framework

A general nonlinear control-affine system can be written as

$$\dot{x} = f(x) + g(x)u,$$

where x is the system state, u is the control input, $f(x)$ represents the uncontrolled system dynamics, and $g(x)$ describes how the control input enters the system [12]. For a tracking problem, the desired state is denoted by

$$x_d = x_d(t).$$

In the nonlinear simulations of this thesis, the tracking error is defined as the actual state minus the desired state:

$$e = x - x_d.$$

For angular states, the heading error is wrapped to $(-\pi, \pi]$ to avoid discontinuity. The control objective is to reduce the tracking error while keeping the control input within a

reasonable range.

For the nonlinear error system, the cost function is written as

$$J(e_0; u) = \int_0^\infty \left(e^T Q e + u^T R u\right) dt,$$

where e_0 is the initial error, Q is the error weighting matrix, and R is the control weighting matrix. The matrix Q determines the relative importance of the error states, while R penalizes the control effort. Increasing the weight in Q places more emphasis on reducing the corresponding error state. Increasing the weight in R reduces aggressive control action.

For the nonlinear simulations, the value function is approximated using a feature-based quadratic form:

$$V(e) = \phi^T(e) P \phi(e),$$

where $\phi(e)$ is a selected nonlinear feature vector and P is the approximate value matrix. This form is used instead of a purely quadratic function in the raw error states, because the nonlinear features can include heading-dependent terms such as $\cos(e_\theta)$ and $\sin(e_\theta)$.

The gradient of the approximate value function is written as

$$\nabla V(e) = 2 J^T(e) P \phi(e),$$

where

$$J(e) = \frac{\partial \phi(e)}{\partial e}.$$

This gradient describes how the approximate long-term cost changes with respect to the tracking error, and it is used to construct the ADP correction term.

The ADP correction term is inspired by the value-function-based optimal control structure. For a nonlinear system in the form

$$\dot{x} = f(x) + g(x) u,$$

the optimal control input can be written as

$$u^*(x) = -\frac{1}{2} R^{-1} g^T(x) \nabla V^*(x).$$

In this thesis, the exact optimal value function $V^*(x)$ is not solved. Instead, the approximate value function $V(e) = \phi^T(e) P \phi(e)$ is used to generate a correction input. The ADP correction

term is therefore written as

$$u_{\mathrm{ADP}} = -\frac{1}{2} R^{-1} G^T \nabla V(e),$$

where G is the input distribution matrix associated with the selected error dynamics or input channel.

This correction term is not claimed to be the exact optimal control law for the nonlinear system. It should be interpreted as an approximate value-function-based correction rather than a guaranteed globally optimal controller. Its effect depends on the selected feature vector $\phi(e)$, the value matrix P, the weighting matrix R, and the input distribution matrix G.

4.2.1 Approximate Value Matrix Selection and Update

In the nonlinear simulations, the matrix P is updated from simulation data by reducing the residual of the approximate Bellman equation. During learning, data are collected from the closed-loop system, and the value matrix is adjusted using a gradient-based update. After each update, P is symmetrized and modified to remain positive definite. This procedure provides an approximate value matrix for constructing the ADP correction input.

The Bellman residual is written in an approximate form as

$$\delta(t) = r(e, u) + \nabla V^T(e)\dot{e},$$

where $r(e, u) = e^T Q e + u^T R u$. The value matrix is updated in the direction that reduces this residual. In the implementation, this update is applied numerically using simulation data, and the resulting matrix is symmetrized as

$$P \leftarrow \frac{1}{2}(P + P^T).$$

If necessary, a small positive diagonal term is added to keep the matrix positive definite.

The matrix P is not treated as the exact solution of the nonlinear Hamilton–Jacobi–Bellman equation. Instead, it is used as an approximate value matrix with dimensions consistent with the selected feature vector. In the unicycle case, the feature vector has five components, so $P \in \mathbb{R}^{5\times5}$. In the surface vessel case, the feature vector has eight components, so $P \in \mathbb{R}^{8\times8}$.

A baseline controller is used together with the ADP correction term. The baseline controller provides the main stabilizing behavior and gives a reasonable initial response. Since

the ADP correction term is derived from an approximate value function, it may not provide reliable performance when used alone, especially for large tracking errors. The baseline controller helps maintain a stable control response, while the ADP term provides an additional value-function-based adjustment.

The final control input is formed by combining the baseline controller and the ADP correction term:

$$u = \lambda u_{\text{Baseline}} + (1 - \lambda) u_{\text{ADP}},$$

where u_{Baseline} is the baseline control input, u_{ADP} is the ADP correction term, and $\lambda \in [0, 1]$ is a weighting parameter.

When

$$\lambda = 1,$$

the controller reduces to the baseline controller. When

$$\lambda = 0,$$

the controller relies only on the ADP correction term. For intermediate values of λ, both terms contribute to the final control input. Different values of λ are tested in the simulation results to study the tradeoff between baseline stability and ADP-based correction.

4.3 Baseline-Assisted ADP Control for the Unicycle Robot

4.3.1 Unicycle Model and Error Definitions

The unicycle robot state and control input are defined as

$$X = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \qquad u = \begin{bmatrix} v \\ \omega \end{bmatrix},$$

where x and y are the robot position in the inertial frame, θ is the heading angle, v is the translational velocity, and ω is the angular velocity. The kinematic model is

$$\dot{x} = v \cos\theta,$$

$$\dot{y} = v \sin\theta,$$

$$\dot{\theta} = \omega.$$

The nonlinear terms $\cos\theta$ and $\sin\theta$ show that the robot motion depends on its heading angle. The unicycle robot is nonholonomic and cannot reduce the position error through direct lateral motion. Position regulation must be achieved by coordinating the heading angle with forward or backward motion [4].

For parking and tracking tasks, the desired state is defined as

$$X_d = \begin{bmatrix} x_d \\ y_d \\ \theta_d \end{bmatrix}.$$

The position errors in the inertial frame are

$$e_x = x - x_d, \qquad e_y = y - y_d.$$

For the unicycle robot, the position error is also expressed in the body-fixed frame to make the error variables consistent with the robot motion constraints. Since the robot moves along its heading direction and changes direction through the angular velocity input, the body-frame error provides a useful representation for controller design. The body-frame tracking errors are defined as

$$e_{x,b} = \cos\theta\, e_x + \sin\theta\, e_y,$$

$$e_{y,b} = -\sin\theta\, e_x + \cos\theta\, e_y,$$

$$e_\theta = \text{wrapToPi}(\theta - \theta_d),$$

where $e_{x,b}$ denotes the forward error in the robot body frame, $e_{y,b}$ denotes the lateral error in the body-fixed frame, and e_θ denotes the heading error.

4.3.2 Baseline Controller and ADP Correction Term

Using the error definition introduced in the previous subsection, the state error is written as

$$e = X - X_d.$$

where

$$X = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \qquad X_d = \begin{bmatrix} x_d \\ y_d \\ \theta_d \end{bmatrix}.$$

The heading error is wrapped to $(-\pi, \pi]$:

$$e_\theta = \text{wrapToPi}(\theta - \theta_d).$$

For the parking task, the desired state is fixed at the origin. The position error magnitude is

$$\rho = \sqrt{e_x^2 + e_y^2}.$$

The desired heading toward the target is computed as

$$\theta_{\text{tar}} = \text{atan2}(-e_y, -e_x),$$

and the heading-to-target error is

$$e_{\text{tar}} = \text{wrapToPi}(\theta_{\text{tar}} - \theta).$$

When the robot is far from the target, the baseline parking controller uses

$$v_{\text{Baseline}} = k_x \rho \max(\cos(e_{\text{tar}}), 0),$$

$$\omega_{\text{Baseline}} = k_\theta e_{\text{tar}}.$$

The term $\max(\cos(e_{\text{tar}}), 0)$ reduces forward motion when the robot is not aligned with the target direction. When the robot is close to the target, the translational velocity is reduced. Near the target pose, the controller sets the translational velocity to zero and regulates the final heading angle:

$$v_{\text{Baseline}} = 0, \qquad \omega_{\text{Baseline}} = -k_{vw}\theta.$$

For the circular tracking task, the inertial-frame error is transformed into body-frame position errors:

$$e_{x,b} = \cos\theta\, e_x + \sin\theta\, e_y,$$

$$e_{y,b} = -\sin\theta\, e_x + \cos\theta\, e_y.$$

The desired translational and angular velocities are denoted by v_d and ω_d. The baseline

tracking controller is written as

$$v_{\text{Baseline}} = v_d \cos(e_\theta) - k_x e_{x,b},$$

$$\omega_{\text{Baseline}} = \omega_d - k_y e_{y,b} - k_\theta \sin(e_\theta).$$

The body-frame forward error mainly affects the translational velocity command, while the lateral error and heading error mainly affect the angular velocity command.

For the unicycle model, the system is written as

$$\dot{X} = g(X)u,$$

where

$$g(X) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix}.$$

The approximate value function uses the nonlinear feature vector

$$\phi(e) = \begin{bmatrix} e_x \\ e_y \\ e_\theta \\ \cos(e_\theta) \\ \sin(e_\theta) \end{bmatrix}.$$

The value function approximation is

$$V(e) = \phi^T(e) P \phi(e),$$

where $P \in \mathbb{R}^{5\times 5}$ is the approximate value matrix. The gradient of the value function is

$$\nabla V(e) = 2J^T(e) P \phi(e),$$

where

$$J(e) = \frac{\partial \phi(e)}{\partial e}.$$

The ADP correction input is then computed as

$$u_{\text{ADP}} = -\frac{1}{2} R^{-1} g^T(X) \nabla V(e).$$

The final control input is obtained by combining the baseline controller and the ADP correction term:

$$u = \lambda u_{\text{Baseline}} + (1 - \lambda) u_{\text{ADP}},$$

where

$$u = \begin{bmatrix} v \\ \omega \end{bmatrix}.$$

Here, u_{Baseline} denotes the parking or tracking baseline input, u_{ADP} denotes the approximate ADP correction input, and $\lambda \in [0, 1]$ determines the relative influence of the two terms.

4.3.3 Simulation Setup

Two tasks are tested for the nonlinear unicycle model: parking and circular tracking. In the parking task, the desired state is fixed at the origin. In the tracking task, the desired reference is a time-varying circular trajectory.

The mixed control law is evaluated using different values of the weighting parameter λ:

$$\lambda \in \{0,\ 0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9,\ 1.0\}.$$

When $\lambda = 1$, the controller reduces to the baseline controller. When $\lambda = 0$, the controller uses only the ADP correction term. Intermediate values of λ combine the baseline controller and the ADP correction term. The simulation results are evaluated using the robot trajectory, state responses, position error curves, heading error, and comparisons between different values of λ.

4.3.4 Parking Evaluation

Figure 4.2 shows the parking result of the nonlinear unicycle robot under the mixed ADP-based control law. In this task, the desired state is fixed at the origin. The figure includes the robot trajectory, the state responses, the position errors, and the heading error.

Compared with the linearized ADP parking result in Chapter 3, the nonlinear controller is more suitable for this task. The simulation keeps the nonlinear motion of the unicycle model, which is important for parking behavior. The robot cannot move directly in the lateral direction, so parking must be achieved through coordinated heading adjustment and forward motion.

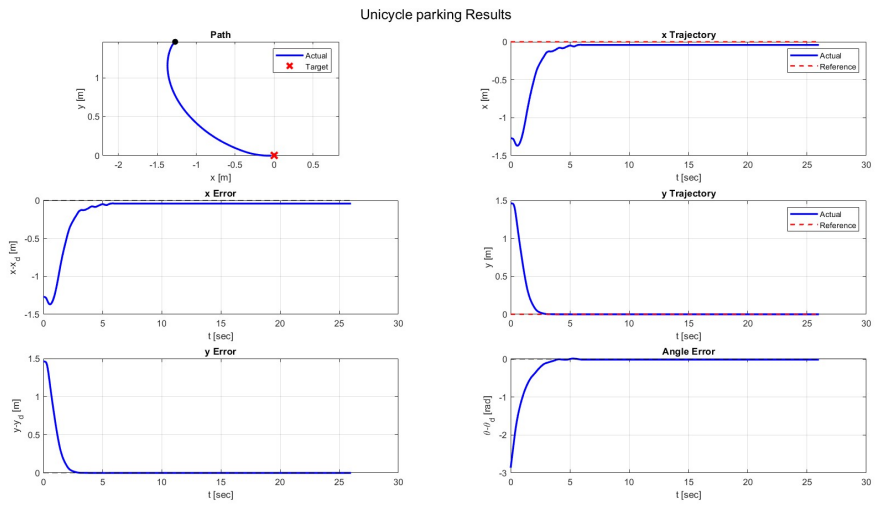

Figure 4.2: Unicycle parking result under the mixed ADP-based control law.

4.3.5 Circular Tracking Evaluation

Figure 4.3 shows the circular tracking result of the nonlinear unicycle robot. The figure includes the trajectory, the x- and y-position responses, the position error curves, and the heading error curve. The result is used to evaluate whether the robot can follow the circular reference trajectory and reduce the tracking errors under the mixed ADP-based controller.

In the circular tracking case, the reference trajectory changes continuously with time. The baseline controller provides the main tracking behavior, while the ADP correction modifies the velocity command based on the approximate value function. The result shows how the nonlinear ADP-based control structure performs when the robot follows a moving reference path.

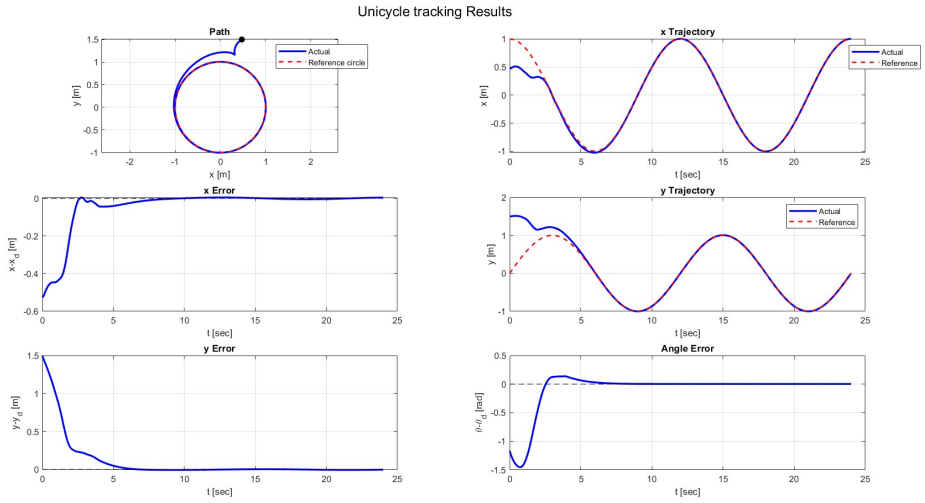

Figure 4.3: Unicycle circular tracking result under the mixed ADP-based control law.

4.3.6 Influence of the Mixing Parameter λ

Figure 4.4 compares the unicycle responses under different values of λ. A larger value of λ makes the controller closer to the baseline controller, while a smaller value gives more influence to the ADP correction term.

The comparison shows that the weighting parameter affects the convergence behavior, trajectory shape, and smoothness of the response. When λ is close to one, the response is mainly determined by the baseline controller and is usually more predictable. When λ is close to zero, the ADP correction has a stronger effect on the control input. The selection of λ should balance the stabilizing behavior of the baseline controller and the performance adjustment from the ADP correction term.

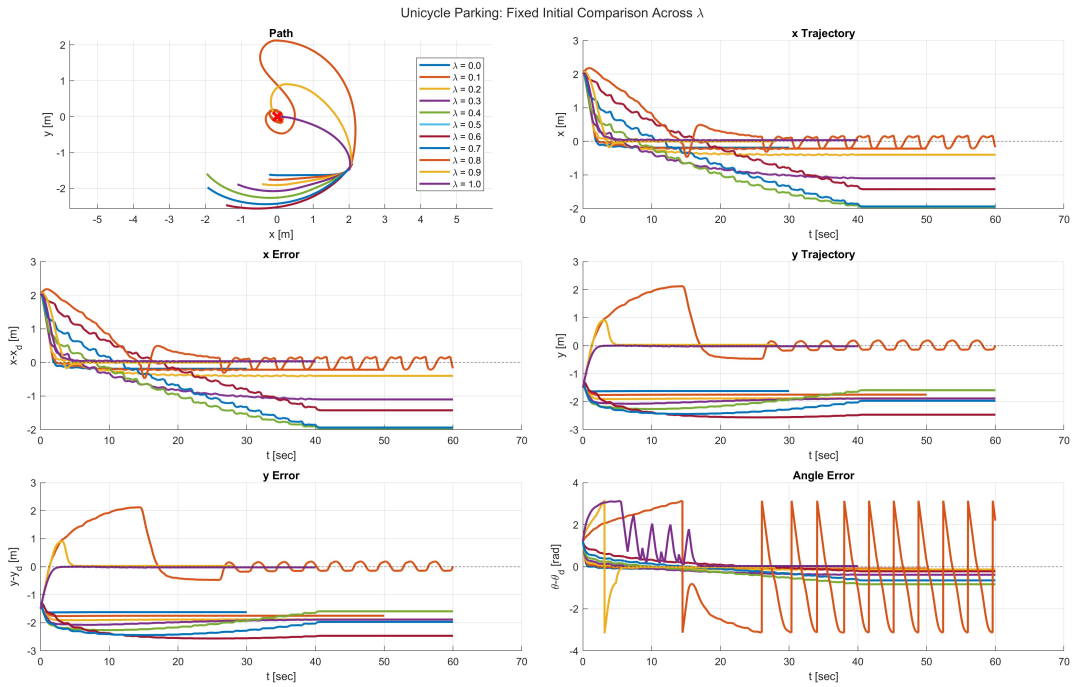

Figure 4.4: Unicycle comparison under different values of λ.

4.3.7 Comparison with the Linearized ADP Controller

The nonlinear ADP-based controller in this section is compared with the linearized ADP method in Chapter 3. Both chapters use the unicycle robot as the test system, but the modeling and controller design are different.

In Chapter 3, the nonlinear tracking-error dynamics are approximated by a local linear model:

$$\dot{e} = Ae + B\tilde{u}.$$

The linear ADP controller is designed for this local model, and the value function is written as $V(e) = e^T Pe$. This approach is useful for local tracking, but it depends on the assumption that the tracking error remains small.

In this chapter, the controller is tested directly on the nonlinear unicycle model. The baseline controller provides the main nonlinear control behavior, and the ADP correction term is added through the approximate value function. The final control input is

$$u = \lambda u_{\text{Baseline}} + (1 - \lambda) u_{\text{ADP}}.$$

The main difference is that the linearized ADP method uses a local approximation of

the error dynamics, while the nonlinear mixed controller keeps the nonlinear motion in the simulation and uses the baseline controller to support larger-error behavior. This difference is especially important for the parking task, where the robot must use nonlinear motion to reduce lateral position error and heading error. Thus, the nonlinear ADP-based control structure is more suitable than the purely linearized ADP controller for parking and large-error maneuvers.

4.4 Baseline-Assisted ADP Control for the Surface Vessel

Compared with the unicycle robot, the surface vessel is a more complex nonlinear system. Its control inputs are surge force and yaw moment, rather than direct velocity commands. The vessel motion is affected by inertia, damping, nonlinear coupling, and underactuation. This makes the surface vessel a more challenging test case for the nonlinear ADP-based control structure.

4.4.1 Surface Vessel Model

The surface vessel state is defined as

$$X = \begin{bmatrix} x & y & \theta & u & v & r \end{bmatrix}^T,$$

where x and y are the vessel position in the earth-fixed frame, θ is the heading angle, u is the surge velocity, v is the sway velocity, and r is the yaw rate. To avoid confusion between the sway velocity v and the control input, the body-fixed velocity vector and the control input are written as

$$\nu = \begin{bmatrix} u \\ v \\ r \end{bmatrix}, \qquad \tau = \begin{bmatrix} \tau_u \\ \tau_r \end{bmatrix},$$

where τ_u is the surge force and τ_r is the yaw moment. The vessel is underactuated because there is no direct control input in the sway direction.

The kinematic equations are

$$\dot{x} = u\cos\theta - v\sin\theta,$$

$$\dot{y} = u \sin\theta + v \cos\theta,$$

$$\dot{\theta} = r.$$

These equations describe how the body-fixed velocities are transformed into earth-fixed position and heading motion.

The dynamic equations are written as

$$\dot{u} = \frac{m_{22}}{m_{11}} vr - \frac{d_{11}}{m_{11}} u + \frac{1}{m_{11}} \tau_u,$$

$$\dot{v} = -\frac{m_{11}}{m_{22}} ur - \frac{d_{22}}{m_{22}} v,$$

$$\dot{r} = \frac{m_{11} - m_{22}}{m_{33}} uv - \frac{d_{33}}{m_{33}} r + \frac{1}{m_{33}} \tau_r,$$

where m_{11}, m_{22}, and m_{33} are inertia parameters, while d_{11}, d_{22}, and d_{33} are damping parameters. The product terms vr, ur, and uv show the coupling among surge, sway, and yaw motion.

4.4.2 Baseline Controller

A baseline controller is used to provide the main stabilizing behavior for the vessel. Unlike the unicycle robot, the surface vessel controller does not directly command the translational and angular velocities. Instead, it generates the surge force τ_u and the yaw moment τ_r.

For the tracking task, the baseline controller uses the position error, heading error, surge velocity error, and yaw rate error to generate the force and moment commands. The surge force mainly regulates the forward motion, while the yaw moment adjusts the heading direction. Since the vessel has no direct sway input, the lateral position error must be reduced indirectly through heading adjustment and surge motion. For the parking task, the baseline controller drives the vessel toward the target pose and reduces the heading error near the target. The response is usually slower than the unicycle case because the vessel has inertia and damping, and the commanded force and moment do not change the position immediately.

4.4.3 ADP Correction Term for the Vessel Model

For the surface vessel, the ADP correction is added at the force and moment level. The control input is

$$\tau = \begin{bmatrix} \tau_u \\ \tau_r \end{bmatrix},$$

where τ_u is the surge force and τ_r is the yaw moment.

The vessel tracking error is defined as

$$e = X - X_d,$$

where $X \in \mathbb{R}^6$ is the vessel state and X_d is the desired state. The heading error is wrapped to $(-\pi, \pi]$. The approximate value function uses the feature vector

$$\phi(e) = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ \cos(e_3) \\ \sin(e_3) \end{bmatrix}.$$

The value function approximation is

$$V(e) = \phi^T(e) P \phi(e),$$

where $P \in \mathbb{R}^{8\times 8}$ is the approximate value matrix. Its gradient is

$$\nabla V(e) = 2 J^T(e) P \phi(e),$$

where $J(e) = \partial\phi(e)/\partial e$.

The control input enters the vessel dynamics through the surge and yaw equations.

Therefore, the input distribution matrix for the force and moment inputs is

$$G = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{m_{11}} & 0 \\ 0 & 0 \\ 0 & \frac{1}{m_{33}} \end{bmatrix}.$$

Using the nonlinear ADP correction structure, the vessel ADP input is written as

$$\tau_{\text{ADP}} = -\frac{1}{2} R^{-1} G^T \nabla V(e).$$

The final vessel control input is formed by combining the baseline controller and the ADP correction:

$$\tau = \lambda \tau_{\text{Baseline}} + (1 - \lambda)\tau_{\text{ADP}}.$$

4.4.4 Simulation Setup

Two tasks are tested for the surface vessel model: parking and circular tracking. In the parking task, the vessel moves toward a fixed target pose. In the circular tracking task, the vessel follows a time-varying circular reference path. The same mixed control structure is used for both tasks:

$$\tau = \lambda \tau_{\text{Baseline}} + (1 - \lambda)\tau_{\text{ADP}}.$$

Different values of λ are tested to study how the balance between the baseline controller and the ADP correction affects the vessel response. The simulation results are evaluated using the vessel trajectory, position responses, tracking errors, heading error, and comparison plots under different values of λ.

4.4.5 Parking Evaluation

Figure 4.5 shows the parking result of the surface vessel under the mixed ADP-based control law. In this task, the desired state is fixed at the target pose. The figure shows the vessel trajectory, state responses, position errors, and heading error.

Compared with the unicycle robot, the vessel response is slower and less direct. The

controller acts through force and moment inputs, which first affect the vessel velocities. The position then changes through the kinematic equations. In addition, the sway direction is not directly actuated. Lateral motion must therefore be adjusted indirectly through the coupled vessel dynamics.

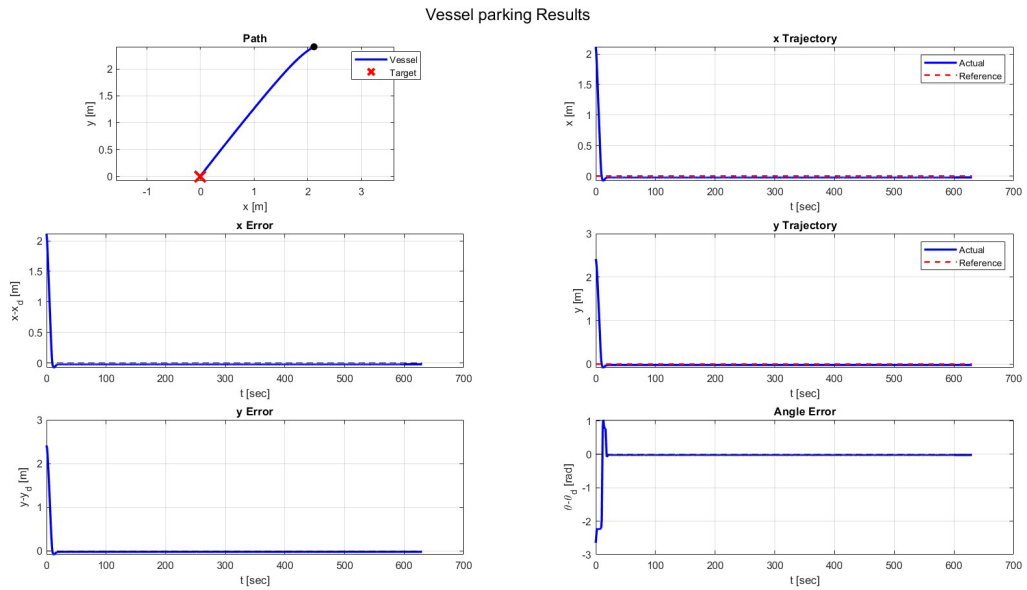

Figure 4.5: Single-vessel parking result under the mixed ADP-based control law.

4.4.6 Circular Tracking Evaluation

Figure 4.6 shows the circular tracking result of the surface vessel. The figure includes the vessel trajectory, the x- and y-position responses, the position error curves, and the heading error curve. The result is used to evaluate whether the vessel can follow the circular reference path under the mixed ADP-based control structure. The transient response is more complex than in the unicycle case. This is expected because the surface vessel has dynamic states, damping terms, nonlinear coupling, and underactuation.

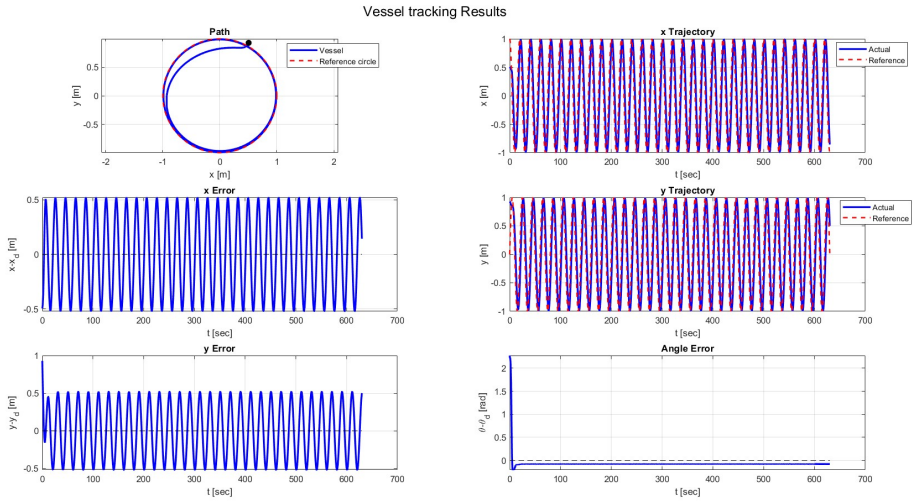

Figure 4.6: Single-vessel circular tracking result under the mixed ADP-based control law.

4.4.7 Influence of the Mixing Parameter λ

Figure 4.7 compares the circular tracking results under different values of λ. A larger value of λ makes the vessel response closer to the baseline controller, while a smaller value gives more influence to the ADP correction term.

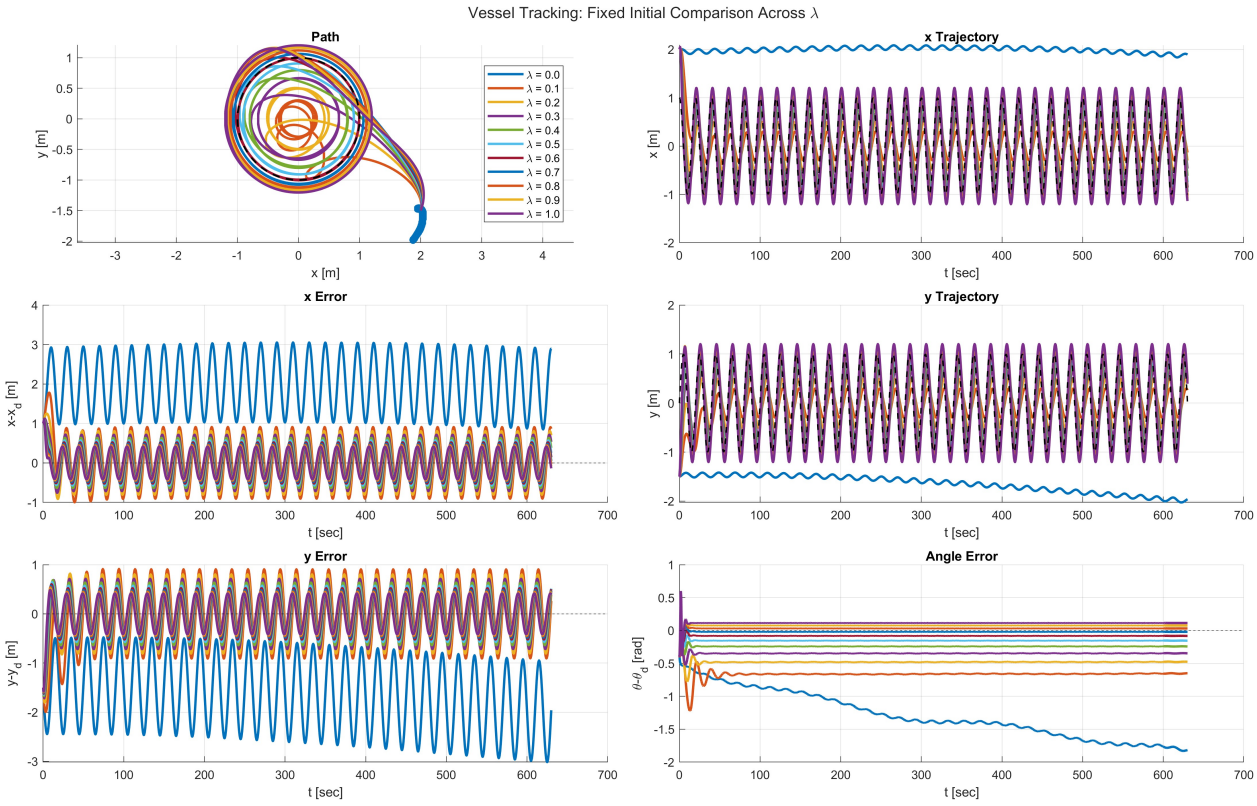

Figure 4.7: Single-vessel circular tracking comparison under different values of λ.

Figure 4.8 compares the parking results under different values of λ. The comparison shows how the weighting parameter affects convergence behavior, trajectory shape, and heading response. A suitable value of λ should reduce the error while avoiding large oscillations or aggressive control input. The best value of λ is not determined only by the smallest final error, but also by the smoothness and stability of the response.

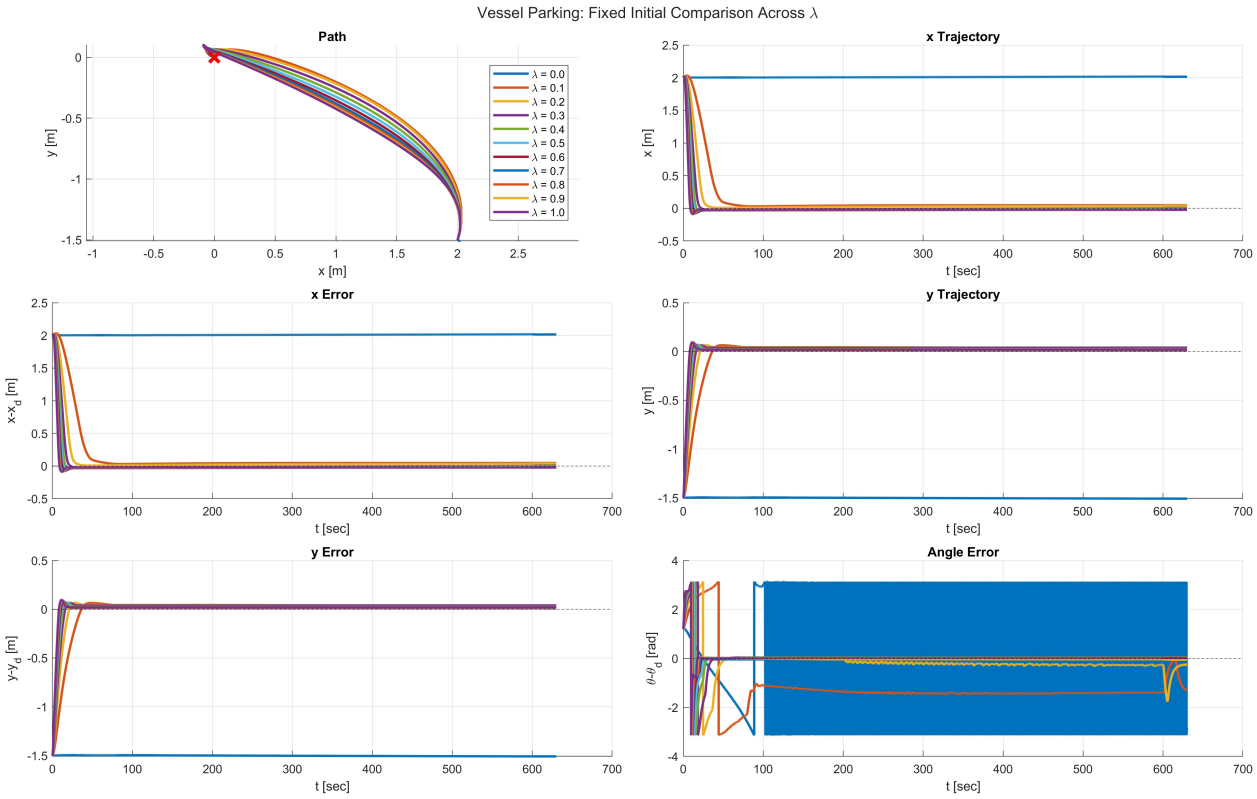

Figure 4.8: Single-vessel parking comparison under different values of λ.

4.5 Random Initial Condition Evaluation

To further evaluate the robustness of the proposed mixed control structure, 50 random initial conditions were tested for each value of λ. The random tests were performed for the unicycle parking task, unicycle tracking task, vessel parking task, and vessel tracking task. The results were summarized using success rate and representative error metrics.

4.5.1 Random Evaluation for the Unicycle Robot

The unicycle results show that larger values of λ lead to more reliable behavior. For the parking task, $\lambda = 0.9$ and $\lambda = 1.0$ both achieved 100% success, while smaller values of λ had much lower success rates. For the tracking task, the controller achieved 100% success when $\lambda \geq 0.7$, and the steady RMS position error decreased as λ increased.

These results suggest that the unicycle parking and tracking tasks depend strongly on the stabilizing behavior of the baseline controller. A stronger ADP correction does not necessarily improve the response in these cases, especially when the baseline controller already provides

stable behavior.

4.5.2 Random Evaluation for the Surface Vessel

The vessel results show a similar trend for parking, but a different trend for tracking. In the vessel parking task, the success rate increased as λ increased, and all cases with $\lambda \geq 0.5$ achieved 100% success. In the vessel tracking task, all cases with $\lambda \geq 0.3$ achieved 100% success, but the lowest steady RMS radial error occurred at $\lambda = 0.6$. This indicates that an intermediate balance between the baseline controller and the ADP correction can improve the tracking accuracy for the vessel tracking case.

4.5.3 Summary of Random Evaluation Results

Table 4.1 summarizes the best-performing or representative values of λ from the random-initial-condition evaluation. For the parking tasks, the main error metric is the mean final position error. For the tracking tasks, the main error metric is the mean steady-state RMS error.

Table 4.1: Selected random-initial-condition evaluation results over 50 trials.

Task	Representative λ	Success Rate	Main Error Metric
Unicycle parking	0.9	100%	Final position error: 0.027
Unicycle tracking	1.0	100%	Steady RMS position error: 0.005
Vessel parking	0.5	100%	Final position error: 0.035
Vessel tracking	0.6	100%	Steady RMS radial error: 0.005

These results show that the effect of λ depends on the task and the vehicle model. In parking tasks, the baseline controller is important for reliable stabilization. In tracking tasks, the ADP correction can improve the response when it is combined with the baseline controller using a suitable value of λ.

4.6 Preliminary Leader-Follower Extension for Multiple Vessels

This section extends the single-vessel tracking problem to a multiple-vessel leader-follower case. The purpose is to test whether the nonlinear vessel control structure can be used in a coordinated motion scenario. In this thesis, the multiple-vessel case is treated as a preliminary extension rather than a complete formation control solution.

4.6.1 Leader-History-Based Reference Generation

Figure 4.9 illustrates the leader-history following reference used in the multiple-vessel simulation. The leader tracks the desired path, while each follower tracks a delayed state from the leader's trajectory history. Therefore, the followers do not directly track the global reference path.

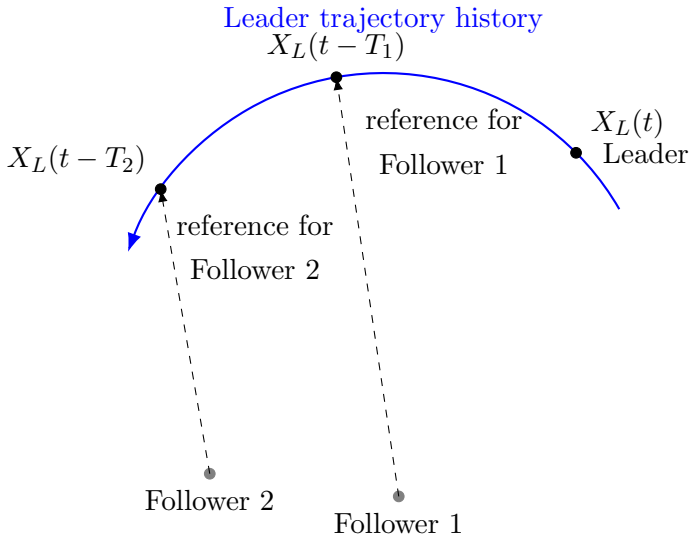

Figure 4.9: Leader-history following reference for the multiple-vessel system. Each follower tracks a delayed state from the leader trajectory rather than the global reference path directly.

For follower i, the desired following distance is denoted by d_i. Since the leader moves with the desired cruise speed v_d, this distance is converted into a time delay:

$$T_i = \frac{d_i}{v_d}, \qquad i = 1, 2.$$

The desired state of follower i is then obtained from the leader trajectory history:

$$X_{d,i}(t) = X_L(t - T_i), \qquad i = 1, 2.$$

Each follower tracks a past state of the leader instead of tracking the global reference path directly. The follower tracking error is computed as the follower state minus the delayed leader reference:

$$e_i = X_i - X_{d,i}, \qquad i = 1, 2.$$

The heading error is wrapped to the interval $(-\pi, \pi]$ to avoid angle discontinuity. The leader-history structure provides a simple way to maintain spacing along the path while allowing all vessels to move with the same nominal cruise speed.

4.6.2 Multiple-Vessel Model

The system includes one leader vessel and two follower vessels. The leader vessel follows the desired reference path, while the followers track delayed states from the leader trajectory. The state of the leader vessel and the state of the i-th follower vessel are defined as

$$X_L = \begin{bmatrix} x_L & y_L & \theta_L & u_L & v_L & r_L \end{bmatrix}^T, \qquad X_i = \begin{bmatrix} x_i & y_i & \theta_i & u_i & v_i & r_i \end{bmatrix}^T, \quad i = 1, 2.$$

Each vessel uses the same surface vessel kinematic and dynamic model introduced in the previous section. The leader vessel tracks the desired reference path, while the follower vessels use delayed states from the leader trajectory history as their references.

4.6.3 Controller Setup

The leader vessel uses the mixed baseline-ADP controller to follow the circular reference path. The leader control input is written as

$$\tau_L = \lambda \tau_{\text{Baseline},L} + (1 - \lambda)\tau_{\text{ADP},L}.$$

The baseline part provides the main path-following behavior, while the ADP correction modifies the surge force and yaw moment through the approximate value function.

The follower vessels use baseline leader-history tracking controllers. Their reference states are taken from delayed states of the leader trajectory, and their control inputs are generated from tracking errors relative to those delayed references. In this implementation, the ADP

correction is applied only to the leader vessel. The followers do not use separate ADP correction terms; they track the leader-history references using the baseline vessel controller.

A small artificial potential field safety correction is also added when the vessels become close to each other. This correction is used only as a safety adjustment in the simulation and is not treated as a complete collision-avoidance design. Therefore, the multiple-vessel case should be understood as a preliminary coordinated cruising extension rather than a full multi-agent ADP or formation-control method.

4.6.4 Cruising Evaluation

Figure 4.10 shows the leader-follower cruising result. The leader follows the circular reference path, while the followers track delayed states from the leader's trajectory history. The figure includes the path view, the x- and y-trajectory responses, the position errors relative to each vessel's own reference, the leader-history tracking errors for the followers, and the heading errors.

In the path subplot, only the first part of the trajectory is shown so that the relative motion between the leader and the followers can be observed more clearly. The result shows that the followers move along the path previously traveled by the leader, rather than tracking the global circular reference directly.

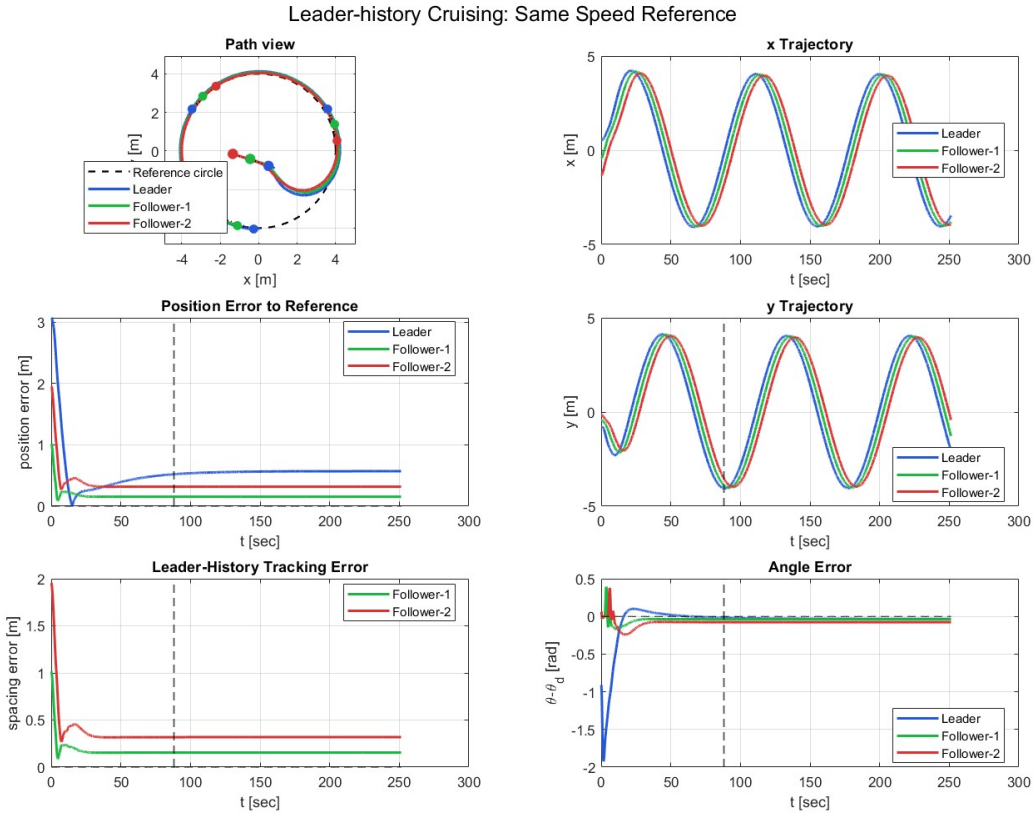

Figure 4.10: Leader-history cruising result with one leader and two followers.

4.6.5 Formation Snapshot Evaluation

Figure 4.11 shows selected time snapshots of the leader-follower motion. The snapshots give a clearer view of the relative vessel positions during cruising. They also show how the followers remain behind the leader along the path according to their leader-history references.

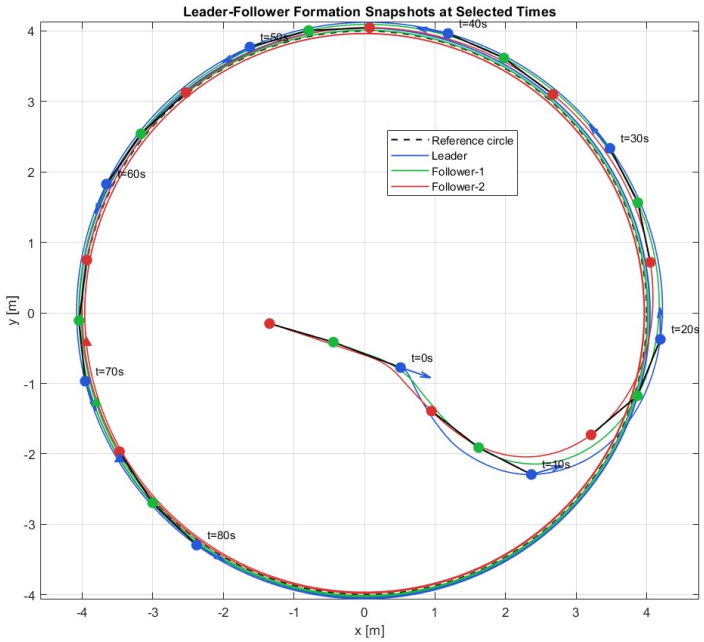

Figure 4.11: Leader-follower formation snapshots at selected time instants.

4.7 Discussion

This chapter applied the mixed baseline-ADP control structure to the nonlinear unicycle robot and the underactuated surface vessel. The same idea was then extended to a preliminary multiple-vessel leader-follower simulation, where the leader used the mixed baseline-ADP controller and the followers used baseline leader-history tracking controllers. Compared with the linearized ADP controller in Chapter 3, the nonlinear structure depends less on a local linear approximation. The baseline controller provides the main stabilizing behavior, and the ADP correction term adjusts the control input through an approximate value function.

For the unicycle robot, the mixed baseline-ADP controller was tested in both parking and circular tracking tasks. The unicycle model is simpler than the vessel model because its control inputs directly specify the translational and angular velocities. However, it still includes nonlinear and nonholonomic motion. The results show that the nonlinear mixed control structure is more suitable for parking than the purely linearized ADP controller in Chapter 3, because the parking task requires nonlinear motion, such as heading adjustment and forward movement.

For the surface vessel, the same control structure was extended to a more complex nonlin-

ear system. Unlike the unicycle robot, the vessel uses surge force and yaw moment as control inputs, rather than direct velocity commands. Its response is affected by inertia, damping, nonlinear coupling, and underactuation. Therefore, the transient response is slower and more complex than the unicycle response. The results show that the mixed ADP-based controller can be applied to both vessel parking and circular tracking tasks, but its performance depends strongly on the baseline controller and the selected weighting parameter.

The weighting parameter λ has a clear influence on the mixed control law. When $\lambda = 1$, the controller reduces to the baseline controller. When $\lambda = 0$, the controller relies only on the ADP correction term. Intermediate values of λ combine both components. In many parking cases, larger values of λ produced smoother and more predictable behavior, since the controller remained close to the baseline design. For the vessel tracking task, however, the best tracking accuracy was obtained with an intermediate value of λ. This result suggests that the selection of λ should be treated as task-dependent rather than fixed.

The multiple-vessel simulation further extends the single-vessel case to a leader-follower cruising task. In this setup, the leader tracks the desired path, while the followers track delayed states from the leader's trajectory history. This allows the followers to move along the path previously traveled by the leader. The results show that the leader-history reference provides a simple way to generate coordinated motion. However, this part is only a preliminary extension. It does not provide a complete formation control design, and it does not fully address collision avoidance, communication delay, or distributed stability.

The simulations in this chapter are used to evaluate the feasibility of the proposed nonlinear ADP-based control structure. They do not provide complete proof of optimality for the nonlinear systems. The value function is approximated, and the final performance depends on the selected error definition, baseline controller, ADP correction term, weighting matrices, and value of λ. Future work may include a more systematic selection of λ, more flexible value-function approximation, and testing under stronger model uncertainty and external disturbances.

Chapter 5

Conclusions and Future Work

5.1 Summary of Findings

This thesis studied Adaptive Dynamic Programming (ADP)-based control for nonlinear vehicle systems. The systems considered in this work include a unicycle-type mobile robot, a single underactuated surface vessel, and a multiple-vessel leader-follower system. The main objective was to examine how ADP-based control can be applied to vehicle models with nonlinear motion, nonholonomic constraints, underactuation, and dynamic coupling.

The thesis first considered a linearized ADP controller for the unicycle robot. The nonlinear unicycle tracking-error dynamics were locally linearized around the reference trajectory, and the resulting linear error model was used for linear ADP design. This part provided a bridge between the linear ADP theory and the nonlinear vehicle control problem. The results showed that the linearized ADP controller can be useful for local tracking, but its performance is limited when the robot performs parking or starts with a relatively large initial error.

The main part of the thesis focused on a mixed baseline-ADP control structure for nonlinear vehicle systems. For the nonlinear vehicle systems, a baseline controller was combined with an ADP correction term. The final control input was written in the mixed form

$$u = \lambda u_{\text{Baseline}} + (1 - \lambda) u_{\text{ADP}},$$

where u_{Baseline} is the baseline control input, u_{ADP} is the ADP-based correction input, and λ is a weighting parameter.

Simulation results were presented for the unicycle robot, the single surface vessel, and

the multiple-vessel system. The unicycle case provided a basic nonlinear test case because it has a simple kinematic model but still includes the nonholonomic constraint. The single-vessel case was more difficult because the controller generated force and moment inputs for an underactuated dynamic system. The multiple-vessel case further extended the study to leader-follower coordinated motion.

The simulation results showed that the weighting parameter λ has an important effect on the controller behavior. Larger values of λ keep the response closer to the baseline controller and often produce smoother behavior. Smaller values of λ increase the influence of the ADP correction term and may change the transient response more strongly. The random initial condition tests also showed that the best choice of λ depends on the task. Parking tasks were mainly supported by the baseline controller, while the vessel tracking case benefited from an intermediate value of λ.

An earlier Deep Deterministic Policy Gradient (DDPG) reinforcement learning controller was also included as supplementary baseline material. The DDPG simulations were developed during the early stage of the project for kinematic and dynamic vehicle models in Simulink. However, since the final research direction shifted toward ADP, the DDPG results were not treated as the main contribution. They were included only as additional background and preliminary learning-based comparison.

5.2 Limitations

Although the simulation results are useful, this thesis has several limitations. The linearized ADP controller was designed from a local approximation of the unicycle tracking-error dynamics. Therefore, it is mainly suitable for local tracking behavior and does not fully represent the nonlinear motion required in parking or large-error cases.

The nonlinear ADP controller used an approximate value function. For nonlinear systems, the exact optimal value function is generally difficult to obtain. Therefore, the ADP correction term should be understood as an approximate control component rather than a guaranteed globally optimal solution. In addition, the performance of the mixed controller depends on the baseline controller. If the baseline controller is not well designed, the final mixed control law may not perform well.

The selection of λ was based on simulation comparison. Several fixed values of λ were tested in this thesis, but an automatic or adaptive method for selecting the best value was not developed. The multiple-vessel case was also treated as a preliminary leader-follower

extension. More advanced issues, such as communication delay, measurement noise, collision avoidance, disturbance rejection, and rigorous formation stability analysis, were not fully studied.

All results in this thesis were simulation-based. The controller was not tested on real hardware or a high-fidelity experimental vessel platform. Therefore, practical implementation issues remain for future study.

5.3 Future Work

Several directions can be considered for future work.

First, the selection of λ can be made more systematic. Instead of testing fixed values manually, an adaptive rule could be developed to adjust λ during simulation. For example, the controller could assign more weight to the baseline controller when the error is large and gradually increase the influence of the ADP correction term as the system moves closer to the desired state.

Second, the value function approximation can be improved. In this thesis, a quadratic approximate value function was used for the ADP correction term. Future work could consider more flexible approximation structures, such as polynomial basis functions, neural-network-based critics, or other parameterized value functions.

Third, stronger uncertainty and disturbance effects can be included in the surface vessel simulations. Real vessels are affected by wind, waves, current, parameter changes, and unmodeled hydrodynamic effects. Adding these effects would make the simulation closer to practical marine applications.

Fourth, the multiple-vessel leader-follower system can be extended to more general formation control problems. Future studies could consider more vessels, different formation shapes, communication constraints, collision avoidance, and distributed control laws. A more complete stability analysis would also be needed for a general multi-vessel system.

Finally, future work could include experimental validation. Since all results in this thesis are simulation-based, testing the proposed controller on hardware or a higher-fidelity vessel simulator would be useful for evaluating practical implementation issues.

Appendix A

A Preliminary DDPG Baseline for Unicycle Control

A.1 DDPG Method

This section presents the preliminary Deep Deterministic Policy Gradient (DDPG) results from the early stage of this project. Before the final research direction shifted toward Adaptive Dynamic Programming (ADP), a DDPG-based reinforcement learning controller was implemented in Simulink for nonlinear vehicle control. The DDPG results are included only as preliminary reinforcement learning results.

The DDPG simulations were performed using kinematic and dynamic vehicle models. The kinematic model uses velocity-level control inputs, while the dynamic model uses force and moment inputs. These two cases helped develop the Simulink reinforcement learning environment and provided experience with observation design, action scaling, reward function design, and training behavior. The DDPG agent contains two main networks: an actor and a critic. The actor generates the control action from the current observation, while the critic estimates the action-value function. During training, the critic learns to evaluate the long-term return, and the actor is updated to generate actions that improve the expected return.

For a state s, the actor can be written as

$$a = \mu(s), \tag{A.1}$$

where a is the continuous action and $\mu(\cdot)$ is the deterministic policy represented by the actor

network.

The critic estimates the action-value function:

$$Q(s, a), \tag{A.2}$$

which represents the expected long-term return when action a is applied in state s.

In the Simulink implementation, the environment provides an observation to the agent at each time step. The actor outputs an action, and this action is applied to the vehicle model. The environment then updates the vehicle state and returns a reward based on the tracking or stabilization performance. Through repeated simulation episodes, the agent learns a policy that improves the accumulated reward.

A.2 DDPG Simulation for the Kinematic Unicycle Model

The first DDPG case uses the unicycle-type kinematic model. The state of the vehicle is

$$X = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \tag{A.3}$$

where x and y are the planar position and θ is the heading angle.

The action vector is defined as

$$a = \begin{bmatrix} v \\ \omega \end{bmatrix}, \tag{A.4}$$

where v is the translational velocity and ω is the angular velocity. The kinematic equations are

$$\begin{aligned} \dot{x} &= v \cos\theta, \\ \dot{y} &= v \sin\theta, \\ \dot{\theta} &= \omega. \end{aligned} \tag{A.5}$$

The control objective is to reduce the position and heading errors with respect to the desired reference or target. The observation provided to the DDPG agent includes information related to the vehicle state and error variables. The reward function is designed to encourage small position error, small heading error, bounded control effort, and safe motion inside the workspace.

A general reward structure can be written as

$$r = r_{\text{tracking}} + r_{\text{heading}} + r_{\text{control}} + r_{\text{boundary}}, \tag{A.6}$$

where r_{tracking} penalizes position error, r_{heading} penalizes heading error, r_{control} penalizes excessive control effort, and r_{boundary} penalizes unsafe motion outside the workspace.

Figure A.1 shows the preliminary DDPG result for the kinematic vehicle model. The result shows that the trained policy can guide the vehicle toward the desired target from different initial conditions. However, the response depends strongly on the reward function design, action limits, and training process.

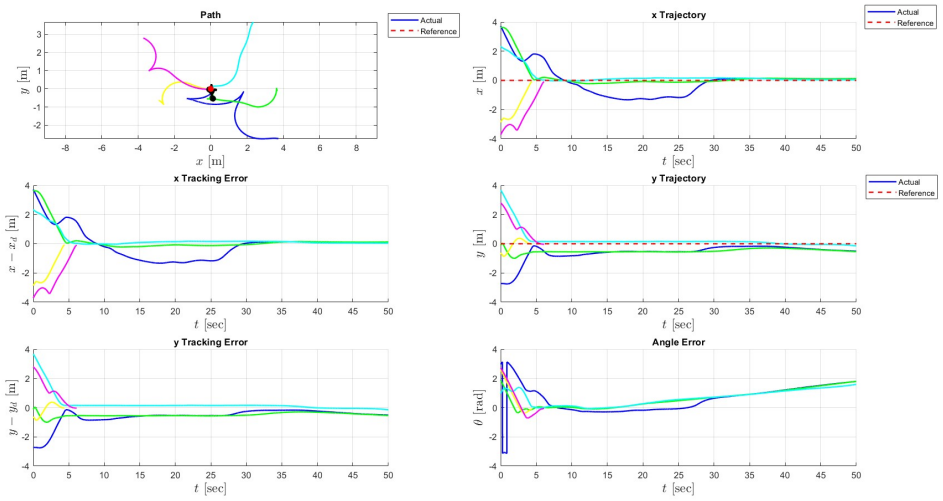

Figure A.1: Preliminary DDPG result for the kinematic vehicle model from different initial conditions.

A.3 DDPG Simulation for the Dynamic Vehicle Model

The second DDPG case uses a dynamic vehicle model. Compared with the kinematic model, the dynamic model is more difficult because the control inputs are force and moment instead

of direct velocity commands. The state includes both pose and velocity variables:

$$X = \begin{bmatrix} x \\ y \\ \theta \\ v_x \\ v_y \\ \omega \end{bmatrix}, \tag{A.7}$$

where x, y, and θ describe the vehicle pose, while v_x, v_y, and ω describe the body-fixed velocities and yaw rate.

The action vector is written as

$$a = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \tag{A.8}$$

where u_1 is the longitudinal force and u_2 is the yaw moment.

The dynamic model can be written in the general form

$$M\dot{\nu} + C(\nu)\nu + D\nu = G_u u, \tag{A.9}$$

where M is the inertia matrix, $C(\nu)$ represents Coriolis and centripetal effects, D is the damping matrix, G_u is the actuation matrix, and u is the control input.

The dynamic model is underactuated because there is no direct control input in the lateral direction. Therefore, the agent must reduce the position and heading errors through the coupling between surge motion, sway motion, and yaw motion. This makes the dynamic DDPG case more difficult than the kinematic case.

The reward function for the dynamic case is designed to balance pose error, velocity damping, control effort, and safety. A general form is

$$r = r_{\text{pose}} + r_{\text{energy}} + r_{\text{control}} + r_{\text{boundary}}. \tag{A.10}$$

The pose term encourages the vehicle to approach the target pose. The energy term penalizes large velocity and helps reduce overshoot. The control term penalizes excessive force and moment. The boundary term prevents the vehicle from leaving the allowed workspace.

Figure A.2 shows a preliminary DDPG result for the dynamic vehicle model. The result indicates that the agent can generate a motion response for the force- and moment-level system. However, compared with the kinematic case, the response is more difficult to tune

because the state dimension is larger and the vehicle motion is affected by inertia, damping, and underactuation.

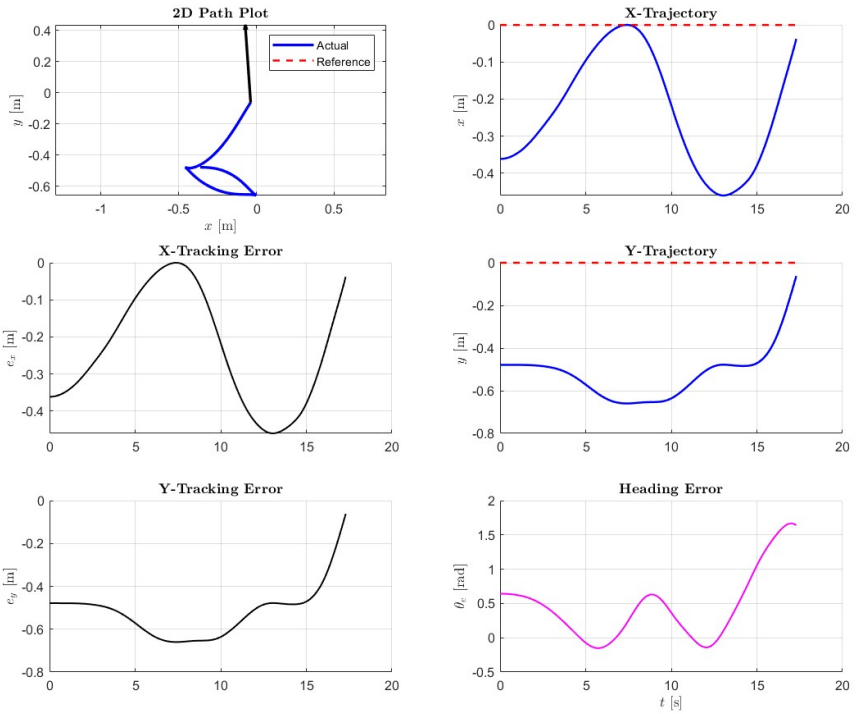

Figure A.2: Preliminary DDPG result for the dynamic vehicle model.

A.4 Discussion and Comparison

The preliminary DDPG simulations show that reinforcement learning can be applied to nonlinear vehicle control problems in Simulink. The kinematic case is simpler because the agent directly generates velocity commands. The dynamic case is more difficult, as the agent generates force and moment commands and the vehicle response is affected by inertia, damping, nonlinear coupling, and underactuation. These simulations were useful in the early stage of this project. They helped build experience with Simulink reinforcement learning environments, observation design, action scaling, reward shaping, and training behavior.

However, the DDPG approach also introduced several practical difficulties. First, the training process may require many episodes before the agent finds useful control behavior. Second, the reward function has a strong influence on the final result, and small changes in reward weights may change the learned policy. Third, the trained policy is represented

by neural networks, so it is not always easy to explain the controller from a traditional control-theoretic point of view.

Compared with the DDPG baseline, the ADP framework used has a clearer connection with value functions, cost functions, policy evaluation, policy improvement, and the HJB-based optimal control structure. While DDPG and the ADP controller are both learning-related control methods, they are used differently. DDPG is a model-free actor-critic reinforcement learning method, which learns a control policy through repeated interaction with the Simulink environment. The quality of the learned policy depends on the reward function, exploration noise, neural network structure, and training process. The ADP-based controller follows a more structured design. The controller combines a baseline controller with an ADP-based input derived from an approximate value function. This structure makes it easier to explain how the control input is generated and how the weighting parameter λ changes the controller behavior.

The purpose of including DDPG is not to claim that one method is always better than the other. The DDPG results provide useful background and a preliminary baseline from the earlier stage of the project. The main focus of this thesis remains the design and evaluation of the ADP-based nonlinear controller.

Bibliography

[1] B. Wang, *Cooperative Control for Heterogeneous Underactuated Autonomous Vehicle Networks*. Villanova University, 2022.

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

[3] B. Wang, T. Han, and G. Wang, "Further results on safety-critical stabilization of force-controlled nonholonomic mobile robots," *ASME Letters in Dynamic Systems and Control*, vol. 6, no. 2, p. 021011, 2026.

[4] C. C. de Wit, H. Khennouf, C. Samson, and O. J. Sordalen, "Nonlinear control design for mobile robots," in *Recent Trends in Mobile Robots*. World Scientific, 1993, ch. 5.

[5] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[6] H. Modares and F. L. Lewis, "Linear quadratic tracking control of partially-unknown continuous-time systems using reinforcement learning," *IEEE Transactions on Automatic Control*, vol. 59, no. 11, pp. 3051–3056, 11 2014.

[7] B. Wang, S. G. Nersesov, and H. Ashrafiuon, "Time-varying formation control for heterogeneous planar underactuated multivehicle systems," *Journal of Dynamic Systems, Measurement, and Control*, vol. 144, no. 4, p. 041006, 2022.

[8] V. T. Vu, Q. H. Tran, T. L. Pham, and P. N. Dao, "Online actor-critic reinforcement learning control for uncertain surface vessel systems with external disturbances," *International Journal of Control, Automation and Systems*, vol. 20, no. 3, pp. 1029–1040, 3 2022.

[9] B. Wang, S. Nersesov, H. Ashrafiuon, P. Naseradinmousavi, and M. Krstić, "Underactuated source seeking by surge force tuning: Theory and boat experiments," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 4, pp. 1649–1662, 7 2023.

[10] J. Woo, C. Yu, and N. Kim, "Deep reinforcement learning-based controller for path following of an unmanned surface vehicle," *Ocean Engineering*, vol. 183, pp. 155–166, 7 2019.

[11] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE Circuits and Systems Magazine*, vol. 9, no. 3, pp. 32–50, 2009.

[12] Y. Jiang and Z.-P. Jiang, *Robust Adaptive Dynamic Programming*. Hoboken, NJ: John Wiley & Sons, 2017.

[13] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2042–2062, 6 2018.

[14] J. Si and Y.-T. Wang, "On-line learning control by association and reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, 2001.

[15] B. Luo, Y. Yang, H.-N. Wu, and T. Huang, "Balancing value and policy iteration for discrete-time control," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, 2020.

[16] E. Pashenkova, I. Rish, and R. Dechter, "Value iteration and policy iteration algorithms for markov decision problem," 1996.

[17] B. Wang, S. G. Nersesov, and H. Ashrafiuon, "Robust formation control and obstacle avoidance for heterogeneous underactuated surface vessel networks," *IEEE Transactions on Control of Network Systems*, vol. 9, no. 1, pp. 125–137, 3 2022.

[18] B. Wang, H. Ashrafiuon, and S. Nersesov, "Leader–follower formation stabilization and tracking control for heterogeneous planar underactuated vehicle networks," *Systems & Control Letters*, vol. 156, p. 105008, 2021.

[19] H. Tan, "Reinforcement learning with deep deterministic policy gradient," *International Conference on Artificial Intelligence, Big Data and Algorithms*, 2021.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, Massachusetts and London, England: The MIT Press, 2018, a Bradford Book.

[21] B. Ma, Z. Liu, Q. Dang, W. Zhao, J. Wang, Y. Cheng, and Z. Yuan, "Deep reinforcement learning of uav tracking control under wind disturbances environments," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, 2023.

[22] B. Recht, "A tour of reinforcement learning: The view from continuous control," 6 2018. [Online]. Available: http://arxiv.org/abs/1806.09460

[23] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.