Learning-Based Control for Mobility Systems

Approximate Dynamic Programming, Reinforcement Learning, and Online Lookahead

Kyunghwan Choi
Korea Advanced Institute of Science and Technology

Working draft 0.6.2-draft
July 2026

Version: 0.6.2-draft (July 2026).

Preface

Learning-based control is often introduced through a sequence of algorithms. This book takes a different route. It begins with an optimal control problem and asks why its nominal solution may be unusable. Two difficulties recur in mobility systems. The first is *computational*: even with a known model, exact dynamic programming may require more state, action, or online computation than is available. The second is *informational*: the model may be incomplete, uncertain, changing, or learned from data. These difficulties can occur together, but they are not the same, and a useful controller must make clear which one it addresses.

Dynamic programming provides the organizing framework. Exact DP supplies the reference equations and policy-improvement logic; approximate DP retains those ideas when exact computation is impractical. The approximation may be placed in value, Q-factor, policy, or model space, and its targets may come from an analytic model, a simulator, or sampled data. Reinforcement learning is treated primarily as methodology for carrying out these approximate evaluations and improvements. This is a control-oriented lens for asking what a method approximates and how the result enters a closed loop, not a claim that every use of the term *reinforcement learning* is identical to approximate DP.

The central thread is the connection between *offline approximation* and *online improvement*. A learned object need not be the final controller. An actor can be a base policy or optimizer warm start, a critic can be a terminal or leaf value, and a learned model can supply short-horizon predictions. Lookahead, rollout, and model predictive control can then use the current state, constraints, and an online computational budget to improve the action. Linear–quadratic control and MPC are therefore a solvable laboratory, not a detour: they make Bellman and Riccati iterations, terminal costs, and online policy improvement visible.

Representative modern RL algorithms enter where their approximation mechanisms belong. DQN accompanies approximate Q-value iteration, while PPO, TD3, SAC, and related actor–critic methods accompany approximation in policy space. They are compared through policy evaluation, policy improvement, data use, and execution-time computation rather than presented as an isolated catalogue. Mobility examples likewise emphasize more than fitting loss or training return: they use nonlearning baselines, physical and control metrics, shifted conditions, constraint behavior, computation time, and reproducible evidence. Guarantees are stated with their assumptions, and numerical observations and engineering recommendations are identified as such.

The intended readers are graduate students and researchers entering from control, mobility systems, optimization, or AI; prior RL coursework is not required. The text uses state, control, and cost, with bridges to reward-based AI terminology. Generative AI assisted with outlines, early drafts, code scaffolding, and editing. The author sets the structure, verifies sources, derivations, and computations, revises the text, and assumes responsibility for its content.

How to Read This Book

The chapters are arranged as a path from optimal-control formulation to approximation and then to execution. The main question is not simply "Which algorithm is used?" It is "What is approximated, how is it learned or computed, and what does the controller do online?" The following reading lens is used throughout the book.

Five Questions to Ask

For each method, example, or software implementation, identify five items.

1. **Decision problem.** What are the state, control, dynamics, cost, constraints, horizon, and information available to the controller?
2. **DP object.** Is the method approximating a value function, Q-factor, policy, model, or more than one of these objects?
3. **Target and data.** Is a Bellman quantity evaluated from an explicit model, generated by a simulator, or estimated from sampled transitions? Is the data on-policy or off-policy?
4. **Policy improvement and execution.** Is the deployed action a direct output of an offline policy, a minimization over a learned critic, or the result of online lookahead, rollout, or MPC? What model and computational budget does this online step use?
5. **Evidence.** Which assumptions support a guarantee, and which conclusions are numerical? Are comparisons made with credible baselines, shifted conditions, physical metrics, constraint behavior, and computation time?

These questions separate several ideas that are easily conflated. A neural network does not make an update model-free. A sampled Bellman target may be generated by a known simulator. A policy trained offline may be executed directly or used only to initialize an online optimizer. Conversely, an online computation need not be online learning: MPC may improve the current action without changing any learned parameter.

Conventions and Terminology

The main text minimizes cost and uses state x, control u, stage cost g, and policy μ. Much of the AI literature maximizes reward and uses state s, action a, reward r, and policy π. Terminology boxes translate between the two conventions; signs must be checked when a reward-based software implementation is compared with a cost-based derivation.

Finite state and action sets are exact reference settings, not definitions of DP. When a mobility example has continuous variables, the text states whether it uses discretization, analytic minimization, continuous online optimization, or a parameterized continuous actor. Similarly, *model-based* and *model-free* are not used as single-axis labels. The text distinguishes the origin and online use of a model from whether Bellman quantities are computed analytically or estimated through samples.

Four types of statement also appear and should be read differently: theorems and propositions hold under their stated assumptions; derivations establish identities within the adopted model; computational observations describe a particular experiment; and design guidance records an engineering recommendation rather than a universal guarantee.

Chapter Map

Chapters 1–3 establish the vocabulary and machinery. Chapter 1 separates computational complexity from model uncertainty, Chapter 2 develops finite- and infinite-horizon exact and approximate DP, and Chapter 3 introduces parameterized values, policies, and models while distinguishing fitting quality from control relevance.

Chapters 4–6 form the core DP-to-control sequence. Chapter 4 develops value- space approximation, sample-based policy evaluation, approximate policy iteration, Q-learning, DQN variants, and rollout. Chapter 5 uses LQR and MPC to make DP geometry, terminal approximation, and online lookahead explicit. Chapter 6 derives policy-gradient and actor–critic methods before interpreting PPO, TD3, SAC, CEM, and actor- or critic-assisted online improvement.

Chapters 7–9 move from learning dynamics and uncertainty to control with learned models, stability and safety claims, and online adaptation. Chapter 10 addresses multiagent information and coordination; Chapter 11 integrates the pieces in mobility case studies. The appendices provide mathematical background, additional finite-horizon methods, and compact modern-RL concept guides that complement Chapters 4 and 6.

Suggested Reading Routes

Full course route. Read Chapters 1–6 in order, then Chapters 7–11 to preserve the progression from exact DP to approximation, online improvement, models, and deployment.

Reader entering from control. Use Chapters 2 and 5 to connect Bellman and Riccati ideas, then read Chapters 3, 4, and 6 for values, critics, and actors. Consult the modern-RL appendix when an algorithm name is unfamiliar.

Reader entering from AI or reinforcement learning. Begin with Chapters 1 and 2 for the cost-minimization and optimal-control formulation, then read Chapters 4 and 6. Use Chapter 5 for terminal costs, base policies, and online control, and Chapters 7–9 for models and safety.

Project-oriented route. Start with the relevant mobility case, but trace every learned value, policy, or model back to Chapters 2–6. Report baselines, information assumptions, offline and online computation, shifted-condition tests, and reproducible code.

Contents

Preface iii

How to Read This Book iv

1 Why Learning-Based Control? 1

2 Exact and Approximate Dynamic Programming 2
2.1 Why Dynamic Programming Comes First . . . 2
2.2 Deterministic Finite-Horizon Optimal Control . . . 3
2.3 Stochastic Finite-Horizon Dynamic Programming . . . 6
2.4 From Exact to Approximate Finite-Horizon DP . . . 8
2.4.1 Approximate values as terminal evaluators . . . 8
2.4.2 Fitted backward recursion . . . 8
2.4.3 Decision pipeline and sources of approximation . . . 9
2.5 Mobility Case: Finite-State HEV Energy Management . . . 10
2.6 Stationary Infinite-Horizon Problems . . . 15
2.7 Finite Tables and Continuous Control Spaces . . . 17
2.8 Exact and Approximate Value and Policy Iteration . . . 18
2.8.1 Exact value iteration . . . 19
2.8.2 Approximate value iteration . . . 20
2.8.3 Exact policy iteration . . . 20
2.8.4 Approximate policy iteration . . . 21
2.8.5 Q-factor value iteration and policy iteration . . . 22
2.8.6 Modified and optimistic policy iteration . . . 22
2.8.7 Multistep policy improvement: a geometric preview . . . 23
2.9 Mobility Case: A Stationary Stochastic Lane-Keeping MDP . . . 23
2.10 From Q-Value Iteration to Q-Learning . . . 25
2.11 Finite-to-Infinite Reformulation . . . 26
2.11.1 Can a slalom task be treated as infinite horizon? . . . 27
2.12 Approximation, Model Use, and Implementation . . . 27
2.12.1 Three levels of exactness . . . 28
2.12.2 Where approximation can enter . . . 28
2.12.3 Model use is not a single binary label . . . 29
2.13 Before Applying Dynamic Programming . . . 30
2.14 Summary . . . 30
2.15 Exercises . . . 31

3 Parametric Approximation **34**
3.1 Why Approximation Architecture Comes Before Approximate DP 35
3.2 What Is Being Parameterized? 35
3.2.1 A map of the architectures used in this chapter 36
3.2.2 Finite-action policies as classification 36
3.3 Linear-in-Parameter Feature Architectures 38
3.3.1 Local, global, and structured features 38
3.3.2 Q-factor and continuous-policy features 40
3.4 Fitting Architectures to a Fixed Target 41
3.4.1 Weighted and regularized least squares 41
3.4.2 What regularization selects—and what it does not fix 42
3.4.3 Stochastic labels and repeated targets 43
3.4.4 Target error and fitting error are distinct 44
3.5 Nonlinear and Neural Architectures 44
3.5.1 Loss, backpropagation, and parameter updates 46
3.5.2 Output structure should match the control object 47
3.6 Data Distribution and Control-Relevant Validation 49
3.6.1 Random rows are not always independent validation data 49
3.6.2 Small value error and correct action are different claims 49
3.7 Mobility Case: A Car-Following Value Surrogate 51
3.7.1 Declared model and reference target 51
3.7.2 Architectures and data support 52
3.7.3 Prediction quality and decision quality 54
3.8 From Fixed Targets to Approximate DP 56
3.9 Evaluation Before Controller Deployment 57
3.10 Summary 57
3.11 Exercises 58

4 Approximation in Value Space **60**
4.1 From an Approximation Architecture to an Approximate DP Method 61
4.2 Fitted Value Iteration: Bellman Backup Followed by Projection 63
4.2.1 The generic fitted-VI step 63
4.2.2 Approximate Q-value iteration 64
4.2.3 How one-step approximation errors accumulate 64
4.3 Policy Evaluation from Sampled Trajectories 65
4.3.1 Monte Carlo returns 65
4.3.2 One-step temporal difference 66
4.3.3 The continuum between MC and TD(0) 67
4.3.4 Linear TD, LSTD, and projected equations 68
4.4 Approximate Policy Iteration 69
4.4.1 Evaluation error, improvement error, and the API error region 70
4.4.2 Why API is not merely fitted VI with different labels 71
4.4.3 The geometric connection to policy iteration and online play 71

4.4.4 Optimistic and simulation-based API . . . 72
4.5 Sampled Q Updates: Q-Learning and SARSA . . . 73
4.5.1 What tabular convergence does and does not say . . . 74
4.6 Deep Q-Networks as Stabilized Fitted Q Iteration . . . 74
4.6.1 Double and dueling modifications . . . 75
4.7 Online Improvement with a Terminal Value . . . 76
4.7.1 One horizon notation for lookahead, rollout, and a terminal critic . . . 77
4.7.2 Rollout as policy improvement . . . 79
4.8 Mobility Case: A Common Lane-Keeping Comparison . . . 79
4.8.1 Six computations, one declared model and data source . . . 80
4.8.2 What the numerical result establishes . . . 81
4.9 A Linear-Programming View of Value Approximation . . . 85
4.10 Value Space and Policy Space Are Complementary Axes . . . 86
4.11 Conditions, Failure Modes, and Deployment Checks . . . 87
4.12 Chapter Summary . . . 88
4.13 Exercises . . . 89

5 LQR and MPC Through the DP Lens 91
5.1 Why Return to a Linear–Quadratic Problem? . . . 92
5.2 Finite-Horizon LQR Is Backward Dynamic Programming . . . 92
5.3 Infinite-Horizon LQR as a Fixed-Point Problem . . . 93
5.4 Terminal Values and Finite Lookahead . . . 94
5.5 Approximate Terminal Values and Their Induced Policies . . . 95
5.6 Geometric View of Policy Improvement and Lookahead . . . 96
5.6.1 The Bellman envelope and a touching policy operator . . . 96
5.6.2 VI, PI, and rollout are different moves . . . 97
5.6.3 Multistep lookahead moves the Newton starting point . . . 98
5.7 Scalar LQR: Riccati Iteration and Lookahead . . . 99
5.7.1 Riccati geometry of value and policy iteration . . . 100
5.8 Vehicle Example: Linearized Lateral-Error Regulation . . . 102
5.9 From Receding-Horizon LQR to Constrained MPC . . . 103
5.9.1 Terminal ingredients and the shift argument . . . 105
5.10 Terminology Across DP, MPC, and Model-Based RL . . . 106
5.11 Design Conditions and Limitations . . . 107
5.12 Reproducible Implementation Pattern . . . 107
5.13 Summary . . . 108
5.14 Exercises . . . 109

6 Approximation in Policy Space 111
6.1 Why Approximate a Policy Directly? . . . 112
6.2 Deterministic and Stochastic Policy Architectures . . . 113
6.2.1 Policy fitting from action labels . . . 114
6.2.2 Rollout and optimizer distillation . . . 114

6.3 Policy Objectives and the Policy-Gradient Identity . . . 115
6.3.1 Deterministic policy gradient . . . 117
6.3.2 Performance difference and the meaning of improvement . . . 117
6.4 Actor–Critic as Approximate Policy Iteration . . . 119
6.4.1 From TD errors to generalized advantage estimates . . . 119
6.4.2 Which distribution fits the critic? . . . 120
6.5 PPO as Restricted On-Policy Improvement . . . 122
6.6 DDPG and TD3: Deterministic Continuous-Action API . . . 123
6.7 SAC as Entropy-Regularized Policy Iteration . . . 124
6.8 Derivative-Free Search in Structured Policy Classes . . . 125
6.9 One Comparison Map for Modern Algorithms . . . 127
6.10 The Learned Actor Is a Starting Point for Online Improvement . . . 129
6.10.1 Actor as a base policy for rollout . . . 129
6.10.2 Actor as a warm start or local search center . . . 129
6.10.3 Critic as a terminal value . . . 130
6.10.4 From three mechanisms to five deployment patterns . . . 130
6.11 Mobility Case: A Policy Actor with Online Lookahead . . . 133
6.11.1 Offline actor search and critic fitting . . . 133
6.11.2 Actor-centered lookahead . . . 135
6.11.3 What is model based and what is sample based? . . . 135
6.12 Design and Deployment Checklist . . . 137
6.13 Chapter Summary . . . 138
6.14 Exercises . . . 138

7 Learning Dynamics and Uncertainty 141

8 Control with Learned Models 142

9 Stability, Safety, and Online Adaptation 143

10 Multiagent Decision and Coordination 144

11 Integrated Mobility Case Studies and Outlook 145

A Mathematical Background 146

B Additional Finite-Horizon Approximate DP Methods 147

C Modern RL Algorithm Bridge and Extended Models 148
C.1 A Common Actor–Critic Template . . . 148
C.2 DQN: A Critic That Also Defines a Discrete Actor . . . 148
C.3 PPO: On-Policy Improvement with a Restricted Ratio Step . . . 150
C.4 TD3: Deterministic Continuous-Action API . . . 150
C.5 SAC: Entropy-Regularized Soft Policy Iteration . . . 151

C.6 Reading the Algorithms Through Three Questions 152
C.7 Other Extended DP Models 152

Bibliography **153**

1 Why Learning-Based Control?

Planned role

This chapter motivates learning-based control as approximate optimal control under computational and model uncertainty, and fixes the terminology used by the rest of the book.

Planned contents

- Mobility control problems and performance opportunities
- Optimal-control problem formulation and the solution-method map
- Computational complexity versus model uncertainty
- Offline training, online planning/play, and online learning
- Control, operations-research, and AI terminology
- Two meanings of model-based and model-free implementation

2 Exact and Approximate Dynamic Programming

Learning objectives

After studying this chapter, the reader should be able to:

1. formulate deterministic and stochastic finite-horizon control problems;
2. derive Bellman's backward recursion from the principle of optimality;
3. distinguish a time-dependent Markov policy from a stationary policy;
4. construct deterministic and stochastic approximate-DP policies using an approximate value function;
5. formulate exact and approximate DP with values and Q-factors;
6. explain when a tabular controller requires state/action discretization and when continuous controls can be retained;
7. define Bellman and policy-evaluation operators for a discounted problem;
8. compare value- and Q-factor forms of exact and approximate value iteration, policy iteration, and optimistic policy iteration;
9. identify which methods require an explicit model and backups over all states; and
10. reformulate a finite-horizon problem as an episodic infinite-horizon problem without claiming a computational shortcut.

Chapter roadmap

The chapter develops one DP principle at two time scales. It first derives finite-horizon backward recursion, including stochastic problems and an HEV case. It then moves to stationary discounted problems, where Bellman operators lead to VI, PI, Q-factor methods, and their approximate versions. The final sections connect finite and infinite horizons and make model use, discretization, and approximation explicit.

2.1 Why Dynamic Programming Comes First

A mobility controller rarely makes one isolated choice. A battery decision changes the energy available later, steering changes the vehicle state seen by the next controller update, and a lane-

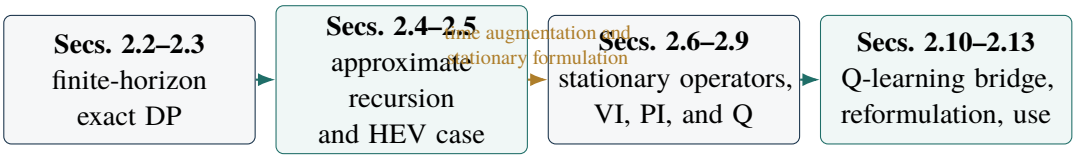


Figure 2.1: Original map of Chapter 2. Exact equations establish the reference; approximation changes their representation or computation, not the underlying optimal-control question.

change decision changes the future interaction geometry. Dynamic programming (DP) starts from this coupling. It assigns a value to every relevant tail problem and uses that value to compare actions whose immediate costs may look similar but whose consequences differ.

The central equation is simple enough to be misleading:

$$\text{optimal value now} = \min_{\text{current control}} \{\text{current cost} + \text{optimal value next}\}. \tag{2.1}$$

The equation can be mathematically exact while being impossible to evaluate at every continuous state. This chapter therefore keeps two questions separate:

1. *Characterization:* what equation does an optimal value function satisfy?
2. *Computation:* how is that equation represented and evaluated in a particular implementation?

The first question gives a reference point for optimal control. The second leads to discretization, approximate dynamic programming, reinforcement learning, and online planning. The finite-horizon development follows the optimal-control viewpoint in Bertsekas (2019) and Bertsekas (2025); the discounted finite-state development is also consistent with the MDP treatment in Sutton and Barto (2018). The derivations, teaching example, and numerical figures in this chapter are written independently.

Control viewpoint

DP is not one solver. It is a decomposition principle. Tabular backward recursion, Riccati recursion, value iteration, policy iteration, rollout, MPC, and actor–critic methods use different representations and computations, but each can be read in terms of evaluation and improvement of future cost.

2.2 Deterministic Finite-Horizon Optimal Control

Consider the system

$$x_{k+1} = f_k(x_k, u_k), \qquad k = 0, \dots, N-1, \tag{2.2}$$

where $x_k \in \mathcal{X}_k$ is the state and $u_k \in \mathcal{U}_k(x_k)$ is an admissible control. For a prescribed initial state x_0, the cost of a control sequence is

$$J_0(x_0; u_0, \dots, u_{N-1}) = \sum_{k=0}^{N-1} g_k(x_k, u_k) + g_N(x_N). \tag{2.3}$$

The terminal cost g_N is part of the problem specification. It may encode a desired final state, a residual value, a soft terminal penalty, or a hard terminal requirement through the extended value $+\infty$.

At stage k, define the tail value

$$J_k^*(x) = \min_{u_k,\dots,u_{N-1}} \left\{ \sum_{t=k}^{N-1} g_t(x_t, u_t) + g_N(x_N) \right\}, \qquad x_k = x, \tag{2.4}$$

where the minimization is restricted to controls satisfying the dynamics and constraints. A finite-horizon state-feedback policy is a sequence $\mu = (\mu_0, \dots, \mu_{N-1})$ with $\mu_k(x) \in \mathcal{U}_k(x)$. It is generally time dependent: the action can depend on both the current state and the stage.

Terminology note

A policy $u_k = \mu_k(x_k)$ is often called a *nonstationary Markov policy*. "Markov" means that the current state is a sufficient summary of the past; "stationary" means that the same mapping μ is used at every stage. Thus a policy can be Markov and nonstationary. Equivalently, it becomes stationary on the augmented state $z_k = (k, x_k)$.

Assumption 2.1 (Finite deterministic problem)**.** For each k and $x \in \mathcal{X}_k$, the feasible action set $\mathcal{U}_k(x)$ is nonempty, all costs used in a feasible comparison are well defined in the extended real line, and the displayed minima are attained. A value $+\infty$ may encode infeasibility, provided the states of interest have at least one finite-cost continuation. Finite state and action sets are a sufficient setting for this chapter's result.

Proposition 2.1 (Principle of optimality)**.** *Suppose assumption 2.1 holds. If a control sequence is optimal from x_0 and reaches x_k^* at stage k, its remaining controls are optimal for the tail problem starting from (k, x_k^*).*

Proof. If the remaining controls were not optimal for the tail problem, replace them with a feasible tail of strictly smaller cost. The prefix through x_k^* is unchanged, so the replacement would produce a feasible full sequence with strictly smaller total cost. This contradicts optimality of the original sequence. □

The proposition concerns tails along an optimal trajectory. The DP recursion goes further: it solves the tail problem from every state in the declared state space.

Theorem 2.1 (Deterministic finite-horizon Bellman recursion). *Under assumption 2.1, initialize*

$$J_N^*(x) = g_N(x). \tag{2.5}$$

Then, for $k = N-1, \dots, 0$,

$$J_k^*(x) = \min_{u \in \mathcal{U}_k(x)} \left\{ g_k(x,u) + J_{k+1}^*\big(f_k(x,u)\big) \right\}. \tag{2.6}$$

Any selector

$$\mu_k^*(x) \in \arg\min_{u \in \mathcal{U}_k(x)} \left\{ g_k(x,u) + J_{k+1}^*\big(f_k(x,u)\big) \right\} \tag{2.7}$$

defines an optimal time-dependent state-feedback policy.

Proof. At $k = N$, no control remains, so Equation (2.5) holds. Suppose J_{k+1}^* is the optimal value of every stage-$(k+1)$ tail problem. If the first control at stage k is u, the immediate cost is $g_k(x,u)$ and the best possible remaining cost is $J_{k+1}^*(f_k(x,u))$. Minimizing their sum over feasible u gives Equation (2.6). Backward induction establishes the value recursion. Choosing a minimizing control at each stage attains the recursively computed value, so the resulting policy is optimal. □

The same exact recursion can be written with a state–control value, or *Q-factor*. For the deterministic problem, define

$$Q_k^*(x,u) = g_k(x,u) + J_{k+1}^*\big(f_k(x,u)\big). \tag{2.8}$$

Then

$$J_k^*(x) = \min_u Q_k^*(x,u), \qquad \mu_k^*(x) \in \arg\min_u Q_k^*(x,u). \tag{2.9}$$

For $k \le N-2$, eliminating J_{k+1}^* gives the Q-factor-only recursion

$$Q_k^*(x,u) = g_k(x,u) + \min_v Q_{k+1}^*\big(f_k(x,u), v\big). \tag{2.10}$$

Thus J_k^* ranks states before the current action is selected, whereas Q_k^* ranks a particular current action together with its best continuation.

Algorithm lens: Backward recursion, forward execution

For a finite-state implementation:

1. set $J_N(x) = g_N(x)$ for every terminal state;
2. for $k = N-1, \dots, 0$, evaluate every feasible state–control pair, store the minimum as $J_k(x)$, and store a minimizing action as $\mu_k(x)$;
3. starting from x_0, apply $u_k = \mu_k(x_k)$ and propagate the system forward.

The backward pass is an offline computation when the model and problem data are known. The forward pass is policy execution. Recomputing a tail problem from the current state

instead would be an online planning method.

If stage k has n_k states and at most m_k controls per state, direct tabulation requires on the order of $\sum_{k=0}^{N-1} n_k m_k$ one-step comparisons. This can be entirely practical for a small graph and entirely impractical for a finely discretized vehicle. The theorem is not weakened by the latter fact; the representation is the bottleneck.

2.3 Stochastic Finite-Horizon Dynamic Programming

Now let the next state depend on a disturbance:

$$x_{k+1} = f_k(x_k, u_k, w_k). \tag{2.11}$$

The controller selects u_k after observing x_k but before observing w_k. This timing is essential. A policy that uses the value of w_k before the disturbance is revealed solves a different problem.

Let $h_k = (x_0, u_0, \dots, x_k)$ denote the observed history. The controlled Markov property is

$$\mathbb{P}(x_{k+1} = x' \mid h_k, u_k = u) = p_k(x' \mid x_k, u). \tag{2.12}$$

The subscript k allows a time-varying, or time-inhomogeneous, Markov model. Markov therefore does not imply stationary.

For finite state and control sets, write the controlled transition probability as

$$p_k(x' \mid x, u) = \mathbb{P}(x_{k+1} = x' \mid x_k = x, u_k = u). \tag{2.13}$$

The expected cost of a policy μ is

$$J_{0,\mu}(x_0) = \mathbb{E}_\mu\left[\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \,\middle|\, x_0\right]. \tag{2.14}$$

Assumption 2.2 (Finite stochastic control model)**.** The state is Markov under each admissible control, the state and control sets are finite, feasible action sets are nonempty, and the expected one-stage and terminal costs are finite. The disturbance law, or equivalently the transition kernel and conditional expected stage cost, is known for the exact recursion.

For compact notation, let

$$\bar{g}_k(x, u) = \mathbb{E}\left[g_k(x, u, w_k) \mid x_k = x, u_k = u\right]. \tag{2.15}$$

Theorem 2.2 (Stochastic finite-horizon Bellman recursion)**.** *Under assumption 2.2, an optimal*

value satisfies

$$J_N^*(x) = g_N(x), \tag{2.16}$$

$$J_k^*(x) = \min_{u \in \mathcal{U}_k(x)} \left\{ \bar{g}_k(x,u) + \sum_{x' \in \mathcal{X}_{k+1}} p_k(x' \mid x,u) J_{k+1}^*(x') \right\}. \tag{2.17}$$

An optimal policy can be chosen as a deterministic sequence $u_k = \mu_k^*(x_k)$ within this finite model.

Proof. Condition on the currently observed state and on the first selected action. The Markov property makes the conditional distribution of the next state depend only on (x,u). The conditional expected cost after the transition is therefore the probability-weighted value of the next tail problem. Minimizing the sum of present and future expected cost gives Equation (2.17). Backward induction proves the recursion. At each finite minimization, a deterministic minimizing action exists; selecting one at every state and stage gives an optimal Markov policy. □

The stochastic exact recursion has the parallel Q-factor form

$$Q_k^*(x,u) = \bar{g}_k(x,u) + \sum_{x'} p_k(x' \mid x,u) J_{k+1}^*(x'), \tag{2.18}$$

$$J_k^*(x) = \min_u Q_k^*(x,u), \qquad \mu_k^*(x) \in \arg\min_u Q_k^*(x,u). \tag{2.19}$$

Substituting $J_{k+1}^*(x') = \min_v Q_{k+1}^*(x',v)$ yields

$$Q_k^*(x,u) = \bar{g}_k(x,u) + \sum_{x'} p_k(x' \mid x,u) \min_v Q_{k+1}^*(x',v). \tag{2.20}$$

The terminal boundary is inherited from g_N. The Q-factor therefore retains the uncertainty through the transition-weighted continuation cost; it is not a deterministic substitute for stochastic DP. It should also not be confused with the state-weight matrix Q used later in LQR.

AI/RL terminology bridge

Writing a Bellman equation in Q-factor form does *not* by itself make the method model free. Exact evaluation of the preceding recursion still uses f_k or the full kernel p_k. The model-free possibility arises because a Q-factor can instead be estimated from sampled transitions, and a stored $\widetilde{Q}_k(x,u)$ then selects an action through $\arg\min_u \widetilde{Q}_k(x,u)$ without calling an explicit model at execution. In AI notation, x, u, μ, α are commonly written s, a, π, γ, and reward is maximized with $r_k = -g_k$. Chapter 4 develops the sample-based algorithms; here the important distinction is between the *object* Q and the *way it is computed*.

The finite-set assumption supports the elementary recursion and an exact table. A continuous-state or continuous-control formulation still has a Bellman equation. In the stochastic finite-horizon case it takes the form

$$J_k^*(x) = \inf_{u \in \mathcal{U}_k(x)} \mathbb{E}\left[g_k(x,u,w_k) + J_{k+1}^*(f_k(x,u,w_k)) \,\middle|\, x,u \right]. \tag{2.21}$$

The infimum replaces a finite minimum when a minimizing action is not known to exist, and the expectation may be an integral over a continuous disturbance or next state. General-state results require measurability, integrability, and measurable-selection assumptions. This chapter deliberately proves the finite-set case; Section 2.7 explains how that reference model connects to continuous mobility control.

2.4 From Exact to Approximate Finite-Horizon DP

Exact backward DP computes $J_k^*(x)$ for every state and stage before the forward policy in Equation (2.7) or Equation (2.17) can be executed. This requirement becomes the computational bottleneck when the state space is large, continuous, or only accessible through data. Approximate DP replaces one or more exact components of the backup by tractable approximations. Approximation may be introduced in the stored value, in the Q-factor, in the expectation, or in the minimization. These choices have different online consequences, so they should be stated separately.

2.4.1 Approximate values as terminal evaluators

For the deterministic problem, an approximate value $\tilde{J}_{k+1}$ defines the one-step lookahead policy

$$\widetilde{\mu}_k(x) \in \arg\min_{u \in \mathscr{U}_k(x)} \left\{ g_k(x,u) + \tilde{J}_{k+1}(f_k(x,u)) \right\}. \tag{2.22}$$

For the stochastic problem, the corresponding policy is

$$\widetilde{Q}_k(x,u) = \mathbb{E}\left[g_k(x,u,w_k) + \tilde{J}_{k+1}(f_k(x,u,w_k)) \,\middle|\, x_k = x, u_k = u \right], \tag{2.23}$$

$$\widetilde{\mu}_k(x) \in \arg\min_{u \in \mathscr{U}_k(x)} \widetilde{Q}_k(x,u). \tag{2.24}$$

Thus stochastic approximate DP is not obtained by deleting uncertainty. It still evaluates the distribution of next states, but may approximate the future value, the expectation, the model, or the minimization. A stored approximate value is an evaluator, not yet a complete controller: the policies in Equations (2.22) and (2.24) still require a one-step model and a minimization when an action is selected.

2.4.2 Fitted backward recursion

Suppose $\tilde{J}_k(x;\theta_k)$ belongs to a tractable parameterized class. A fitted backward pass begins from a chosen terminal approximation and, for $k = N-1, \dots, 0$, constructs targets at training states. The deterministic and stochastic targets are

$$y_k^{\mathrm{det}}(x) = \min_u \left\{ g_k(x,u) + \tilde{J}_{k+1}(f_k(x,u);\theta_{k+1}) \right\}, \tag{2.25}$$

$$y_k^{\mathrm{sto}}(x) = \min_u \mathbb{E}\left[g_k(x,u,w_k) + \tilde{J}_{k+1}(f_k(x,u,w_k);\theta_{k+1}) \,\middle|\, x,u \right]. \tag{2.26}$$

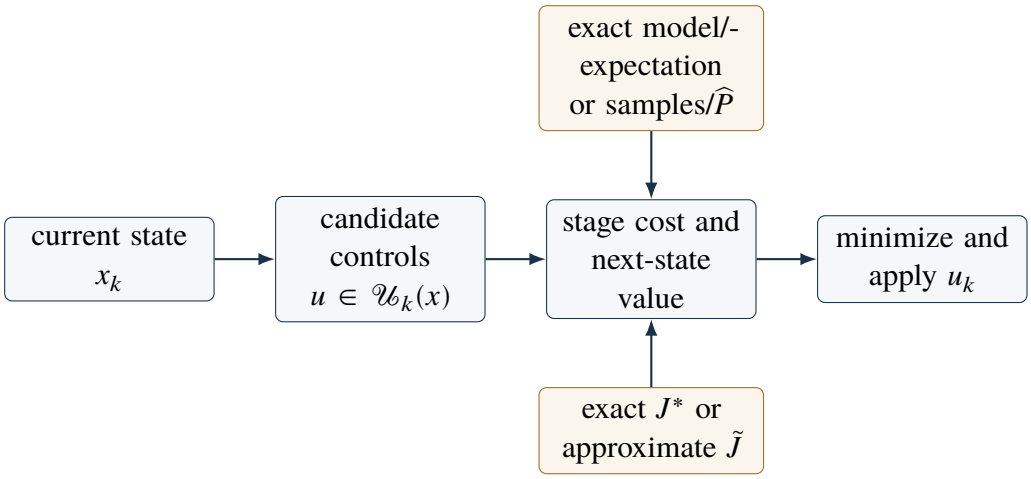


Figure 2.2: Exact and approximate DP share the same decision pipeline. Approximation may enter through the value representation, transition/expectation, candidate controls, or numerical minimization. A new online minimization can improve the action even when the leaf value is approximate.

The parameter θ_k is then fitted so that $\tilde{J}_k(x;\theta_k) \approx y_k(x)$ on the selected states. With a known model, the expectation can be evaluated analytically, by quadrature, or by simulation. With transition data only, samples provide noisy regression targets. Approximation error can therefore enter through state coverage, function class, expectation estimation, and optimization even before the resulting policy is evaluated.

Alternatively, one may fit $\widetilde{Q}_k(x,u)$ directly. The deterministic and stochastic targets are, respectively,

$$y_k^{Q,\mathrm{det}}(x,u) = g_k(x,u) + \min_{v} \widetilde{Q}_{k+1}(f_k(x,u),v), \tag{2.27}$$

$$y_k^{Q,\mathrm{sto}}(x,u) = \mathbb{E}\Big[g_k(x,u,w_k) + \min_{v} \widetilde{Q}_{k+1}(f_k(x,u,w_k),v) \;\Big|\; x,u\Big]. \tag{2.28}$$

These targets may be formed from exact transitions or samples as appropriate. Direct Q-factor fitting removes the explicit one-step model from action selection, although training may still be model based. This is why model use during training and model use during execution must be reported separately.

2.4.3 Decision pipeline and sources of approximation

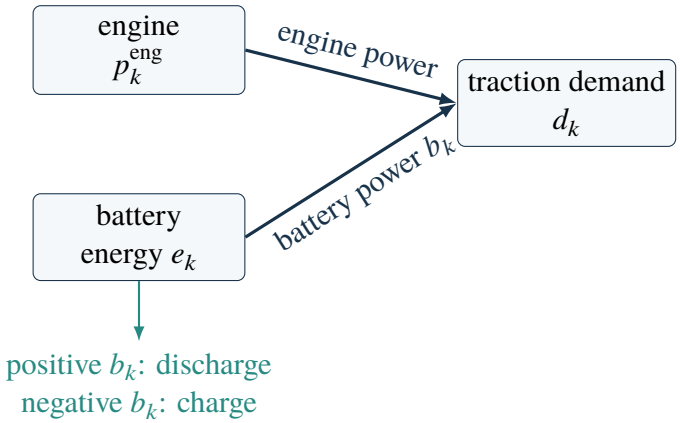


Figure 2.3: Power-flow interpretation of the finite-state HEV example. The known traction demand is split between engine and battery power. Battery power is a control because it changes both the current engine load and the energy available at later stages.

Control viewpoint

There are three distinct products of an approximate backward computation: an approximate value that still needs online lookahead, an approximate Q-factor that supports direct greedy action selection, or a separately fitted policy that directly outputs a control. They share Bellman's logic but have different model and optimization requirements at execution.

2.5 Mobility Case: Finite-State HEV Energy Management

Optimal power split in a hybrid electric vehicle (HEV) is a natural finite- horizon problem: present battery use changes future stored energy, and a known drive cycle creates stage-dependent traction demand. The physical formulation is standard in HEV energy-management treatments such as Onori et al. (2016). The model below is deliberately simplified and normalized: it preserves the power balance, battery-state coupling, local limits, and charge-sustaining terminal requirement, but it is not calibrated to a specific engine, motor, or battery. Its purpose is to expose the DP calculation rather than predict a production vehicle's fuel consumption.

Mobility case: Charge-sustaining power split

Let e_k be battery energy, d_k the known traction demand, b_k battery power, and p_k^{eng} engine power. Positive b_k discharges the battery. The normalized model is

$$p_k^{\text{eng}} = d_k - b_k, \tag{2.29}$$

$$e_{k+1} = e_k - b_k. \tag{2.30}$$

The finite grids and constraints are

$$e_k \in \{0, 1, \dots, 20\}, \qquad b_k \in \{-2, -1, 0, 1, 2\}, \qquad 0 \le p_k^{\mathrm{eng}} \le 8. \tag{2.31}$$

For a 40-stage demand profile, the stage cost is

$$g_k(e_k, b_k) = 0.06(p_k^{\mathrm{eng}})^2 + 0.30 p_k^{\mathrm{eng}} + 0.015 b_k^2. \tag{2.32}$$

The initial and terminal energies are both 10. The exact terminal requirement is represented by

$$g_N(e) = \begin{cases} 0, & e = 10, \\ +\infty, & e \neq 10. \end{cases} \tag{2.33}$$

The known sequence $d_0, \dots, d_{N-1}$ plays the role of a prescribed drive cycle. The constraints in Equation (2.31) are local: they must hold at each stage. The equality $e_N = 10$ is global because a battery decision early in the cycle affects whether the final target remains reachable. It prevents a controller from reporting low fuel use simply by depleting the battery.

The quadratic engine term makes high engine power relatively expensive. The battery can therefore shift engine effort from high-demand stages to low-demand stages, but it must return to its initial energy. Backward DP computes $J_k^*(e)$ and the minimizing battery action for every feasible pair (k, e).

This is a model-based calculation in the DP sense. The controller is given the deterministic energy transition, the entire demand sequence, the stage-cost map, and every constraint before the backward pass. It evaluates every feasible state–action successor on the declared grid; none of these quantities is learned from trajectories in this example.

For comparison, a viability-aware myopic controller minimizes only Equation (2.32). It rejects a control if the successor state cannot reach the terminal target in the remaining stages, so the baseline is feasible, but it ignores the magnitude of future cost. This is a deliberately strong diagnostic: both policies know the constraints and terminal reachability; only DP prices the future. The result also illustrates nonstationarity: at the same battery energy, an optimal action can change with the remaining demand profile and the distance to the terminal stage.

Worked Example 2.1 (One backward Bellman backup)**.** At the last stage, the demand is $d_{39} = 4$. If $e_{39} = 12$, terminal feasibility forces $b_{39} = 2$, so $p_{39}^{\mathrm{eng}} = 2$ and

$$J_{39}^*(12) = 0.06(2)^2 + 0.30(2) + 0.015(2)^2 = 0.90.$$

At $e_{39} = 10$, the only terminal-feasible action is $b_{39} = 0$, giving $J_{39}^*(10) = 2.16$. One stage earlier, with $e_{38} = 10$ and $d_{38} = 4$, Bellman's comparison is

Table 2.1: Verified results for the normalized finite-state HEV grid. Costs are normalized and should be interpreted only within the declared teaching model.

Policy	Total cost	Terminal energy	Cost relative to myopic
Backward DP	62.88	10	−12.34%
Viability-aware myopic	71.73	10	baseline

b_{38}	current cost	$J^*_{39}(e_{39})$	total
−2	4.020	0.900	4.920
−1	3.015	1.455	4.470
0	2.160	2.160	4.320
1	1.455	3.015	4.470
2	0.900	4.020	4.920

The immediate-cost minimum $b_{38} = 2$ is not optimal after the tail cost is included; the Bellman-optimal action is $b_{38} = 0$. This small calculation is the operation performed at every state and stage by the code.

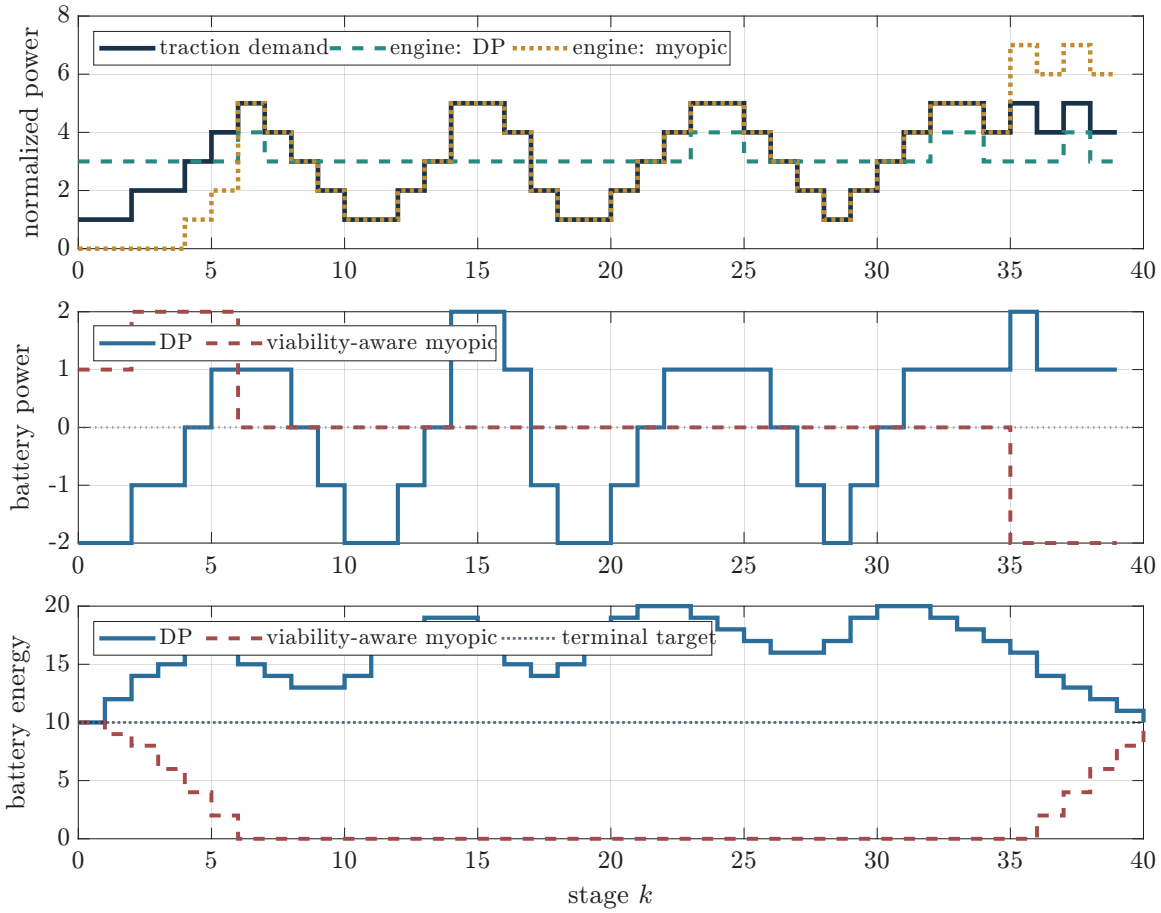


Figure 2.4: Original numerical results for the finite-state HEV problem. DP charges during selected lower-demand intervals and discharges during higher-demand intervals. The myopic controller uses battery energy early and is forced to recharge near the end. Both satisfy the same terminal target.

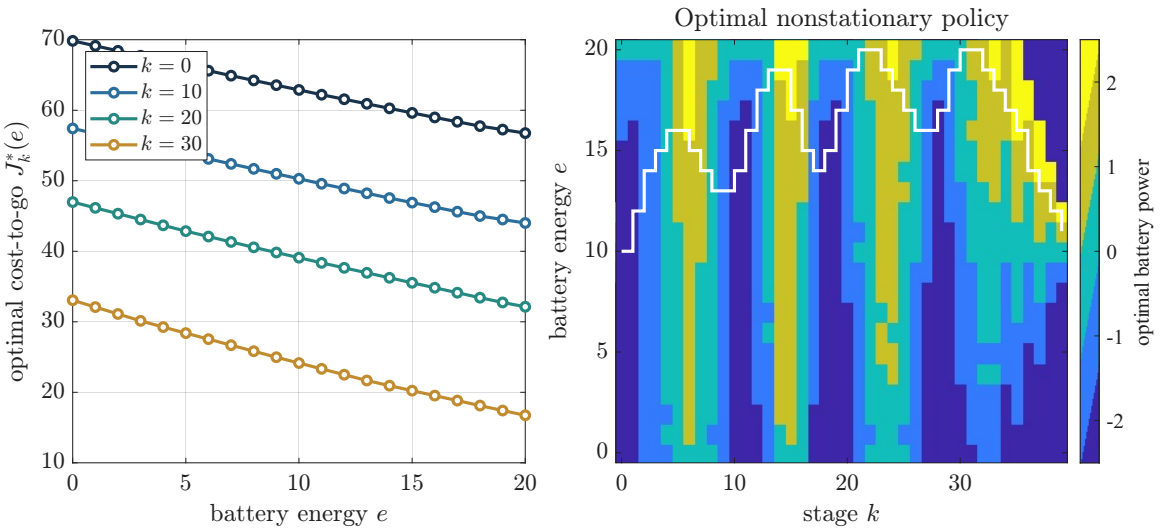


Figure 2.5: Optimal cost-to-go functions at four stages (left) and the complete nonstationary battery policy (right). The white line on the policy map is the optimal trajectory from $e_0 = 10$. The time variation is caused by the demand profile and the shrinking distance to the terminal requirement.

Implementation note

The script `code/ch02_exact_dp/generate_figures.m` uses base MATLAB, sets no random seed because the experiment is deterministic, checks terminal feasibility and Bellman consistency, and exports both figures and CSV data. Its result is exact on the declared finite grid. It is not an exact solution of a continuous or calibrated HEV model.

2.6 Stationary Infinite-Horizon Problems

An infinite horizon does not by itself make a problem stationary. Stationarity requires time-invariant dynamics, cost, constraints, and disturbance law, or a state augmentation that contains their time dependence. We now impose this structure explicitly.

Let $\mathcal{X}$ be a finite state space. At state x, choose $u \in \mathcal{U}(x)$, incur bounded cost $g(x,u)$, and move to x' with probability $p(x' \mid x,u)$. For a stationary policy μ, define the discounted cost

$$J_\mu(x) = \mathbb{E}_\mu\left[\sum_{k=0}^{\infty} \alpha^k g(x_k, \mu(x_k)) \,\middle|\, x_0 = x\right], \qquad 0 < \alpha < 1. \tag{2.34}$$

Assumption 2.3 (Discounted finite MDP)**.** The state and action sets are finite, every $\mathcal{U}(x)$ is nonempty, $|g(x,u)| \le \bar{g} < \infty$, the transition probabilities are stationary, and $0 < \alpha < 1$.

Before introducing operators, write the equations one state at a time. For deterministic dynamics $x^+ = f(x,u)$, policy evaluation and optimality are

$$J_\mu(x) = g(x, \mu(x)) + \alpha J_\mu(f(x, \mu(x))), \tag{2.35}$$

$$J^*(x) = \min_{u \in \mathcal{U}(x)} \{g(x,u) + \alpha J^*(f(x,u))\}. \tag{2.36}$$

For stochastic dynamics, conditioning on the first transition gives

$$J_\mu(x) = g(x, \mu(x)) + \alpha \sum_{x' \in \mathcal{X}} p(x' \mid x, \mu(x)) J_\mu(x'), \tag{2.37}$$

$$J^*(x) = \min_{u \in \mathcal{U}(x)} \left\{ g(x,u) + \alpha \sum_{x' \in \mathcal{X}} p(x' \mid x,u) J^*(x') \right\}. \tag{2.38}$$

The two cases have the same decision structure; the deterministic model has one known successor, whereas the stochastic model averages over successor states.

For any candidate value J, define the corresponding one-step Q-factor by

$$Q_J^{\mathrm{det}}(x,u) = g(x,u) + \alpha J(f(x,u)), \tag{2.39}$$

$$Q_J^{\mathrm{sto}}(x,u) = g(x,u) + \alpha \sum_{x'} p(x' \mid x,u) J(x'). \tag{2.40}$$

The value $J(x)$ answers "how costly is it to start at state x before choosing an action?" The Q-factor $Q_J(x,u)$ answers "how costly is action u now if J evaluates what follows?" Consequently,

$$J^*(x) = \min_u Q_{J^*}(x,u), \qquad Q^*(x,u) = Q_{J^*}(x,u). \tag{2.41}$$

Using J stores one number per state but requires a model-based one-step calculation to compare actions. Using Q stores one number per state–action pair and makes the greedy action directly available once the Q-factor has been computed. The Bellman operator is shorthand for carrying out this comparison at every state.

For a function $J : \mathcal{X} \to \mathbb{R}$, define the Bellman optimality operator

$$(TJ)(x) = \min_{u \in \mathcal{U}(x)} \left\{ g(x,u) + \alpha \sum_{x' \in \mathcal{X}} p(x' \mid x,u) J(x') \right\}, \tag{2.42}$$

and the policy operator

$$(T_\mu J)(x) = g(x,\mu(x)) + \alpha \sum_{x' \in \mathcal{X}} p(x' \mid x,\mu(x)) J(x'). \tag{2.43}$$

The symbol T_μ in this section is the policy-specific version of the global notation T_μ.

Theorem 2.3 (Contraction and fixed points). *Under assumption 2.3, both T and every T_μ are monotone and are contractions in the sup norm with modulus α:*

$$\|TJ - TJ'\|_\infty \le \alpha \|J - J'\|_\infty . \tag{2.44}$$

Consequently, T has a unique fixed point J^*, each T_μ has a unique fixed point J_μ, and a policy greedy with respect to J^* is optimal.

Proof. If $J \le J'$ componentwise, every action value formed with J is no larger than the corresponding action value formed with J'. Taking minima proves monotonicity. Also, for every x,

$$\begin{aligned} |(TJ)(x) - (TJ')(x)| &\le \max_{u \in \mathcal{U}(x)} \alpha \left| \sum_{x'} p(x' \mid x,u)(J(x') - J'(x')) \right| \\ &\le \alpha \|J - J'\|_\infty . \end{aligned}$$

Taking the maximum over x gives Equation (2.44); the policy operator proof is identical without the minimization. The contraction mapping theorem gives unique fixed points and convergence of repeated operator application. Choose μ^* greedy with respect to J^*. Then $T_{\mu^*}J^* = TJ^* = J^*$. Uniqueness of the policy-evaluation fixed point implies $J_{\mu^*} = J^*$. For any other policy μ, $TJ_\mu \le T_\mu J_\mu = J_\mu$. Repeated application of T, followed by the contraction limit, gives $J^* \le J_\mu$. Thus μ^* is optimal, and the fixed point is the minimum policy cost rather than merely a solution of an operator equation. □

The fixed-point equation

$$J^* = TJ^* \tag{2.45}$$

is exact. It does not say that a continuous-state J^* can be stored, that the expectation can be evaluated without error, or that the minimization is cheap. Those are computational questions introduced by the chosen representation.

Terminology note

Assumption 2.3 deliberately uses finite state and action sets so the contraction, minimizer, and tabular algorithms can be developed without measure-theoretic machinery. This is also the principal elementary setting for discounted VI and PI in Bertsekas (2019) and Bertsekas (2025). It is a theorem scope, not a claim that DP only applies to finite spaces. General-state versions replace sums by integrals and minima by suitable infima, with measurability and selection assumptions. Their numerical solution normally uses structure, continuous optimization, sampling, or function approximation rather than mandatory state–action discretization.

2.7 Finite Tables and Continuous Control Spaces

The state space and the control space are two independent modeling choices. A problem may have a continuous state and a finite action library, a finite state and a continuous control, or both spaces continuous. The finite-state, finite-control assumption in assumption 2.3 is used here because it makes every Bellman backup, minimizer, and convergence statement explicit. It is not part of the definition of DP.

For a deterministic stationary problem with continuous state or control, the optimality equation is still

$$J^*(x) = \inf_{u \in \mathscr{U}(x)} \{g(x,u) + \alpha J^*(f(x,u))\}. \tag{2.46}$$

For a stochastic model it is

$$J^*(x) = \inf_{u \in \mathscr{U}(x)} \mathbb{E}\left[g(x,u,w) + \alpha J^*(f(x,u,w)) \mid x,u\right]. \tag{2.47}$$

The equations are conceptually unchanged, but the expectation may be an integral, the infimum may require a continuous optimizer, and the value is now a function that cannot generally be stored as a complete table.

Table 2.2: Three routes from a continuous physical control problem to an implementable DP-based controller.

Route	What is retained	What becomes approximate or computationally demanding
Tabular model	finite states and controls	state/control quantization, interpolation, and model abstraction
Continuous minimization	continuous u and a model	online numerical optimization and a represented value $\tilde{J}$
Parameterized policy	continuous actor $u = \mu_\theta(x)$	policy-class restriction, sample coverage, and training error

If a continuous steering or torque command is represented by the finite set used in this chapter's tabular algorithms, then the implemented command can only be one of those finite choices. In that specific formulation, discretization—or a naturally finite maneuver library—is unavoidable. What is *not* unavoidable is using that formulation for every controller. LQR keeps the continuous action and solves the minimization analytically; MPC keeps it and solves a numerical optimization; a continuous actor directly outputs a real-valued action.

Control viewpoint

The course will use the finite table as a reference model, then relax one axis at a time. Chapter 4 begins with finite-state and finite-action VI, PI, Q-learning, and their guarantees, then replaces the state table by function approximation. DQN still assumes a finite action set. A continuous action paired with a critic requires either an inner numerical minimization or a separate actor; the latter route leads to TD3 and SAC in Chapter 6. Chapter 5 provides the analytic and optimization-based control bridge before those learned methods appear.

2.8 Exact and Approximate Value and Policy Iteration

The stationary discounted infinite-horizon problem in Section 2.6 is characterized by the fixed point $J^* = TJ^*$, but a fixed-point equation does not compute itself. Value iteration (VI) and policy iteration (PI) are algorithms for solving that infinite-horizon problem on the declared finite model. This differs from the single backward pass used for finite-horizon DP: VI repeats optimality backups until a fixed point is reached, whereas PI alternates infinite-horizon policy evaluation and policy improvement.

Control viewpoint

Exact tabular VI and PI are planning methods. A standard sweep assumes that $g(x,u)$ and either $f(x,u)$ or $p(x' \mid x,u)$ are available for every feasible state–control pair. It also updates or evaluates every state, including states not visited by the current physical trajectory. The completed table may be cheap to execute online, but constructing it can be expensive offline. This gap motivates asynchronous, sample-based, and function-approximation methods.

2.8.1 Exact value iteration

Value iteration repeatedly applies the optimality operator:

$$J^{(i+1)} = TJ^{(i)}. \tag{2.48}$$

Written without operator notation, its deterministic and stochastic forms are

$$J^{(i+1)}(x) = \min_u \left\{ g(x,u) + \alpha J^{(i)}(f(x,u)) \right\}, \qquad \text{deterministic,} \tag{2.49}$$

$$J^{(i+1)}(x) = \min_u \left\{ g(x,u) + \alpha \sum_{x'} p(x' \mid x,u) J^{(i)}(x') \right\}, \qquad \text{stochastic.} \tag{2.50}$$

Here i is an algorithm iteration, not physical time. One synchronous iteration evaluates the right-hand side at every x. By theorem 2.3,

$$\left\| J^{(i)} - J^* \right\|_\infty \le \alpha^i \left\| J^{(0)} - J^* \right\|_\infty. \tag{2.51}$$

Because J^* is unknown, a Bellman residual gives a computable certificate:

$$\|J - J^*\|_\infty \le \frac{1}{1-\alpha} \|TJ - J\|_\infty. \tag{2.52}$$

Indeed, $\|J - J^*\|_\infty \le \|J - TJ\|_\infty + \|TJ - TJ^*\|_\infty$ and the contraction term can be moved to the left.

After any iterate, a greedy policy is obtained from

$$\mu_i(x) \in \arg\min_{u \in \mathcal{U}(x)} \left\{ g(x,u) + \alpha \sum_{x'} p(x' \mid x,u) J^{(i)}(x') \right\}. \tag{2.53}$$

Small value error often helps, but policy quality depends on action-value differences. If two controls are nearly tied, a small approximation error can change the selected action.

Algorithm lens: Exact tabular value iteration

Choose $J^{(0)}$. At iteration i, use the model to evaluate all feasible actions at every state, store the minimum as $J^{(i+1)}(x)$, compute a Bellman residual, and stop at a declared tolerance. The limiting statement is exact for the declared finite model; finite-precision

termination is tolerance based.

2.8.2 Approximate value iteration

Suppose values are restricted to a tractable class, such as a linear architecture $\tilde{J}(x; r) = \phi(x)^\top r$, a kernel approximator, or a neural network. Let Π denote the operation that fits or projects a Bellman target back into that class. Approximate VI (AVI) has the generic form

$$\tilde{J}^{(i+1)} \approx \Pi T \tilde{J}^{(i)}. \tag{2.54}$$

Its deterministic and stochastic targets are, respectively,

$$y_i^{\mathrm{det}}(x) = \min_u \left\{ g(x,u) + \alpha \tilde{J}^{(i)}(f(x,u)) \right\}, \tag{2.55}$$

$$y_i^{\mathrm{sto}}(x) = \min_u \mathbb{E}\left[g(x,u,w) + \alpha \tilde{J}^{(i)}(f(x,u,w)) \,\middle|\, x, u \right]. \tag{2.56}$$

The new parameters are fitted so that $\tilde{J}^{(i+1)}(x) \approx y_i(x)$ on selected training states. With a known model, the expectation can be computed or simulated offline. Without an explicit model, samples can provide noisy regression targets.

Unlike exact VI, an arbitrary projection ΠT need not be a contraction and may not converge. Later chapters treat approximation architecture, training-state coverage, target construction, and error in detail.

2.8.3 Exact policy iteration

Policy iteration (PI) separates two operations.

For *policy evaluation*, compute the unique solution of

$$J_{\mu_i} = T_{\mu_i} J_{\mu_i}. \tag{2.57}$$

In vector form this is

$$(I - \alpha P_{\mu_i}) J_{\mu_i} = g_{\mu_i}. \tag{2.58}$$

For deterministic dynamics the same evaluation equation is

$$J_{\mu_i}(x) = g(x, \mu_i(x)) + \alpha J_{\mu_i}(f(x, \mu_i(x))), \tag{2.59}$$

whereas Equation (2.58) represents the finite stochastic case. For *policy improvement*, choose a policy greedy with respect to the evaluated value:

$$\mu_{i+1}(x) \in \arg\min_{u \in \mathcal{U}(x)} \left\{ g(x,u) + \alpha \sum_{x'} p(x' \mid x, u) J_{\mu_i}(x') \right\}. \tag{2.60}$$

Theorem 2.4 (Policy improvement). *Under assumption 2.3, let $\bar{\mu}$ be greedy with respect to J_μ. Then*

$$J_{\bar{\mu}} \leq J_\mu \tag{2.61}$$

componentwise.

Proof. Greediness and the policy-evaluation equation give

$$T_{\bar{\mu}} J_\mu = T J_\mu \leq T_\mu J_\mu = J_\mu.$$

Monotonicity implies $T_{\bar{\mu}}^m J_\mu \leq J_\mu$ for every m. The contraction property gives $T_{\bar{\mu}}^m J_\mu \to J_{\bar{\mu}}$, proving the result. □

With finitely many stationary policies and consistent tie handling, exact PI reaches an optimal policy in finitely many strict improvement steps. This does not imply that approximate policy iteration is monotone: evaluation error and restricted policy classes can invalidate the ordering used in the proof.

Algorithm lens: Exact tabular policy iteration

Start with a stationary policy μ_0. Use the full model to evaluate its value at every state, improve the action at every state, and repeat until no action changes. Policy evaluation may solve a linear system or iterate $J \leftarrow T_{\mu_i} J$. Exact PI can require fewer improvement steps than VI, but each evaluation can be much more expensive than one VI sweep.

2.8.4 Approximate policy iteration

Approximate PI replaces exact policy evaluation by an approximation $\tilde{J}_{\mu_i} \approx J_{\mu_i}$ and may also restrict improvement to a parameterized policy class. With an unrestricted control minimization, its deterministic and stochastic improvement steps are

$$\mu_{i+1}(x) \in \arg\min_u \left\{ g(x,u) + \alpha \tilde{J}_{\mu_i}(f(x,u)) \right\}, \tag{2.62}$$

$$\mu_{i+1}(x) \in \arg\min_u \mathbb{E}\left[g(x,u,w) + \alpha \tilde{J}_{\mu_i}(f(x,u,w)) \,\middle|\, x,u \right]. \tag{2.63}$$

Approximate evaluation may be model based, using transitions generated from a known f or p, or model free in the DP-backup sense, using trajectories from the system or a simulator. If a Q-factor $\widetilde{Q}_{\mu_i}$ is learned, the improvement step becomes $\mu_{i+1}(x) \in \arg\min_u \widetilde{Q}_{\mu_i}(x,u)$ and needs no explicit transition sum at execution.

Approximate PI does not automatically inherit Equation (2.61). Evaluation error, incomplete state coverage, and a restricted actor can cause policy oscillation or degradation. The true cost of each candidate policy should therefore be evaluated separately from the critic used to construct it.

2.8.5 Q-factor value iteration and policy iteration

Both VI and PI have exact Q-factor forms. For the stochastic discounted model, the optimal Q-factor satisfies

$$Q^*(x,u) = g(x,u) + \alpha \sum_{x'} p(x' \mid x,u) \min_{v \in \mathcal{U}(x')} Q^*(x',v), \tag{2.64}$$

with the deterministic form obtained by replacing the sum by $\min_v Q^*(f(x,u),v)$. Define

$$(T_Q Q)(x,u) = g(x,u) + \alpha \sum_{x'} p(x' \mid x,u) \min_v Q(x',v). \tag{2.65}$$

Then Q-factor value iteration exists and is simply

$$Q^{(i+1)} = T_Q Q^{(i)}. \tag{2.66}$$

It updates every state–control pair rather than every state. Approximate Q-value iteration fits

$$\widetilde{Q}^{(i+1)} \approx \Pi_Q T_Q \widetilde{Q}^{(i)}. \tag{2.67}$$

For Q-factor policy iteration, evaluation of a fixed policy solves

$$Q_{\mu_i}(x,u) = g(x,u) + \alpha \sum_{x'} p(x' \mid x,u) Q_{\mu_i}(x', \mu_i(x')), \tag{2.68}$$

For deterministic dynamics, the continuation term becomes $Q_{\mu_i}(f(x,u), \mu_i(f(x,u)))$. Policy improvement then selects

$$\mu_{i+1}(x) \in \arg\min_u Q_{\mu_i}(x,u). \tag{2.69}$$

The J- and Q-factor versions generate the same exact greedy improvement when they are evaluated consistently. The tradeoff is representational: Q stores more entries, but once stored it exposes the action comparison without an explicit transition sum. Exact Q-VI and Q-PI are nevertheless model based during computation; the sample-based step that leads to model-free Q-learning is introduced after the mobility example.

2.8.6 Modified and optimistic policy iteration

VI and PI are endpoints of a useful spectrum. Given an approximate value $J^{(i)}$, choose μ_i greedy with respect to it and perform m_i policy- evaluation sweeps:

$$J^{(i+1)} = T_{\mu_i}^{m_i} J^{(i)}, \qquad T_{\mu_i} J^{(i)} = T J^{(i)}. \tag{2.70}$$

When $m_i = 1$, the update is VI. In the limit of exact evaluation, it becomes the evaluation step of PI. Intermediate choices trade the frequency of policy improvement against the work devoted to evaluating the current policy.

This deterministic m_i-sweep scheme is usually called *modified policy iteration*. Bertsekas (2019) uses *optimistic policy iteration* for the broader family in which policy improvement occurs before full policy evaluation, possibly through asynchronous or simulation-based partial updates. Modified PI is therefore a structured special case of the broader optimistic idea.

2.8.7 Multistep policy improvement: a geometric preview

A different use of multiple Bellman steps is online lookahead. Given a terminal approximation $\tilde{J}$, an ℓ-step planner applies the first action from the computation represented by

$$T^{\ell}\tilde{J}. \tag{2.71}$$

Equivalently, define the effective evaluator $\bar{J} = T^{\ell-1}\tilde{J}$ and perform a one-step greedy improvement with respect to $\bar{J}$. The selected policy $\widetilde{\mu}$ satisfies

$$T_{\widetilde{\mu}}\bar{J} = T\bar{J}. \tag{2.72}$$

For a finite-state model, the graph of T is the lower envelope of the affine policy operators T_{μ}. The greedy policy selects an operator that touches this envelope at $\bar{J}$; solving $J = T_{\widetilde{\mu}}J$ is therefore a Newton-type policy-evaluation step for the Bellman fixed-point equation. This is only a preview. Chapter 5 gives the geometric picture and makes the construction explicit through the scalar Riccati map, following the interpretation in Bertsekas (2022).

The terminal approximation evaluates the leaf states; the online minimizations improve the action before execution. More lookahead is not an unconditional performance guarantee when models or minimizations are approximate.

The two depths must not be confused:

$$\underbrace{m}_{\text{policy-evaluation depth}} \quad \text{and} \quad \underbrace{\ell}_{\text{policy-improvement/lookahead depth}}. \tag{2.73}$$

They can be varied independently and can also be combined.

AI/RL terminology bridge

The evaluation–improvement split is the main bridge to modern actor–critic methods. A critic approximates policy evaluation; an actor performs improvement within a parameterized policy class. PPO, TD3, and SAC differ in objective, regularization, sampling, and value representation, but each can be located in this approximate-PI template. Using an actor as a base policy or warm start and a critic as a terminal value for online lookahead adds a further policy- improvement step at execution time. Chapter 6 develops this connection rather than presenting the algorithms as an unrelated catalog.

2.9 Mobility Case: A Stationary Stochastic Lane-Keeping MDP

The algorithms are now applied to a stationary infinite-horizon example. To parallel the finite-horizon HEV case, consider a vehicle traveling on a straight road under a stationary lateral disturbance. Let the normalized lateral-error state, steering action, and disturbance be

$$e \in \{-4, -3, \dots, 4\}, \qquad u \in \{-1, 0, 1\}, \qquad w \in \{-1, 0, 1\}, \tag{2.74}$$

with disturbance probabilities $(0.10, 0.60, 0.30)$. The transition and stage cost are

$$e_{k+1} = \mathrm{clip}(e_k + u_k + w_k, -4, 4), \tag{2.75}$$

$$g(e, u) = e^2 + 0.20u^2, \qquad \alpha = 0.95. \tag{2.76}$$

The model is stationary because the transition probabilities, cost, and constraints do not depend on k. It is a normalized teaching model rather than a calibrated vehicle model, but every Bellman quantity can be audited.

Exact VI and PI use the known transition kernel for all nine states and all three controls. Two shorter model-based planners are included as baselines. The myopic policy is greedy with respect to a zero terminal value,

$$\mu_{\mathrm{myopic}}(e) \in \arg\min_u g(e, u), \tag{2.77}$$

so it selects zero steering everywhere. The two-stage receding-lookahead policy first computes $T0$ and is greedy with respect to that one-stage evaluator; its root action is represented by $T^2 0$. Both exact DP and the two-stage planner use the transition model, whereas the myopic action selection uses only the present cost. Evaluating the infinite-horizon cost of all four policies uses the same declared transition kernel.

Table 2.3: Verified lane-keeping results. The mean is uniform over all nine initial states and reports the true infinite-horizon cost of each policy.

Method	Construction	Cost from $e = 0$	Mean policy cost
Exact VI	optimality backups to convergence	9.1200	21.0966
Exact PI	exact evaluation and improvement	9.1200	21.0966
Two-stage lookahead	greedy with respect to $T0$	9.1200	21.0966
Myopic	greedy with respect to zero	144.0896	178.8844

In this deliberately small model, two-stage lookahead happens to select the same stationary policy as exact VI and PI. This is an auditable feature of this example, not a general two-stage optimality result. The large myopic loss shows why the current lateral error alone is an inadequate action criterion under a persistent disturbance.

Implementation note

The lane-keeping calculation is generated by the same Chapter 2 MATLAB script as the HEV example. Exact VI is run to a sup-norm update tolerance of 10^{-12}; exact PI solves a model-based policy-evaluation linear system at each improvement step. The myopic and two-stage policies are also evaluated by solving their exact model-based policy equations, so plotted values are achieved policy costs rather than terminal heuristics.

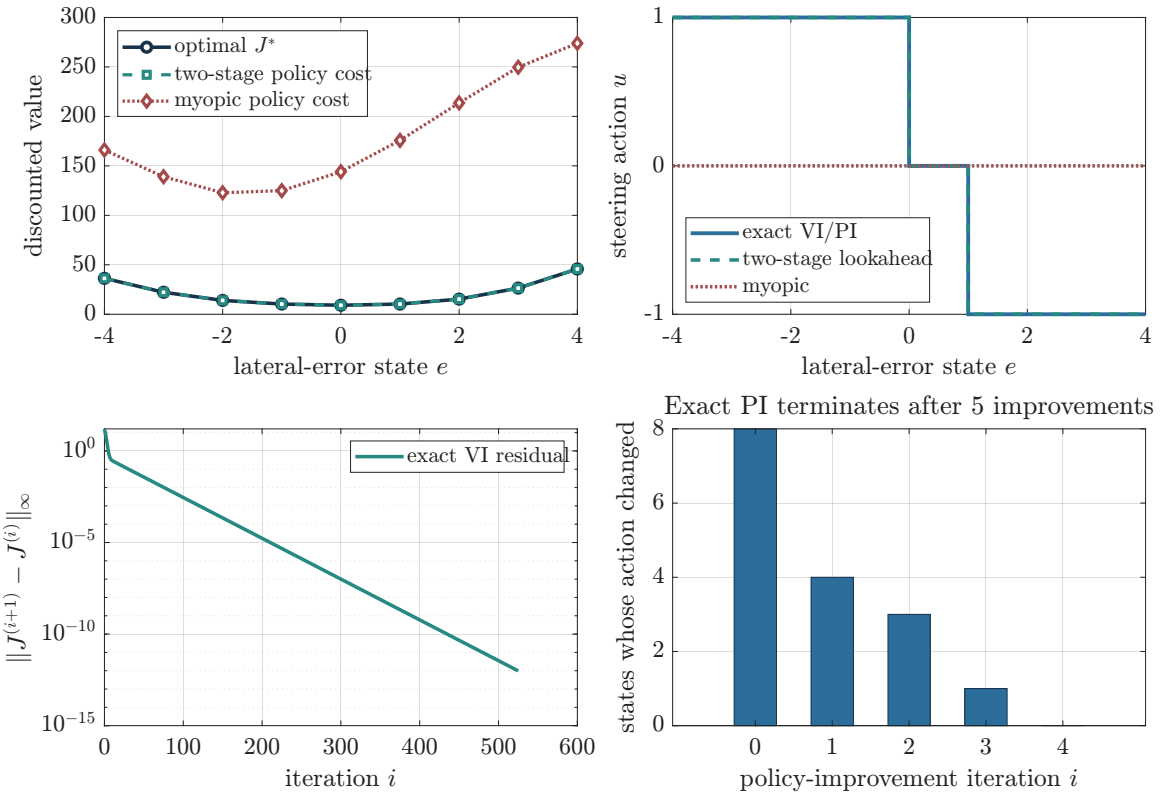


Figure 2.6: Model-based exact and short-lookahead calculations for the stationary stochastic lane-keeping model. The top panels compare achieved policy costs and actions; the bottom panels expose VI residual convergence and the number of state actions changed by PI. The two-stage policy coincides with the optimum in this teaching model, while the myopic policy does not.

2.10 From Q-Value Iteration to Q-Learning

Once Q^* from Equation (2.64) is available,

$$J^*(x) = \min_u Q^*(x,u), \qquad \mu^*(x) \in \arg\min_u Q^*(x,u). \tag{2.78}$$

Exact Q-VI still uses the model in every backup. Q-learning replaces that explicit transition sum by a sampled transition (x_i, u_i, g_i, x_{i+1}):

$$Q^{(i+1)}(x_i,u_i) = Q^{(i)}(x_i,u_i) + \beta_i \left[g_i + \alpha \min_v Q^{(i)}(x_{i+1},v) - Q^{(i)}(x_i,u_i) \right], \tag{2.79}$$

while other table entries remain unchanged. For a deterministic system the observed successor is fixed by (x_i, u_i); for a stochastic system the bracket is a noisy sample of the Q-Bellman target.

In the finite tabular discounted setting, standard convergence results (Bertsekas 2019; Sutton and Barto 2018) require, among other conditions, repeated visitation of every relevant state–control pair and step sizes satisfying $\sum_i \beta_i = \infty$ and $\sum_i \beta_i^2 < \infty$ along each updated pair.

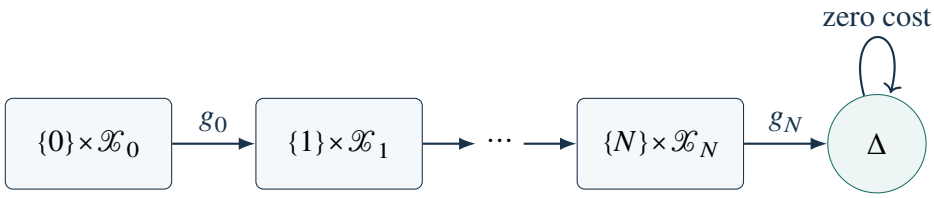


Figure 2.7: Finite-to-infinite reformulation. Stage-indexed state layers are combined into one augmented state space and followed by a cost-free absorbing state. A stationary policy on the augmented state retains the original stage information.

Neural-network Q-learning no longer inherits this elementary guarantee. Chapter 4 develops sampling, temporal differences, target networks, and DQN; the purpose here is to show that Q-learning is a sample-based approximation of the same DP equation, not a separate starting point.

2.11 Finite-to-Infinite Reformulation

A finite-horizon problem can be represented as a stationary infinite-horizon problem by putting time into the state. Let

$$z_k = (k, x_k), \qquad k = 0, \dots, N, \tag{2.80}$$

and add an absorbing termination state Δ. From (k, x) with $k < N$, the augmented system moves to $(k+1, f_k(x,u))$ and incurs $g_k(x,u)$. From (N,x) it moves to Δ and incurs $g_N(x)$; thereafter the cost is zero.

The transition rule and cost are stationary as functions of z, even though the original model was time varying. A stationary policy in z is exactly a nonstationary policy $\mu_k(x)$ in the original state. The two formulations therefore have the same feasible trajectories and total costs.

Proposition 2.2 (Equivalence of the time-augmented formulation). *Every feasible policy for the original N-stage problem induces a feasible stationary policy on the augmented state space with identical cost, and vice versa. Hence the optimal costs are equal.*

Proof. Map the original decision rule $\mu_k(x)$ to $\bar{\mu}((k,x)) = \mu_k(x)$. The augmented transition reproduces the same state sequence and charges the same stage and terminal costs before entering Δ. Conversely, restrict any augmented stationary policy to states of the form (k,x) to obtain the original nonstationary rules. The mappings preserve feasibility and cost in both directions. □

This equivalence is conceptual, not a complexity reduction. If the original state grid has n points at every stage, the augmented representation has roughly $(N+1)n$ state–time pairs. State augmentation can also make an otherwise compact physical state much larger.

AI/RL terminology bridge

The augmented problem is an episodic MDP written on an infinite time axis. It allows finite-horizon control problems to use the common language of stationary policies, value functions, Q-factors, and RL algorithms. The equivalence is valid only if stage, remaining time, terminal cost, and terminal constraints are encoded. Omitting them changes the MDP. This reformulation justifies using infinite-horizon methodology as a unifying language in later chapters, not the claim that every task is naturally continuing or discounted.

2.11.1 Can a slalom task be treated as infinite horizon?

A prescribed finite slalom is nonstationary if the controller observes only the vehicle state while the desired path changes with time or distance. It can nevertheless be formulated for approximate infinite-horizon DP by choosing an appropriate information state. Let x_k^{veh} contain vehicle motion states and let σ_k denote progress or phase along the reference path. Include preview information $r(\sigma_k)$ when the controller uses upcoming curvature or cone geometry:

$$z_k = (x_k^{\text{veh}}, \sigma_k, r(\sigma_k)). \tag{2.81}$$

The dynamics and stage cost may then be stationary functions of (z_k, u_k).

There are two useful interpretations:

1. A finite course becomes an episodic infinite-horizon problem with absorbing success, collision, and track-exit states.
2. A closed or repeatedly generated slalom becomes a continuing problem; path phase is part of the state, and discounted or average performance can be defined over repeated traversal.

The state and action are continuous, so a tabular solution is not required or usually desirable. A critic $\tilde{J}(z)$ or $\widetilde{Q}(z, u)$, a continuous actor, and optional model-based lookahead provide an approximate-DP implementation. PPO, TD3, or SAC can train the actor and critic; an LQR/MPC baseline and a short online optimizer can expose what is gained by learning.

Mobility case: Slalom as a midterm replacement project

The task is appropriate as an approximate infinite-horizon DP project if students must explicitly define the augmented Markov state, episode termination and terminal cost, cost/reward sign conversion, model use during training and execution, and a nonlearned baseline. Evaluation should report tracking error, cone or road-boundary violations, steering magnitude and rate, computation time, and robustness to speed and tire-model mismatch. Training return alone does not establish a useful controller.

2.12 Approximation, Model Use, and Implementation

This section separates three questions that are often compressed into the word *approximate*: which mathematical object is represented, how Bellman information is obtained, and how the

resulting action is selected.

2.12.1 Three levels of exactness

The word *exact* is useful only when its level is stated. A short three-level distinction is sufficient for the remainder of the book.

Table 2.4: Three levels at which exactness may be claimed.

Level	Meaning and required qualification
Bellman characterization	Exact under stated assumptions; it does not imply that the value can be stored or computed.
Declared numerical model	Exact VI/PI may solve a finite table; the table can still approximate the physical state, action, or dynamics.
Implemented policy	A policy can be evaluated accurately on the declared model; this does not by itself establish performance on the physical plant.

Three functions should be kept distinct:

$$J^*, \qquad \tilde{J}, \qquad J_{\tilde{\mu}}. \tag{2.82}$$

The first is the optimal value of the declared problem. The second is a stored approximation used to make a decision. The third is the true cost of the policy induced by that approximation. In general,

$$\tilde{J} \neq J_{\tilde{\mu}} \neq J^*. \tag{2.83}$$

The lane-keeping example demonstrates this distinction without a separate function approximator. The myopic controller uses the terminal evaluator zero, but its achieved cost from $e = 0$ is 144.0896; the two-stage controller uses $T0$ as an evaluator, but its achieved cost is 9.1200. A leaf evaluator and the true cost of its induced receding-lookahead policy are different objects.

2.12.2 Where approximation can enter

Let $\hat{T}$ denote a backup formed with an approximate model, sampled expectation, or approximate minimization. A broad approximate-DP template is

$$\tilde{J}^{(i+1)} \approx \Pi \hat{T} \tilde{J}^{(i)}, \tag{2.84}$$

$$\tilde{\mu}(x) \approx \arg\min_u \widehat{Q}_{\tilde{J}}(x, u). \tag{2.85}$$

This notation separates representation error through Π, model or sampling error through $\hat{T}$, and optimization or policy-class error through the approximate minimization.

Table 2.5: Principal approximation locations in DP and learning-based control.

Approximation location	Exact object	Representative approximation
State representation	full information state x	grid, feature state, or belief state
Value space	J^* or J_μ	$\tilde{J}(x;r)$, critic, terminal value
Q-factor space	Q^* or Q_μ	$\widetilde{Q}(x,u;r)$, DQN or continuous critic
Transition/expectation	f or $p(x' \mid x,u)$	learned model, nominal forecast, Monte Carlo sample
Minimization	global $\min_{u\in\mathcal{U}}$	sampled controls, local solver, short lookahead
Policy space	unrestricted selector μ	actor μ_θ or stochastic policy π_θ
Problem	original dynamics and constraints	reduced model, certainty equivalence, relaxed constraints

If a d-dimensional continuous state is discretized with M points per coordinate, a full Cartesian grid contains M^d states. Ten points per coordinate produce 10^4 states in four dimensions and 10^8 states in eight dimensions, before controls and disturbances are enumerated. This exponential growth is one form of the curse of dimensionality.

2.12.3 Model use is not a single binary label

The words *model based* and *model free* are overloaded. This book will track three separate questions:

Table 2.6: Three model-use axes that should be reported separately.

Axis	Question to report
Training backup	Is f or p used explicitly, or are Bellman targets formed from transition samples?
Online action selection	Is a stored actor/Q-policy executed directly, or is a model-based lookahead or MPC problem solved?
Model origin	If a predictive model is used, is it known physics, an engineering approximation, or learned dynamics?

A learned value is not automatically model free: fitted VI may use a known model. Online improvement is often model based, but it may instead use a simulator, sampled rollout, or learned Q-factor. Chapter 1 will introduce these axes, and later chapters will identify where each method lies on them.

Approximation changes the computation, not the logic of the control question. Chapters 3 and 4 replace tabular value functions with parameterized approximations and samples. Chapter 6

moves approximation into the policy space. The reference objects in this chapter—J^*, J_μ, Q^*, Q_μ, T, and T_μ—make it possible to say precisely what is being approximated.

2.13 Before Applying Dynamic Programming

Conditions to check

Before calling a sequential decision model a DP or MDP, check the following.

1. **State sufficiency.** Does x_k contain the information needed to predict the next-state law and future costs under a control? If not, add memory, an estimator state, or a belief state.
2. **Information timing.** Is the control chosen before or after the current disturbance, forecast update, or neighboring-vehicle message?
3. **Admissibility.** Are state, input, and coupled terminal constraints represented in $\mathcal{U}_k(x)$ or in an equivalent extended cost?
4. **Terminal modeling.** Does g_N describe the cost beyond the explicit horizon, or does it merely force an arbitrary endpoint?
5. **Optimization scope.** Are the displayed minima attained, and is the implementation finding global or only local minimizers?
6. **Probability model.** Is an expectation computed from a known kernel, a learned model, a forecast ensemble, or samples? Each has a different error source.
7. **Infinite-horizon criterion.** Is discounting physically meaningful, a mathematical convenience, or a proxy for termination? Near-unity discount factors need not reproduce an undiscounted objective.
8. **Claimed exactness.** Is the claim about the Bellman relation, a finite abstract model, or the original physical system?

Discounting deserves particular care. It guarantees contraction in theorem 2.3, but it also changes how costs at different times are weighted. Average-cost, stochastic-shortest-path, and undiscounted control problems need their own assumptions; they cannot inherit the discounted proof by setting $\alpha = 1$.

2.14 Summary

- The principle of optimality turns a sequential optimization problem into coupled tail problems.
- Finite-horizon Bellman recursion is backward in time and generally produces a nonstationary policy.
- Stochastic DP replaces the next tail value by a conditional expectation; the information

pattern and Markov state must be explicit.

- Deterministic and stochastic approximate DP replace exact values, Q-factors, expectations, models, minimizations, or policies by tractable approximations. These approximation locations should be named explicitly.
- A tabular finite-control implementation of a continuous physical problem requires action discretization or a finite maneuver library. DP itself can instead retain continuous controls through analytic or numerical minimization, or a parameterized actor.
- For a finite discounted MDP, Bellman operators are contractions. This gives unique fixed points, VI convergence, and a clean policy-improvement proof.
- Exact VI and PI use a model and compute over all states. Approximate VI and PI trade this exhaustive computation for representation, sampling, model, optimization, and coverage errors.
- VI and PI can be written with either state values or Q-factors. Exact Q-VI and Q-PI remain model based; Q-learning replaces the explicit transition expectation by sampled updates.
- Modified PI is a structured optimistic-PI method that truncates policy evaluation. Multistep lookahead changes improvement depth instead and admits a Newton-type geometric interpretation developed in Chapter 5.
- Q-learning is a sample-based asynchronous approximation of Q-factor value iteration; it does not begin from a different optimality equation.
- Actor–critic methods inherit the evaluation–improvement structure; online lookahead can improve the learned actor using the learned critic.
- Time or path-phase augmentation makes finite-horizon and slalom tasks stationary in an enlarged state. This enables episodic infinite-horizon formulations without removing their computational burden.

2.15 Exercises

Exercise 2.1 (Backward recursion by hand)**.** Consider $x_{k+1} = x_k + u_k$ for $k = 0, 1, 2$, with $x_k \in \{-2, -1, 0, 1, 2\}$, $u_k \in \{-1, 0, 1\}$ whenever the next state remains in the state set, stage cost $x_k^2 + 0.2u_k^2$, and terminal cost $2x_3^2$. Compute $J_3^*, J_2^*, J_1^*, J_0^*$ and the optimal policy. Explain why the policy depends on k.

Exercise 2.2 (A stochastic Bellman step)**.** A vehicle can remain in its current lane or request a lane change. Construct a two-state, two-action transition model with a failed-request probability. Specify costs for delay and request effort, then compute one backward Bellman step for a two-stage horizon. State exactly when the request outcome becomes known to the controller.

Exercise 2.3 (Deterministic and stochastic approximate lookahead)**.** For the problem in Exercise 2.1, replace J_2^* by $\tilde{J}_2(x) = 1.5x^2$ and compute the approximate stage-1 policy using Equation (2.22). Then add a disturbance $w \in \{-1, 1\}$ with equal probabilities and compute Equation (2.23). Identify separately the value-approximation and expectation operations.

Exercise 2.4 (Contraction and residual)**.** Prove that T_μ is an α-contraction under assumption 2.3. Starting from the triangle inequality and the fixed-point relation, derive Equation (2.52) without referring to the proof in the text.

Exercise 2.5 (Value and Q-factor iterations)**.** Create a discounted MDP with three states and two controls per state. Implement J-based VI, Q-factor VI, J-based PI, and Q-factor PI. Verify that the final policies agree. Report Bellman backups, linear solves, table sizes, and residuals. Counting an entire linear solve as one iteration is not a fair comparison; propose and justify a more informative work metric.

Exercise 2.6 (Modified policy iteration)**.** For the MDP in the previous exercise, run Equation (2.70) with $m = 1, 2, 5$, and exact policy evaluation. Compare convergence and total operator applications. Identify the VI and PI endpoints in the data.

Exercise 2.7 (Short lookahead and achieved policy cost)**.** Run the stationary lane-keeping example. Reproduce Table 2.3. Derive the myopic and two-stage policies from 0, $T0$, and T^20, and evaluate both policies on the infinite-horizon model. Then change the disturbance bias and steering penalty. Determine when two-stage lookahead ceases to coincide with the optimal policy, and explain why a terminal evaluator is not the achieved policy cost.

Exercise 2.8 (One Q-learning update)**.** For the lane-keeping MDP, initialize $Q^{(0)} = 0$. Suppose a sample transition is $(e, u, g, e') = (2, -1, 4.2, 1)$. Compute one update of Equation (2.79) with $\beta_0 = 0.5$. Explain what changes if the transition is deterministic, and why repeated visitation remains necessary.

Exercise 2.9 (Continuous control without an action grid)**.** Replace the lane-keeping action set by $u \in [-1, 1]$ and use a differentiable approximation $\tilde{J}(e)$. Derive the scalar continuous minimization used for one-step lookahead. Compare three implementations: discretizing u, using a numerical optimizer, and learning a continuous actor.

Exercise 2.10 (HEV reproducibility and sensitivity)**.** Run the Chapter 2 MATLAB script identified in the implementation note. Verify the total costs in Table 2.1. Then change the terminal target, battery cycling penalty, and engine-power limit one at a time. Explain changes in

both the cost-to-go and policy map. Do not call a modified run physically realistic unless the normalized model has been calibrated.

Exercise 2.11 (Hard versus soft terminal requirements)**.** Replace Equation (2.33) by $g_N(e) = \rho(e - 10)^2$. Sweep ρ over several orders of magnitude. Plot terminal error against operating cost and discuss why a large finite penalty is not identical to a hard constraint.

Exercise 2.12 (Finite-to-infinite augmentation)**.** Write the complete augmented transition graph for the problem in Exercise 2.1. Count the original state–time pairs and the augmented states. Show explicitly how a stationary augmented policy encodes the nonstationary finite-horizon policy.

Exercise 2.13 (Slalom as an episodic MDP)**.** Formulate a finite slalom as an episodic infinite-horizon problem. Specify the vehicle state, path progress or phase, preview information, continuous control, success and failure terminal states, and terminal costs. State whether training and execution are model based or sample based using the three axes in Table 2.6. Propose an LQR or MPC baseline and physical evaluation metrics.

Exercise 2.14 (Modeling audit)**.** Choose one mobility problem: eco-driving, lane changing, thermal management, or charging coordination. Define the state, control, disturbance, information timing, constraints, cost, and horizon. Then identify one omitted variable that would break the Markov property and propose a state augmentation.

3 Parametric Approximation

Learning objectives

After studying this chapter, the reader should be able to:

1. distinguish the object being approximated from the algorithm used to generate its training targets;
2. construct polynomial, tile, radial-basis, structured, and MLP architectures for values and Q-factors;
3. formulate a finite-action policy approximator as a classifier and distinguish it from a multihead Q-factor;
4. derive regularized weighted least squares and batch, minibatch, and Adam parameter updates;
5. explain how ℓ_1, ℓ_2, smoothness, early stopping, dropout, and decoupled weight decay encode different preferences;
6. distinguish fitting error from error already present in a training target;
7. explain why normalization, validation design, and trajectory-level splitting matter in control data;
8. separate value-prediction error from greedy-action error and decision regret;
9. state an action-gap condition under which an approximate Q-factor preserves the exact greedy action;
10. distinguish a neural-network architecture from a DP or RL training algorithm; and
11. assess whether a fitted approximator is ready to be used inside an online controller.

Chapter roadmap

This chapter deliberately pauses the Bellman recursion and studies the function class itself. It moves from the object being approximated to linear features, fixed-target fitting, and neural architectures. Data-distribution and decision-quality checks then lead to a common car-following experiment, after which Chapter 4 reintroduces Bellman and trajectory-generated targets.

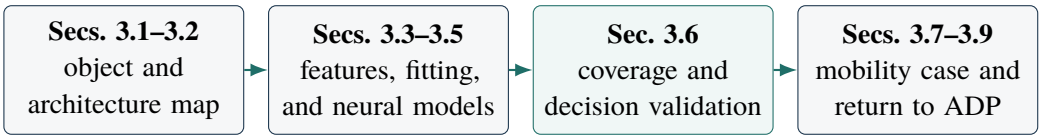


Figure 3.1: Original map of Chapter 3. Fitting a target and judging whether the fitted function supports the intended control decision are separate steps.

3.1 Why Approximation Architecture Comes Before Approximate DP

Chapter 2 introduced approximate value iteration in the form

$$\tilde{J}^{(i+1)} \approx \Pi T \tilde{J}^{(i)}. \tag{3.1}$$

The symbol Π hides several design decisions. Which functions can be represented? At which states is the fit accurate? How are noisy targets weighted? How expensive is evaluation inside an online minimization? Before studying the Bellman targets in Chapter 4, we isolate these questions by considering a fixed target function.

This separation is deliberate. An approximation architecture is a family of functions. A training rule chooses one member of that family. A DP or RL algorithm determines how the targets are generated and refreshed. Conflating the three makes it difficult to identify the source of an observed failure.

The chapter follows the parametric-architecture viewpoint of Bertsekas (2019) and Bertsekas (2025), while adding control-facing validation, continuous-action interpretation, and a mobility experiment. It is not intended to replace a general machine-learning text.

Control viewpoint

The relevant notion of accuracy depends on how the approximation is used. For action selection, an additive error common to every action does not change the minimizer, while an error that varies rapidly with the action can change it. This is the "constant offset versus error slope" observation emphasized by Bertsekas (2019, Sec. 2.1). It does *not* imply that value RMSE is unimportant or that a simple architecture is generally superior. It implies that prediction error and induced decision error answer different questions and should both be reported.

3.2 What Is Being Parameterized?

Let r denote a finite-dimensional parameter vector. The same mathematical idea can be applied to four different objects:

$$\tilde{J}(x; r_J) \approx J(x), \qquad \widetilde{Q}(x, u; r_Q) \approx Q(x, u), \tag{3.2}$$

$$\tilde{\mu}(x; r_\mu) \approx \mu(x), \qquad \tilde{f}(x, u; r_f) \approx f(x, u). \tag{3.3}$$

The notation records the object explicitly because its online consequence is different.

Table 3.1: What the stored approximation supplies at execution.

Stored object	Execution consequence
Value $\tilde{J}(x)$	A model and an online minimization are still needed to compare successor states.
Q-factor $\widetilde{Q}(x,u)$	A finite action can be selected directly by $\arg\min_u \widetilde{Q}(x,u)$; a continuous action still requires an optimizer or an actor.
Policy $\widetilde{\mu}(x)$	The action is produced directly, subject to any saturation or safety layer.
Model $\tilde{f}(x,u)$	Prediction becomes available, but an optimizer or policy is still required for control.

For finite-horizon problems, values and policies are generally stage dependent:

$$\tilde{J}_k(x;r_k), \qquad \widetilde{Q}_k(x,u;s_k), \qquad k=0,\dots,N-1. \tag{3.4}$$

Sharing parameters across stages is possible only after time, remaining horizon, or other non-stationary information has been encoded in the input. A single stationary network without this information silently changes the problem.

AI/RL terminology bridge

In AI terminology, $\tilde{J}$ is a value network or critic, $\widetilde{Q}$ is a Q-network, and $\widetilde{\mu}$ is an actor or policy network. The word *network* specifies an architecture, not how its targets were obtained. A Q-network trained from exact model-based Bellman targets and one trained from real transition samples have the same input–output form but different learning algorithms and model-use classifications.

3.2.1 A map of the architectures used in this chapter

Before deriving any one architecture, it helps to see the available choices. The first three rows of Table 3.2 use a feature map fixed before training and are linear in the fitted weights. The last row learns the feature map and is nonlinear in most of its parameters. These categories are not competing DP algorithms: any of them can be trained on supervised, Bellman, Monte Carlo, or temporal-difference targets.

The progression will therefore be: choose fixed features, understand how a target is fitted, and then let a neural network learn the features as well. Section 3.7 compares all four choices on the same mobility target.

3.2.2 Finite-action policies as classification

Policy approximation is not restricted to continuous actions or to a particular feature architecture. For a finite action set $\mathcal{U} = \{u^{(1)},\dots,u^{(m)}\}$, a policy approximator can be viewed as a *classifier*:

Table 3.2: Architecture map. "Local" means that changing one feature or weight affects only a limited state region; "global" means that it can affect much of the domain.

Architecture	Representation and typical behavior
Polynomial or structured fixed features	Global smooth terms and physics-informed quantities are chosen before training. The result is compact and interpretable; extrapolation follows the imposed structure.
Tile fixed features	Indicator functions are assigned to declared cells. The result is local and table-like, but discontinuous unless overlap or interpolation is added.
RBF fixed features	Smooth local kernels use selected centers and widths. They interpolate locally, while coverage deteriorates away from the centers.
MLP	A multilayer perceptron's widths and activation types are chosen, while internal features are learned from data. Training is nonlinear and generally nonconvex.

the state or information state is the object being classified, and each admissible action is a class. Let $z_j(x; r)$ be one score per action and define

$$p_j(x; r) = \frac{\exp z_j(x; r)}{\sum_{\ell=1}^{m} \exp z_\ell(x; r)}, \tag{3.5}$$

$$\widetilde{\mu}(x; r) = u^{(j^*)}, \qquad j^* \in \arg\max_j p_j(x; r). \tag{3.6}$$

The scores z_j may come from linear features, an MLP, a CNN, an RNN, or another parametric architecture. Classification describes the output object and training target; it does not imply a particular internal representation.

If improved or expert actions supply labels j_s, the standard cross-entropy fit is

$$r^* \in \arg\min_r -\sum_{s=1}^{q} w_s \log p_{j_s}(x^{(s)}; r) + \lambda \mathscr{R}(r). \tag{3.7}$$

One-hot least squares is another possible classifier fit. At execution, sampling from p_j gives a randomized policy, whereas Equation (3.6) gives a deterministic maximum- probability action. A binary stop/continue decision is the simplest special case. This is the finite-action policy-approximation viewpoint of Bertsekas (2025, Sec. 3.4.1).

This classifier should not be confused with a finite-action Q-factor. A Q approximator estimates action-conditioned costs and then selects $\arg\min_u \widetilde{Q}(x, u)$; a policy classifier learns action labels or probabilities directly and need not preserve cost magnitudes. The two can use the same multi-output architecture while representing different mathematical objects and requiring different targets.

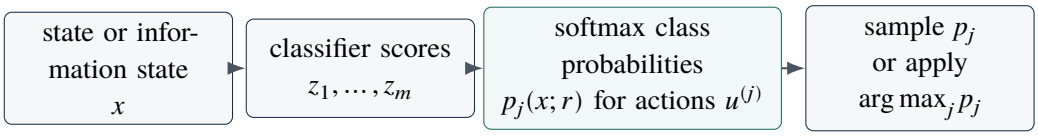


Figure 3.2: Original finite-action policy-classifier architecture. The same probability outputs support either a randomized training/execution policy or a deterministic maximum-probability controller; that choice must be stated.

3.3 Linear-in-Parameter Feature Architectures

Choose a feature map

$$\phi : \mathcal{X} \to \mathbb{R}^m, \qquad \phi(x) = \begin{bmatrix} \phi_1(x) & \cdots & \phi_m(x) \end{bmatrix}^\mathsf{T}. \tag{3.8}$$

A linear feature architecture has the form

$$\tilde{J}(x; r) = \phi(x)^\mathsf{T} r = \sum_{j=1}^{m} r_j \phi_j(x). \tag{3.9}$$

The architecture is linear in the parameters, not necessarily in the state. Polynomial, trigonometric, hinge, radial, and physics-derived nonlinear functions of x may all appear in ϕ.

3.3.1 Local, global, and structured features

Four choices recur throughout approximate control. They all fit Equation (3.9), but they impose different geometry on the approximating function. Piecewise-constant and polynomial examples are classical linear feature architectures in Bertsekas (2019, Sec. 3.1.1); tiles and radial bases are also standard constructions in Sutton and Barto (2018, Ch. 9).

Polynomial features. For a two-state regulation problem $x = (e_d, e_v)$, a quadratic vector is

$$\phi_{\text{quad}}(x) = \begin{bmatrix} 1 & e_d & e_v & e_d^2 & e_d e_v & e_v^2 \end{bmatrix}^\mathsf{T}. \tag{3.10}$$

Each basis function is global: changing one coefficient changes the fitted surface over the full domain. This is efficient when the true shape is close to quadratic but restrictive when it contains localized transitions.

Piecewise-constant or tile features. Let $\{\mathcal{C}_j\}_{j=1}^m$ partition or cover the state space. With

$$\phi_j(x) = \mathbf{1}\{x \in \mathcal{C}_j\}, \qquad \tilde{J}(x; r) = \sum_{j=1}^{m} r_j \mathbf{1}\{x \in \mathcal{C}_j\}, \tag{3.11}$$

the weight r_j controls only its active cell. A nonoverlapping partition is a compressed lookup table; overlapping tilings reduce visible cell-boundary artifacts but activate several weights per state.

Radial-basis features. A smooth local feature centered at c_j is

$$\phi_j(x) = \exp\left(-\frac{\|x - c_j\|^2}{2\sigma_j^2}\right). \tag{3.12}$$

Small σ_j gives local resolution but may leave holes between centers; large σ_j gives smoother coverage but couples distant states. RBFs do not acquire sensible extrapolation merely by increasing their number: far from every center, all radial features can become nearly zero.

Control-informed features. Known physical or decision structure can be encoded directly. For a spacing constraint $d \geq d_{\min}$, one useful feature is

$$\phi_{\text{risk}}(d) = [d_{\min} - d]_+^2, \qquad [z]_+ = \max\{0, z\}. \tag{3.13}$$

Including it does not enforce the constraint. It only makes a particular shape of cost easier to represent. Safety still requires an appropriate decision layer and verification.

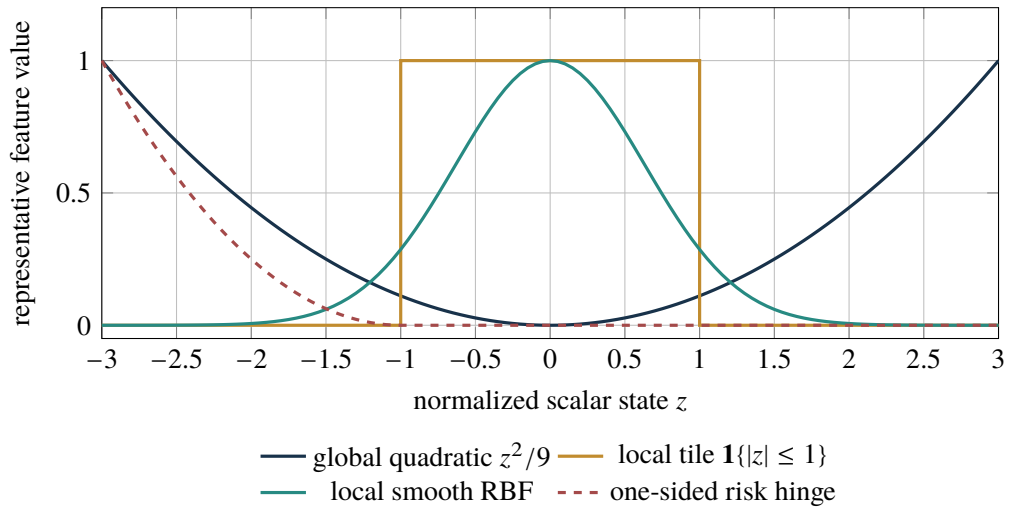


Figure 3.3: Original comparison of four representative fixed features. The curves are basis functions, not complete fitted values. A polynomial affects the whole domain, a tile is local and discontinuous, an RBF is local and smooth, and a structured hinge activates only on the declared risk side.

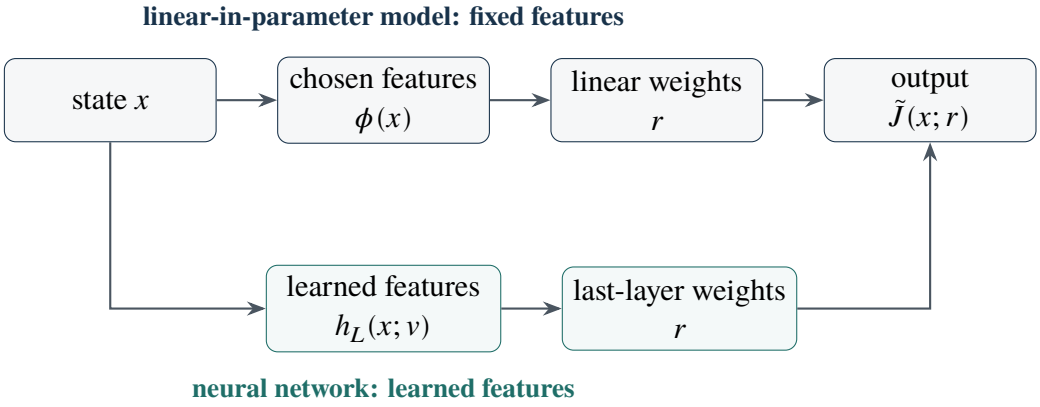


Figure 3.4: Original architecture diagram. A linear feature model learns only the last weights for a chosen representation. A neural model also learns the feature map. Both remain parametric function approximators; neither specifies the Bellman or policy-improvement algorithm that supplies the targets.

3.3.2 Q-factor and continuous-policy features

For a Q-factor, the features depend on both state and action:

$$\widetilde{Q}(x,u;r) = \psi(x,u)^{\mathsf{T}} r. \tag{3.14}$$

With a finite action set, one may use separate parameters for each action or a shared representation with multiple output heads. Separate heads can express different action surfaces but share no statistical strength unless their lower layers are coupled.

For a continuous control, a quadratic-in-action critic is especially useful:

$$\widetilde{Q}(x,u) = c(x) + b(x)^{\mathsf{T}} u + \tfrac{1}{2} u^{\mathsf{T}} H(x) u, \qquad H(x) \succ 0. \tag{3.15}$$

Its greedy action is available analytically before constraints, $u = -H(x)^{-1} b(x)$. A generic neural critic does not provide this convenience; it needs numerical minimization or a separate actor.

A continuous policy may also be parameterized directly. For scalar actuation, for example,

$$\widetilde{\mu}(x;r) = u_c + u_s \tanh\big(\phi(x)^{\mathsf{T}} r\big) \tag{3.16}$$

maps every state into the declared interval $[u_c - u_s, u_c + u_s]$. This is a policy architecture, not a Q-factor: its parameters must be trained by imitation, policy optimization, or another policy-space method rather than by minimizing it over u.

These forms are used in practice, but not with equal prevalence. Linear Q-factors remain important in least-squares policy iteration and related sample-efficient methods (Lagoudakis and Parr 2003); a finite-action multihead Q-network is the neural generalization used by DQN. Quadratic action dependence is exact for LQR Q-factors, appears locally in second-order optimal control, and was built into the normalized-advantage-function architecture for continuous Q-learning (Gu et al. 2016). For high-dimensional continuous actions, a separate actor is now often more flexible than enforcing a globally quadratic critic. Thus Equation (3.15) is a useful structured option, not a claim that every modern critic has this form.

3.4 Fitting Architectures to a Fixed Target

This section applies to both the fixed-feature models of Section 3.3 and the nonlinear models introduced in Section 3.5. The loss function defines what it means to fit; the architecture determines whether that optimization is a linear least- squares problem or a nonlinear training problem.

Suppose a target function J can be queried at selected states. The training set is

$$\mathcal{D} = \{(x^{(s)}, \beta^{(s)})\}_{s=1}^{q}, \qquad \beta^{(s)} = J(x^{(s)}) + \varepsilon^{(s)}. \tag{3.17}$$

Here $\varepsilon^{(s)}$ is the *label error*: the difference between the available label and the stated target at sample s. It is random noise when, for example, $\beta^{(s)}$ is a Monte Carlo return or a measurement. It may also contain deterministic numerical error or systematic bias when the label comes from interpolation, a learned model, or an incomplete optimizer. Calling every $\varepsilon^{(s)}$ zero-mean noise would therefore be too strong.

3.4.1 Weighted and regularized least squares

For a general architecture $\tilde{J}(x;\theta)$, weighted squared-error training takes the form

$$\theta_\lambda \in \arg\min_\theta \sum_{s=1}^{q} w_s \left(\tilde{J}(x^{(s)};\theta) - \beta^{(s)}\right)^2 + \lambda \mathcal{R}(\theta), \tag{3.18}$$

where $w_s \geq 0$ is a sample weight, $\lambda \geq 0$ is a regularization coefficient, and $\mathcal{R}$ penalizes selected parameter behavior. The objective is quadratic only when the architecture is linear in its fitted parameters.

Let the feature matrix $\Phi \in \mathbb{R}^{q\times m}$ have row $\phi(x^{(s)})^\top$, let $W = \operatorname{diag}(w_1, \dots, w_q)$, collect labels in β, and choose a regularization operator $L \in \mathbb{R}^{p\times m}$. The linear-in-parameter special case of Equation (3.18) solves

$$r_\lambda \in \arg\min_r (\Phi r - \beta)^\top W (\Phi r - \beta) + \lambda \|Lr\|^2. \tag{3.19}$$

If the indicated matrix is nonsingular, the solution is

$$r_\lambda = (\Phi^\top W \Phi + \lambda L^\top L)^{-1} \Phi^\top W \beta. \tag{3.20}$$

Otherwise a pseudoinverse or a numerically stable factorization is required. For ordinary ridge regression $L = I$. More generally, L may leave an intercept unpenalized, penalize high-order features more strongly, or encode differences between neighboring tile weights to encourage smoothness. This linear least-squares development and the noisy-label interpretation follow Bertsekas (2019, Sec. 3.1.2).

The weights and the sample locations are not merely numerical details. They define which states matter to the regression. Let ρ be a probability distribution over states. The distribution-weighted error

$$\left\|\tilde{J} - J\right\|_\rho^2 = \mathbb{E}_{x\sim\rho}\left[(\tilde{J}(x) - J(x))^2\right]. \tag{3.21}$$

has the same squared-cost units as the pointwise error. It is not a new optimal control objective; it is a compact statement of where prediction fidelity is being measured. If the states $x^{(s)}$ are sampled independently from ρ, the unweighted sample mean

$$\frac{1}{q}\sum_{s=1}^{q}\left(\tilde{J}(x^{(s)}) - J(x^{(s)})\right)^2 \tag{3.22}$$

estimates Equation (3.21). A nominal-driving distribution may assign very little probability to near-collision states, so a small nominal error can coexist with a large safety-boundary error. Changing the sampling distribution or the weights generally changes the fitted function even when the architecture is unchanged.

Implementation note

Normalize continuous inputs before using polynomial, radial, or neural features. Otherwise the units of a state component can dominate the conditioning of $\Phi^\top\Phi$ and the gradient scale. The physical units must still be retained in the model definition and in reported control metrics; normalization is an internal numerical transformation.

3.4.2 What regularization selects—and what it does not fix

When many parameters fit the observed labels similarly, regularization states which fitted member is preferred. In Equation (3.18), the data term asks for agreement with the supplied targets and $\mathcal{R}$ encodes an additional preference. Increasing λ strengthens that preference relative to the data fit. It can improve conditioning and validation performance, but it deliberately introduces bias and cannot correct biased Bellman, Monte Carlo, or expert labels.

Common explicit penalties include

$$\mathcal{R}_2(\theta) = \tfrac{1}{2}\left\|\theta - \theta_{\text{ref}}\right\|_2^2, \tag{3.23a}$$

$$\mathcal{R}_1(\theta) = \|\theta\|_1, \tag{3.23b}$$

$$\mathcal{R}_{\text{EN}}(\theta) = \lambda_1\|\theta\|_1 + \tfrac{\lambda_2}{2}\|\theta\|_2^2, \tag{3.23c}$$

$$\mathcal{R}_{\text{smooth}}(\theta) = \|L\theta\|_2^2. \tag{3.23d}$$

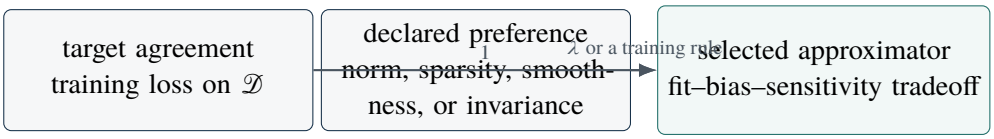
The reference in Equation (3.23a) need not be zero: it may be an initial controller or a trusted nominal parameter. The ℓ_1 penalty can select sparse coefficients but is nonsmooth. Elastic-net regularization combines shrinkage and sparsity. The operator L can penalize differences between neighboring tiles, high-order coefficients, or other declared roughness rather than parameter magnitude itself.

Control-oriented penalties may act on the represented function rather than on raw weights. Two examples are

$$\mathcal{R}_{\text{Jac}}(\theta) = \frac{1}{q}\sum_{s=1}^{q}\left\|\nabla_x\widetilde{\mu}(x^{(s)};\theta)\right\|_{\text{F}}^2, \qquad \mathcal{R}_{\Delta u}(\theta) = \frac{1}{q}\sum_{s=1}^{q-1}\left\|\widetilde{\mu}(x^{(s+1)};\theta) - \widetilde{\mu}(x^{(s)};\theta)\right\|^2. \tag{3.24}$$

The first discourages excessive state-to-action sensitivity on sampled states; the second discourages action variation along a declared trajectory ordering. Neither is a stability or actuator-rate certificate. Both can suppress a genuinely sharp control response if their coefficient is too large.

Not all regularization appears as an additive penalty. Early stopping selects an iterate before the training loss is minimized. Dropout randomly removes units during neural training and approximates an ensemble-like averaging effect (Srivastava et al. 2014). Input or label perturbations encode a chosen noise model. Data augmentation encodes declared invariances. With adaptive optimizers, decoupled weight decay (AdamW) is not generally identical to adding an ℓ_2 term to the loss (Loshchilov and Hutter 2019). Consequently, a reproducible report should state the mechanism, coefficient or stopping rule, parameters excluded from the penalty, and the validation distribution—not merely say "regularization was used."



Validation must still test target quality, decision quality, distribution shift, and closed-loop constraints.

Figure 3.5: Original regularization map. Regularization selects among fitted models by adding a declared preference; it is not a substitute for validating the targets or the resulting controller.

3.4.3 Stochastic labels and repeated targets

If $\mathbb{E}[\varepsilon^{(s)} \mid x^{(s)}] = 0$, squared-error regression targets the conditional mean $\mathbb{E}[\beta \mid x] = J(x)$. Suppose M labels at one state are conditionally independent and have the same conditional variance. Their average satisfies

$$\bar{\beta}(x) = \frac{1}{M}\sum_{j=1}^{M} \beta_j(x), \qquad \operatorname{Var}[\bar{\beta}(x) \mid x] = \frac{1}{M}\operatorname{Var}[\beta(x) \mid x]. \tag{3.25}$$

The second identity follows because variances of conditionally independent terms add and the prefactor is $1/M^2$. It is therefore an elementary probability result, while its use with sampled cost labels is part of the least-squares training framework in Bertsekas (2019, Secs. 3.1.2–3.1.3). It does not apply unchanged to correlated rollouts that reuse the same disturbance realization or initial trajectory.

Variance reduction costs additional simulation or experiments. If label variance changes with state, inverse-variance weighting can improve statistical efficiency under its assumptions. A different choice—upweighting rare safety-critical states—expresses control importance instead. Reliability weighting and importance weighting should not be conflated, and the adopted rule must be reported.

3.4.4 Target error and fitting error are distinct

Suppose the available label is an approximation $\hat{J}$ rather than the desired J. Then

$$\tilde{J} - J = (\tilde{J} - \hat{J}) + (\hat{J} - J). \tag{3.26}$$

The first term is discrepancy from the supplied target and contains representation, finite-data, and optimization effects. The second term is the target error itself. These terms need not be orthogonal, so squared errors do not simply add; the always-valid norm statement is the triangle bound

$$\left\|\tilde{J} - J\right\|_\rho \le \left\|\tilde{J} - \hat{J}\right\|_\rho + \left\|\hat{J} - J\right\|_\rho . \tag{3.27}$$

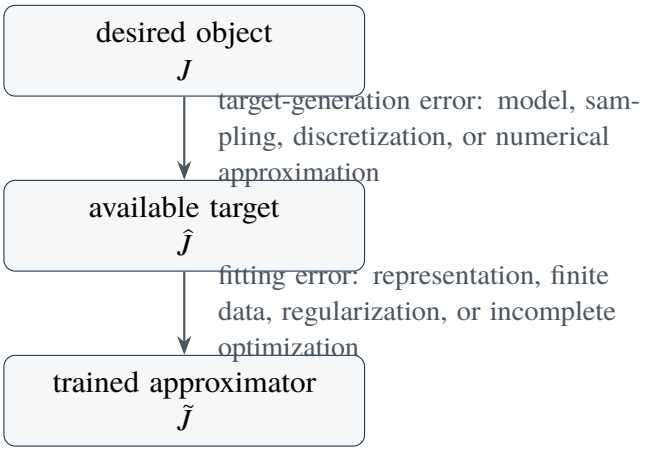


Figure 3.6: Original error-lineage diagram. Validation against the available target $\hat{J}$ examines only the right-hand link. It cannot by itself detect bias already present in $\hat{J}$.

For example, a neural network can interpolate every value produced by a coarse-grid DP calculation and still disagree with the continuous physical problem because the grid, action discretization, or model is inaccurate. Likewise, fitting Monte Carlo labels can be statistically accurate for the behavior-policy return but irrelevant to an optimal-value claim. Independent reference calculations, Bellman checks, or closed-loop evaluation are needed to investigate the left-hand link in Figure 3.6. Chapter 4 will show how target error is refreshed and sometimes propagated by approximate DP.

3.5 Nonlinear and Neural Architectures

A *multilayer perceptron* (MLP) is a feedforward neural network composed of affine maps and componentwise nonlinear activations. The term is standard, although a modern MLP usually has many units and is not limited to the historical single perceptron. For a fixed-dimensional vector-valued mobility state it is the simplest neural baseline. Other neural architectures remain valid value, Q-factor, policy, and model approximators; their main difference is the structure

imposed on the input and on the learned features. The feedforward and multilayer construction follows Bertsekas (2019, Secs. 3.2–3.2.2).

Let $z(x)$ be a normalized input and define

$$h_0 = z(x), \tag{3.28}$$

$$h_j = \sigma_j(W_j h_{j-1} + b_j), \qquad j = 1, \dots, L, \tag{3.29}$$

$$\tilde{J}(x;\theta) = w^{\mathsf{T}} h_L + b, \qquad \theta = \{W_j, b_j, w, b\}. \tag{3.30}$$

For fixed hidden-layer parameters, this is again a linear feature architecture with features h_L. Neural training also adjusts the feature extractor.

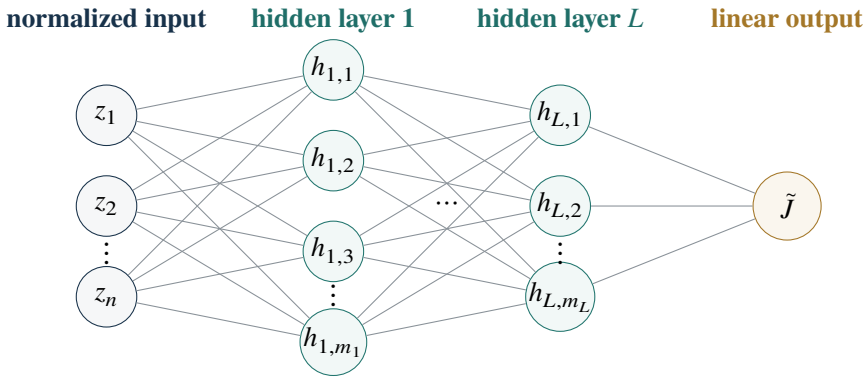


Figure 3.7: Original MLP diagram for a scalar value approximation. Every arrow has a trainable weight; every hidden unit also has a bias and activation. The last hidden vector $h_L(x)$ is a learned feature vector, while the final layer forms $w^{\mathsf{T}} h_L + b$.

Rectified-linear, smooth rectified-linear, and hyperbolic-tangent activations make different smoothness and extrapolation choices. No activation is best for every control problem. For example, a bounded tanh hidden layer may be numerically convenient on a declared operating envelope but can saturate outside it; an unbounded rectified-linear network can extrapolate with large piecewise-linear slopes.

Terminology note

Beyond an MLP: CNNs and RNNs as approximators. A convolutional neural network (CNN) is appropriate when an observation has a spatial or grid structure whose local patterns should share parameters: a camera image, occupancy grid, road raster, or spatial traffic field. A recurrent neural network (RNN), including gated variants such as an LSTM or GRU, processes an observation–control history and maintains a learned hidden state. It is useful when the physical state is only partially observed or when a fixed-length history would otherwise be supplied to an MLP. Attention and graph networks provide analogous structure for long sequences and interacting agents.

These labels describe the *approximation architecture*, not a new DP or RL algorithm. For example, DQN may use a CNN to approximate the same finite- action Q-factor that an MLP

approximates from a low-dimensional state. An RNN critic may approximate $Q(h_k, u_k)$ from a history summary h_k. The recurrent hidden state is not automatically a sufficient Markov state: its adequacy must still be checked through prediction, control performance, and distribution shift. Chapters 4 and 6 attach Bellman and policy-improvement targets to these architectures.

3.5.1 Loss, backpropagation, and parameter updates

Specializing Equation (3.18) to equal sample weights, training may minimize

$$\mathscr{L}(\theta) = \frac{1}{q}\sum_{s=1}^{q}\left(\tilde{J}(x^{(s)};\theta) - \beta^{(s)}\right)^2 + \lambda\mathscr{R}(\theta). \tag{3.31}$$

Here $\lambda \geq 0$ is the regularization coefficient and $\mathscr{R}(\theta)$ is a declared penalty. A common weight-decay choice is

$$\mathscr{R}(\theta) = \sum_{j=1}^{L}\|W_j\|_{\mathrm{F}}^2 + \|w\|^2, \tag{3.32}$$

usually with biases excluded. Other penalties encode different assumptions; the symbol $\mathscr{R}$ does not specify one automatically.

Backpropagation applies the chain rule to compute $\nabla_\theta \mathscr{L}$. It is not itself the rule that updates θ. Full-batch gradient descent uses

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \mathscr{L}(\theta_t), \tag{3.33}$$

where $\eta_t > 0$ is a stepsize. For a minibatch $\mathscr{B}_t \subset \{1, \dots, q\}$, define

$$g_t = \frac{1}{|\mathscr{B}_t|}\sum_{s\in\mathscr{B}_t}\nabla_\theta \ell_s(\theta_t) + \lambda\nabla_\theta\mathscr{R}(\theta_t), \quad \ell_s(\theta) = \left(\tilde{J}(x^{(s)};\theta) - \beta^{(s)}\right)^2. \tag{3.34}$$

One sample gives stochastic or incremental gradient; several samples give a minibatch update. Random reshuffling by epoch avoids repeatedly privileging the same samples. Ordinary and incremental gradient, stepsize, and Newton-type choices are developed in Bertsekas (2019, Sec. 3.1.3).

Adam augments the minibatch gradient with exponential first- and second-moment estimates (Kingma and Ba 2015):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2)(g_t \odot g_t), \tag{3.35}$$

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \widehat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{3.36}$$

$$\theta_{t+1} = \theta_t - \eta_t \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \delta_A}. \tag{3.37}$$

Here $0 \leq \beta_1, \beta_2 < 1$, $\delta_A > 0$ prevents division by zero, and $\odot$, the square root, and the division are componentwise. Adam changes the scaling and momentum of the update; it does not repair

biased targets, guarantee global minimization of a nonconvex loss, or make dependent validation data independent.

The five update families compared in Table 3.3 can be written using the same notation. Let $F(\theta) = q^{-1}\sum_{s=1}^{q}\ell_s(\theta) + \lambda\mathcal{R}(\theta)$ be the full objective and let g_t denote the gradient estimate available at iteration t. Representative updates are

$$\text{full batch:} \quad \theta_{t+1} = \theta_t - \eta_t \nabla F(\theta_t), \tag{3.38a}$$

$$\text{incremental/SGD:} \quad \theta_{t+1} = \theta_t - \eta_t(\nabla \ell_{s_t}(\theta_t) + \lambda \nabla \mathcal{R}(\theta_t)), \tag{3.38b}$$

$$\text{minibatch:} \quad \theta_{t+1} = \theta_t - \eta_t \left(\frac{1}{|\mathcal{B}_t|} \sum_{s\in\mathcal{B}_t} \nabla \ell_s(\theta_t) + \lambda \nabla \mathcal{R}(\theta_t) \right), \tag{3.38c}$$

$$\text{Adam:} \quad \theta_{t+1} = \theta_t - \eta_t \frac{\widehat{m}_t}{\sqrt{\hat{v}_t} + \delta_A}, \tag{3.38d}$$

$$\text{curvature based:} \quad \theta_{t+1} = \theta_t - \eta_t (H_t + \delta_H I)^{-1} g_t. \tag{3.38e}$$

Here s_t is the sample selected for an incremental step, $\mathcal{B}_t \subset \{1, \dots, q\}$ is a minibatch, and H_t is a Hessian or a positive-semidefinite curvature approximation. Damping $\delta_H > 0$ improves conditioning and makes the linear solve well posed when H_t is singular. Equations (3.35)– (3.37) define the moment states suppressed in the compact Adam line. Thus the five methods differ in how they estimate and scale a descent direction; they do not change the training target itself.

Table 3.3: What changes across common training updates.

Method	Data use and main tradeoff
Full-batch gradient	Uses all q samples for a relatively stable direction but is expensive when q is large.
Incremental/SGD	Uses one sample for a cheap, noisy update; sampling order and stepsize matter.
Minibatch gradient	Uses a subset $\mathcal{B}_t$ to combine vectorized computation with controllable gradient noise.
Adam	Uses a minibatch and moment states for adaptive coordinate scaling and momentum. It adds hyperparameters and has no universal superiority guarantee.
Curvature-based	Gauss–Newton and Newton-type updates add a curvature or curvature approximation. Local progress may improve at the cost of memory and linear algebra.

3.5.2 Output structure should match the control object

The final layer should respect how the result will be used:

- a value or scalar Q-factor uses a scalar output;

- a finite-action Q-network may use one output per action;
- a bounded continuous actor may use $u = u_c + u_s \tanh(z_\mu(x))$;
- a stochastic actor outputs distribution parameters, including a positive scale through an exponential or softplus map;
- a positive-definite action curvature in Equation (3.15) needs a factorization that enforces $H(x) \succ 0$.

These choices encode useful structure but do not by themselves guarantee closed-loop stability, constraint satisfaction, calibrated uncertainty, or optimality.

3.6 Data Distribution and Control-Relevant Validation

This section combines two sources of reasoning. Distribution-weighted approximation norms and the distinction between approximation error and induced policy performance are standard in approximate DP (Bertsekas 2019; Bertsekas 2025). The trajectory-level and shifted-condition split below is an authorial validation recommendation for correlated mobility data, not a theorem attributed to those books.

Let ρ_{train}, ρ_{val}, and ρ_{op} denote the training, validation, and closed-loop operating state distributions. Define errors with the same physical units,

$$E_{\text{train}} = \left\|\tilde{J} - J\right\|_{\rho_{\text{train}}}, \qquad E_{\text{op}} = \left\|\tilde{J} - J\right\|_{\rho_{\text{op}}}. \tag{3.39}$$

It is possible that $E_{\text{train}} \ll E_{\text{op}}$. Unlike the earlier statement "less than 1," this comparison is meaningful without pretending that a dimensioned cost error has an absolute unit-free threshold. The result is not a paradox: the two norms weight different states. In mobility control, rare states near a road boundary, collision set, friction limit, or actuator saturation may matter more than their frequency in nominal data suggests.

3.6.1 Random rows are not always independent validation data

Consecutive samples from one trajectory share initial conditions, disturbances, controller behavior, and often nearly identical states. A random row split can therefore place almost duplicate samples in training and validation sets. Prefer splits by complete trajectory, route, driver, operating condition, or random seed. A separate shifted test should vary the physical quantities that may change in deployment, such as speed, friction, mass, delay, or disturbance statistics.

3.6.2 Small value error and correct action are different claims

For a finite action set, define the optimal action $u^*(x) \in \arg\min_u Q^*(x,u)$ and let $\hat{u}(x)$ minimize $\widetilde{Q}(x,u)$. The following elementary bound explains why a Q-factor must be judged through action comparisons.

The one-step *decision regret*, also called the reference suboptimality gap of the chosen action, is

$$\mathcal{R}_{\text{dec}}(x) = Q^*(x, \hat{u}(x)) - \min_u Q^*(x,u) \geq 0. \tag{3.40}$$

It answers: "How much worse is the approximate greedy action when both actions are evaluated by the same reference Q-factor?" It is zero for any reference-optimal action, including ties, and it has the units of the Q-factor. In bandit and online-learning texts, cumulative regret sums analogous losses over repeated decisions (Lattimore and Szepesvári 2020). Here the qualifier *one-step* is essential: Equation (3.40) is neither an emotional notion nor a guarantee on full closed-loop excess cost.

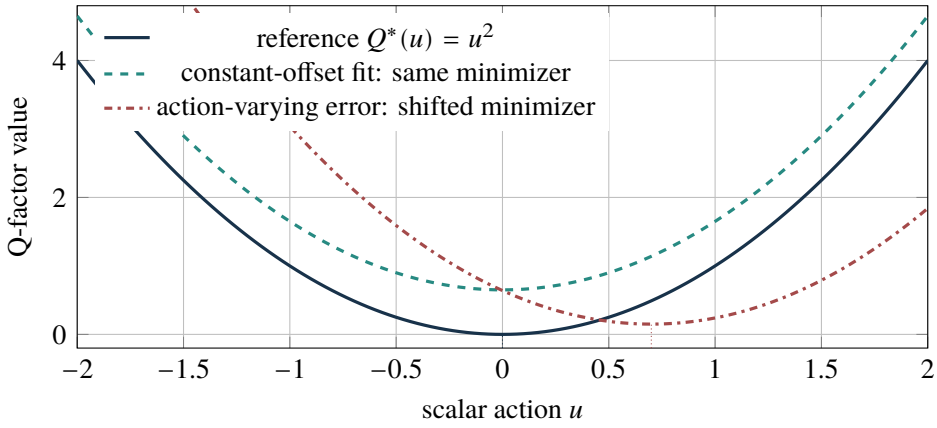


Figure 3.8: Original illustration of the control-relevant error observation in Bertsekas (2019, Sec. 2.1). A constant bias can produce nonzero value error without changing the greedy action. An action-dependent error can move the minimizer; its variation or slope near competing actions matters.

Proposition 3.1 (Q-error and greedy decision regret). *At a fixed state x, suppose*

$$\max_{u \in \mathcal{U}(x)} \left| \widetilde{Q}(x,u) - Q^*(x,u) \right| \le \epsilon. \tag{3.41}$$

Then the greedy action from $\widetilde{Q}$ satisfies

$$Q^*(x,\hat{u}) - Q^*(x,u^*) \le 2\epsilon. \tag{3.42}$$

Define the smallest strictly suboptimal action gap by

$$\Delta(x) = \min_{u:\, Q^*(x,u) > Q^*(x,u^*)} \left[Q^*(x,u) - Q^*(x,u^*) \right], \tag{3.43}$$

with $\Delta(x) = +\infty$ if every feasible action is optimal. If $\Delta(x) > 2\epsilon$, every approximate greedy action is optimal.

Proof. Greediness of $\hat{u}$ and the uniform error bound give

$$\begin{aligned} Q^*(x,\hat{u}) &\le \widetilde{Q}(x,\hat{u}) + \epsilon \\ &\le \widetilde{Q}(x,u^*) + \epsilon \\ &\le Q^*(x,u^*) + 2\epsilon. \end{aligned}$$

If $\hat{u}$ were strictly suboptimal when $\Delta(x) > 2\epsilon$, the left side of Equation (3.42) would be larger than 2ϵ, a contradiction. This argument allows multiple optimal actions. □

The converse does not hold: a large common offset in all action values can produce poor RMSE but preserve every greedy action. Conversely, a small error concentrated near action-switching boundaries can change many actions. For a continuous action set, local gradients and curvature replace the discrete action gap, and a small scalar value error alone is even less informative.

No single metric is sufficient. Action agreement is easy to interpret but counts a harmless near-tie and a severe mistake equally. One-step decision regret grades the severity using a declared reference Q-factor but inherits any error in that reference model and horizon. Closed-loop excess cost and constraint metrics include future state-distribution changes, but they are more expensive and scenario dependent. The mobility example reports the first two and treats closed-loop testing as the next validation layer.

Terminology note

Deep RL combines a deep function approximator with an RL target and data generation process. DQN is not just a neural network: it is a neural Q-factor architecture together with sample-based Bellman targets, replay, target networks, and other stabilization choices. PPO, TD3, and SAC similarly add specific policy-evaluation and policy-improvement rules. Their algorithmic roles are developed in Chapters 4 and 6.

3.7 Mobility Case: A Car-Following Value Surrogate

We now fit a continuation value in a small longitudinal-control problem. The purpose is not to claim a high-fidelity car-following model. It is to keep the target, architecture, data support, and induced decision fully auditable.

3.7.1 Declared model and reference target

Let $e_d = d - d_{\text{ref}}$ be spacing error and let $e_v = v_{\text{lead}} - v_{\text{ego}}$ be relative speed. Thus $e_v < 0$ means that the ego vehicle is closing on the lead vehicle. With a constant-speed lead vehicle and zero-order-held ego acceleration u, the teaching model is

$$e_{d,k+1} = e_{d,k} + T_s e_{v,k} - \tfrac{1}{2}T_s^2 u_k, \tag{3.44}$$

$$e_{v,k+1} = e_{v,k} - T_s u_k. \tag{3.45}$$

The horizon is $N = 20$, $T_s = 0.25\,\text{s}$, and

$$u_k \in \{-3, -1.5, 0, 1.5, 3\}\ \text{m/s}^2. \tag{3.46}$$

For $d_{\text{ref}} = 15\,\text{m}$ and $d_{\text{safe}} = 5\,\text{m}$, the stage cost is

$$g(e_d, e_v, u) = 0.05e_d^2 + 0.30e_v^2 + 0.035u^2 + 12[d_{\text{safe}} - (d_{\text{ref}} + e_d)]_+^2. \tag{3.47}$$

The terminal weights on e_d^2, e_v^2, and the risk hinge are 0.8, 1.5, and 40, respectively.

Backward DP is run on 61 spacing-error points, 49 relative-speed points, and the five controls. Off-grid successors use bilinear interpolation. The result is therefore a *reference numerical target* J_1^{ref}, not an exact solution of a continuous physical problem. The known model is used both to construct every target and to perform the one-step action comparison.

3.7.2 Architectures and data support

Four architectures are fitted to J_1^{ref}:

1. **Quadratic fixed features.** With $z_d = e_d/12$ and $z_v = e_v/6$, this model uses
$$\phi_{\text{Q}} = \begin{bmatrix} 1 & z_d & z_v & z_d^2 & z_d z_v & z_v^2 \end{bmatrix}^{\mathsf{T}}. \tag{3.48}$$
It is the smallest and most globally structured candidate.
2. **Structured fixed features.** This model augments ϕ_{Q} with a soft proximity indicator and a closing-speed indicator,
$$h = \left[\frac{-8 - e_d}{4}\right]_+, \qquad c = [-z_v]_+, \qquad \phi_{\text{S}} = \begin{bmatrix} \phi_{\text{Q}}^{\mathsf{T}} & h & h^2 & hc & h^2 c \end{bmatrix}^{\mathsf{T}}. \tag{3.49}$$
The activation threshold $e_d = -8$ m deliberately precedes the declared -10 m safety boundary; it is a feature-design margin, not a hard constraint.
3. **RBF fixed features.** A 7×5 array of centers covers the normalized state support. The model uses an intercept, the two affine terms, and 35 Gaussian features of the form Equation (3.12) with normalized width 0.38.
4. **Shallow MLP.** A one-hidden-layer network with 24 tanh units learns its features from normalized (e_d, e_v). The scalar output is linear and the weights are trained by minibatch Adam with held-out early stopping.

The four names in bold are the canonical short names used in every table and figure. All models fit the same standardized target and are transformed back to the original cost units before evaluation. Hence their metric differences come from architecture and training, not from different labels or units.

The declared training support is

$$-10.5 \le e_d \le 10, \qquad |e_v| \le 4.75. \tag{3.50}$$

There are 1521 training states and 507 held-out states inside this support. The remaining 961 grid states form a shifted test. This split is intentionally geometric rather than random over the entire grid, so extrapolation is visible.

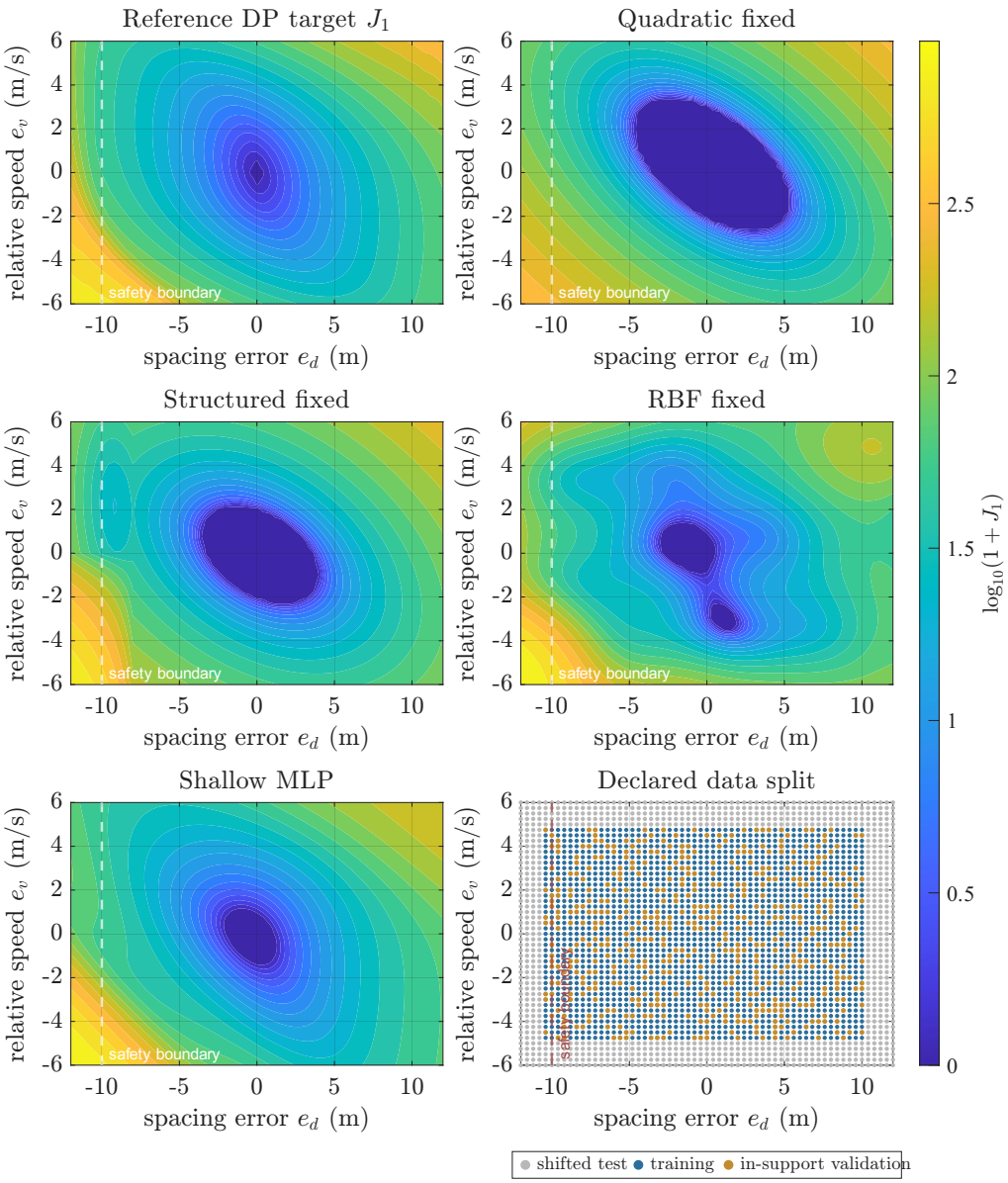


Figure 3.9: Original car-following continuation-value comparison. Five panels use the common $\log_{10}(1+J_1)$ color scale: Reference DP target, Quadratic fixed, Structured fixed, RBF fixed, and Shallow MLP. The sixth panel makes the training, in-support validation, and shifted-test geometry explicit. None of the four fits reproduces the reference surface everywhere outside the declared training support.

3.7.3 Prediction quality and decision quality

Each fitted continuation value is inserted into the same one-step model-based minimization used by the reference calculation. In this experiment, Equation (3.40) becomes

$$\mathcal{R}^{\mathrm{ref}}_{\mathrm{dec}}(x) = Q^{\mathrm{ref}}_0(x, \hat{u}(x)) - \min_u Q^{\mathrm{ref}}_0(x, u). \tag{3.51}$$

Thus prediction and decision are evaluated against the same declared numerical problem.

Table 3.4: Verified Chapter 3 approximation results. 'Val.' is held out inside the training support; 'shift' is outside it. Action match is the percentage of states with the same selected acceleration as the reference DP calculation.

Architecture	RMSE val.	RMSE shift	Action match val.	Action match shift
Quadratic fixed	31.916	116.415	84.6%	93.1%
Structured fixed	17.286	65.463	83.4%	89.9%
RBF fixed	8.321	61.713	66.1%	77.4%
Shallow MLP	5.531	40.991	88.8%	88.9%

Table 3.5: Mean one-step decision regret in reference cost units. Lower is better; zero means that the selected action is reference optimal.

Architecture	In-support validation	Shifted test
Quadratic fixed	0.0294	0.0257
Structured fixed	0.0459	0.0845
RBF fixed	0.6572	9.7711
Shallow MLP	0.1318	0.6629

The Shallow MLP has the smallest value RMSE in both test regions, but the Quadratic fixed model has the highest shifted action agreement and the smallest mean decision regret: 0.0294 inside the support and 0.0257 on the shifted set. The Shallow MLP's corresponding regrets are 0.1318 and 0.6629. The RBF fixed model fits values well inside the support yet has substantially poorer action agreement and a shifted mean regret of 9.7711.

This result is not evidence that extrapolation is harmless. Extreme states in this small problem often have a wide action margin and a saturated optimal acceleration, so a rough quadratic surface can preserve the ordering of the five actions despite a large common value error. Near switching boundaries, the same model makes visible action mistakes. The experiment illustrates the logic of proposition 3.1: prediction ranking and action margins must accompany RMSE.

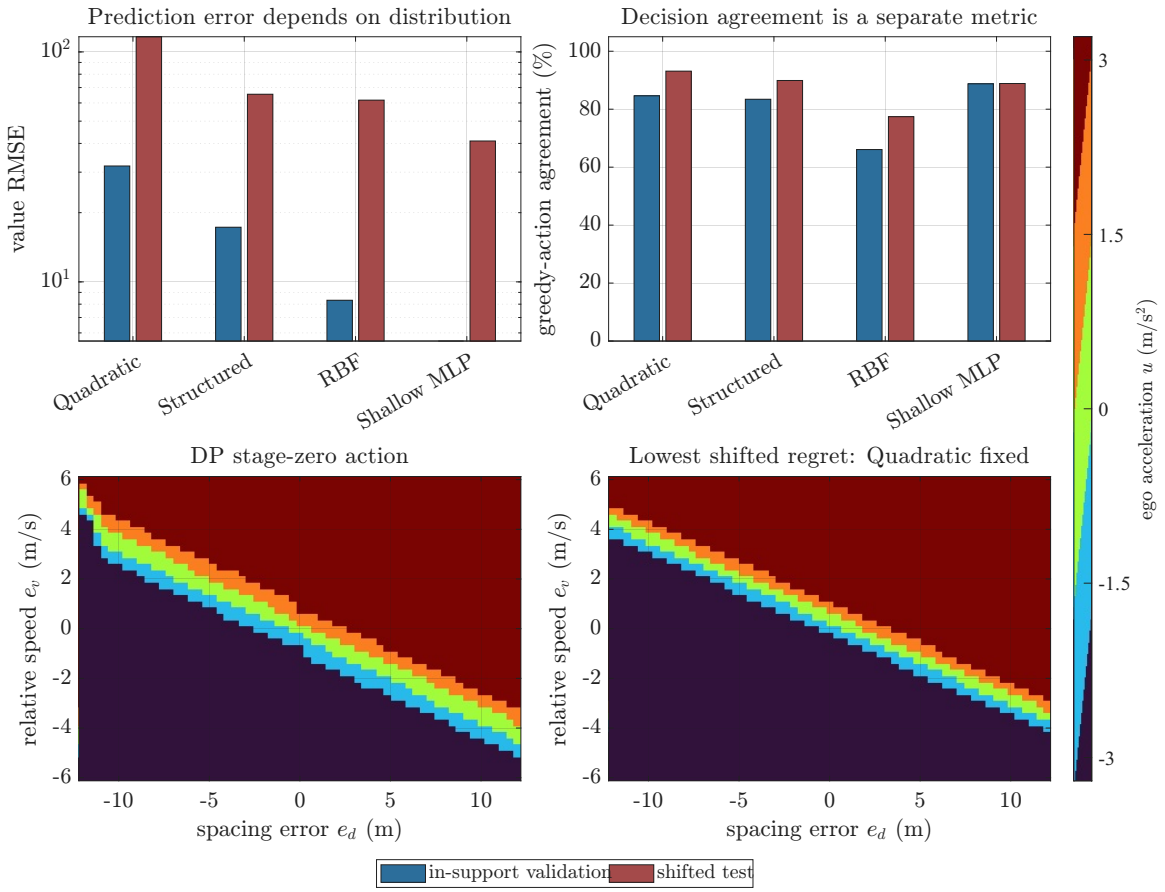


Figure 3.10: Prediction and control-facing validation for the car-following experiment. The upper panels separate in-support and shifted tests. The lower panels compare the reference stage-zero action with the architecture selected by the smallest shifted decision regret. Large-action-margin regions can retain the correct decision even when value RMSE is large.

Implementation note

The experiment is generated by `code/ch03_parametric_approximation/generate_figures.m` using base MATLAB only. It writes the value and action figures, a metric table, all state predictions, and a numerical summary to `figures/generated/`. The code is suitable for a guided architecture-comparison exercise; a graded lab should add a starter version, unit tests, and prescribed held-out scenarios.

3.8 From Fixed Targets to Approximate DP

This chapter held the target fixed to expose architecture and fitting. In approximate DP, target generation becomes part of the algorithm. The same regression machinery can be attached to different targets:

Table 3.6: How target construction changes the meaning of the same approximator.

Training target	Resulting interpretation
Reference value $J(x)$	Supervised value approximation, as in this chapter.
Bellman target $T\tilde{J}^{(i)}$	Fitted or approximate value iteration.
Policy return from trajectories	Monte Carlo policy evaluation.
Bootstrapped one-step target	Temporal-difference policy evaluation.
Sampled Q-Bellman target	Q-learning, SARSA, or DQN depending on target and data choices.
Actions from an expert or optimizer	Policy approximation by regression or classification.

For example, a sampled one-step Bellman residual has the form

$$\delta(r; x, u, w) = g(x, u, w) + \alpha \tilde{J}(f(x, u, w); r) - \tilde{J}(x; r). \tag{3.52}$$

Minimizing δ^2 is not the same problem as fitting oracle labels $J(x)$. The target depends on the current approximation, and its sampling distribution is generated by a behavior policy. These feedback effects are the subject of Chapter 4.

The same distinction appears in model learning. A one-step dynamics loss fits x_{k+1}, whereas a multistep loss may use

$$\mathscr{L}_H(r_f) = \sum_s \sum_{h=1}^{H} \omega_h \left\| \hat{x}_{s+h|s}(r_f) - x_{s+h} \right\|^2. \tag{3.53}$$

Longer prediction losses expose compounding error but are more expensive and couple the rollout to the input sequence. Chapter 7 develops this issue for learned mobility dynamics.

3.9 Evaluation Before Controller Deployment

"Fit quality" is a common descriptive phrase, but "control readiness" is not used here as a standardized technical term or as terminology attributed to the cited Bertsekas texts. The durable question is whether the approximation has been evaluated for its intended controller role. The following authorial checklist will be reused in later chapters.

Conditions to check

1. **Object:** Is the fit a value, Q-factor, policy, or model?
2. **Target source:** Exact calculation, numerical DP, real samples, simulator returns, or learned-model rollouts?
3. **Model use:** Was an explicit known or learned model used during training, execution, both, or neither?
4. **Support:** What state–action and operating-condition region is represented in the data?
5. **Split:** Are complete trajectories and shifted physical conditions held out?
6. **Prediction metrics:** Are units, weighting distribution, and rare critical regions reported?
7. **Decision metrics:** Are greedy-action agreement, regret, closed-loop cost, constraints, and computation time checked?
8. **Execution:** Does the stored object require a model, optimizer, actor, or safety filter online?

This checklist does not demand that every architecture be neural or highly expressive. A small structured model can be preferable when it preserves the relevant action ordering, extrapolates transparently, and evaluates reliably inside an online optimization.

3.10 Summary

- An architecture, fitting method, and DP/RL target-generation algorithm are distinct design layers.
- Linear feature architectures can use nonlinear state features and offer transparent regularized least-squares training.
- A finite-action policy can be fitted directly as a classifier; its softmax output may define a randomized policy or a deterministic maximum-probability action, but it is not a Q-factor.
- Regularization selects a fitted member by adding a declared preference. It may improve conditioning or generalization, but it cannot repair biased targets or replace closed-loop validation.
- Neural networks learn the feature map as well as the last-layer weights; they are approximation architectures rather than complete controllers.

- Backpropagation computes gradients; batch gradient, SGD, minibatch methods, Adam, and Newton-type methods use those derivatives in different parameter updates.
- Fitting an available target accurately does not establish that the target itself is accurate for the desired control problem.
- The sampling and weighting distribution define the norm in which a fit is accurate. Trajectory-level and shifted validation are essential for mobility data.
- Uniform Q-error bounds greedy decision regret, but low value RMSE alone does not guarantee action agreement.
- In the declared car-following experiment, the architecture with the smallest value RMSE is not the one with the smallest one-step decision regret; this is a case result, not a universal ranking of architectures.
- Chapter 4 will attach Bellman, Monte Carlo, temporal-difference, and Q-learning targets to these architectures.

3.11 Exercises

Exercise 3.1 (Feature matrix and regularized fit). For samples $(x^{(s)}, \beta^{(s)})$ and features $\phi(x) = [1, x, x^2]^\mathsf{T}$, construct Φ and derive Equation (3.20). Show how the solution changes when the intercept is excluded from the regularization matrix L.

Exercise 3.2 (Scaling and conditioning). Fit a quadratic function of speed in m/s and distance in meters. Repeat after expressing speed in km/h. Compare the condition number of $\Phi^\mathsf{T}\Phi$ before and after standardizing both inputs. Explain why the physical prediction should be invariant even though the numerical problem is not.

Exercise 3.3 (Finite-action policy classifier). Let $\mathcal{U} = \{-1, 0, 1\}$ and suppose an online optimizer labels a collection of states with its minimizing action. Write a three-class linear-softmax policy, its cross-entropy loss, and its deterministic deployment rule. Explain why the three scores cannot be interpreted as Q-factors without an additional calibration claim.

Exercise 3.4 (Regularization is a design choice). For a tile-coded policy, compare ordinary ridge regularization with a first- difference penalty $\|L\theta\|^2$. State which one shrinks all action levels and which one primarily discourages changes between neighboring tiles. Give one closed-loop situation in which excessive smoothing would be harmful.

Exercise 3.5 (MLP forward map and parameter count). For the car-following Shallow MLP with two inputs, 24 tanh hidden units, and one linear output, write the complete forward map and verify that it has 97 scalar parameters including biases. State which 24-dimensional quantity

is the learned feature vector in Figure 3.7.

Exercise 3.6 (Minibatch and Adam update)**.** For a two-parameter quadratic loss, calculate one full-batch gradient step and one minibatch step. Starting with $m_0 = v_0 = 0$, then calculate the first Adam step from Equations (3.35) and (3.37). Explain why bias correction matters at $t = 1$ and why Adam does not change a biased training label.

Exercise 3.7 (Noisy target averaging)**.** Suppose a Monte Carlo value label has variance $\sigma^2(x)$. Derive the variance of an average of $M(x)$ independent returns. Under a fixed total simulation budget, propose a rule for allocating more returns to high-variance or safety-critical states and state which regression norm the rule induces.

Exercise 3.8 (Action gap)**.** Prove proposition 3.1 without assuming a unique optimal action. Construct a three-action numerical example in which value RMSE is large but the greedy action is always correct, and another in which small RMSE changes the greedy action near a tie.

Exercise 3.9 (Car-following architecture comparison)**.** Run `code/ch03_parametric_approximation/generate_figures.m`. Change the training support, RBF width, and number of neural hidden units one at a time. Report in-support and shifted RMSE, action agreement, decision regret, and computation time. Do not select a model using RMSE alone.

Exercise 3.10 (Trajectory-level validation)**.** Generate car-following trajectories under three different baseline controllers. Compare a random row split with a split that holds out one entire controller and one lead-vehicle acceleration profile. Explain which estimate is more representative of deployment under a changed policy.

Exercise 3.11 (Discrete and continuous Q architectures)**.** For five discrete accelerations, design a shared-state network with five Q outputs. Then design a critic for continuous acceleration using Equation (3.15). Compare their action-selection cost, representational restrictions, and ability to impose acceleration bounds.

Exercise 3.12 (From supervised fit to fitted value iteration)**.** Replace the fixed labels in the car-following experiment by targets $T\tilde{J}^{(i)}$ generated from the preceding fit. Track validation error, Bellman residual, action agreement, and changes in the training distribution. Identify which new difficulties arise because the target is no longer fixed.

4 Approximation in Value Space

Chapter 2 established exact value and Q-factor equations and previewed their approximate forms. Chapter 3 then separated the approximation architecture from the data used to fit it. This chapter joins those two layers. Bellman, Monte Carlo, and temporal- difference constructions now generate targets for a value or Q-factor approximator, and the resulting object is used for policy improvement.

The organizing question is not merely "Which reinforcement-learning algorithm should be used?" It is

What value-space object is approximated, how is its target generated, and how does it produce a control?

The answer distinguishes fitted value iteration from policy evaluation, Q-learning from SARSA, and a stored critic from an executable controller.

Learning objectives

After studying this chapter, the reader should be able to:

1. connect the exact and approximate VI/PI equations of Chapter 2 to fitted and sample-based algorithms;
2. distinguish representation error, Bellman-target error, sampling error, and policy-improvement error;
3. derive Monte Carlo, TD(0), multistep, Q-learning, and SARSA updates using the book's cost-minimization convention;
4. explain why Q-learning is an asynchronous sampled Q-value-iteration method and why SARSA is an on-policy evaluation–improvement method;
5. interpret replay, target networks, Double DQN, and dueling networks as modifications of approximate Q-value iteration;
6. state what tabular convergence results do and do not imply for neural approximators;
7. use an approximate terminal value in rollout or multistep lookahead and interpret the resulting performance bound;
8. relate value-space approximation to policy-space approximation, actor–critic methods, and online planning; and
9. audit a learned value method using policy cost, constraints, coverage, and computation rather than training loss alone.

4.1 From an Approximation Architecture to an Approximate DP Method

A neural network, polynomial basis, or table specifies where a numerical function can be stored. It does not specify which function should be stored. The same architecture $\tilde{J}(x;\theta)$ can be trained toward an exact reference value, a Monte Carlo return, a one-step TD target, or an optimality Bellman target. Those choices lead to different algorithms and different claims.

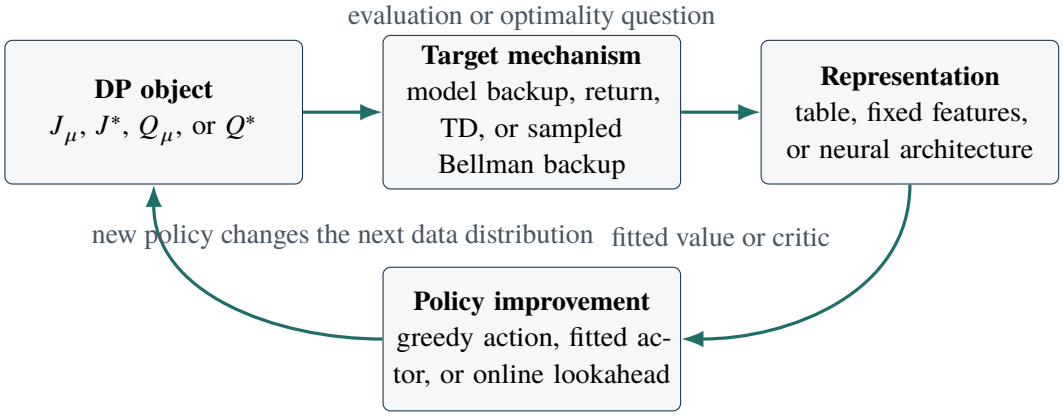


Figure 4.1: Original architecture of a value-space method. Chapter 3 focused on the representation block. This chapter focuses on the target and improvement loops. A method is not fully specified until all four blocks are declared.

Table 4.1 locates the main methods before their derivations. "Model use" refers to the backup or target computation, not to whether a model was used elsewhere to design features, simulate data, or audit the final policy.

AI/RL terminology bridge

In common AI language, a value network or Q-network is a *critic*. A critic trained for a fixed policy estimates J_μ or Q_μ; a Q-network trained by an optimality target aims at Q^*. Calling both objects "the value" hides an essential difference. Likewise, a critic is not necessarily an actor. A finite-action Q critic defines an implicit actor by an arg min, whereas a state-value critic still needs a model-based successor- state comparison or a separate policy network.

Chapter roadmap

The chapter follows the two loops already present in exact DP. The *optimality loop* approximates a Bellman minimum directly; fitted VI and Q-learning belong to this branch. The *policy-iteration loop* first evaluates a declared policy and then improves it; MC, TD, approximate PI, and rollout belong to this branch. DQN modifies the Q-learning implementation, whereas online lookahead changes how the learned value is used at execution.

Table 4.1: A first map of value-space methods.

Method	Stored object	Target source	Control produced
Fitted VI	$\tilde{J} \approx J^*$	Model-based Bellman optimality target, then regression	Model-based greedy minimization
Approximate PI	$\tilde{J}_\mu$ or $\widetilde{Q}_\mu$	Policy returns or policy Bellman targets	Greedy policy or fitted actor
MC evaluation	$\tilde{J}_\mu$	Sampled multistep return under μ	None until improvement is added
TD evaluation	$\tilde{J}_\mu$	Bootstrapped sample under μ	None until improvement is added
Q-learning and DQN	$\widetilde{Q} \approx Q^*$	One-step TD optimality error, using a sampled Q-Bellman target	Finite-action greedy policy
SARSA	$\widetilde{Q}_\mu$ while μ changes	Sampled next action under the behavior policy	On-policy greedy/exploratory policy
Rollout and lookahead	terminal $\tilde{J}$ plus online tree	Explicit model or sampled forward search	Root action of online optimization

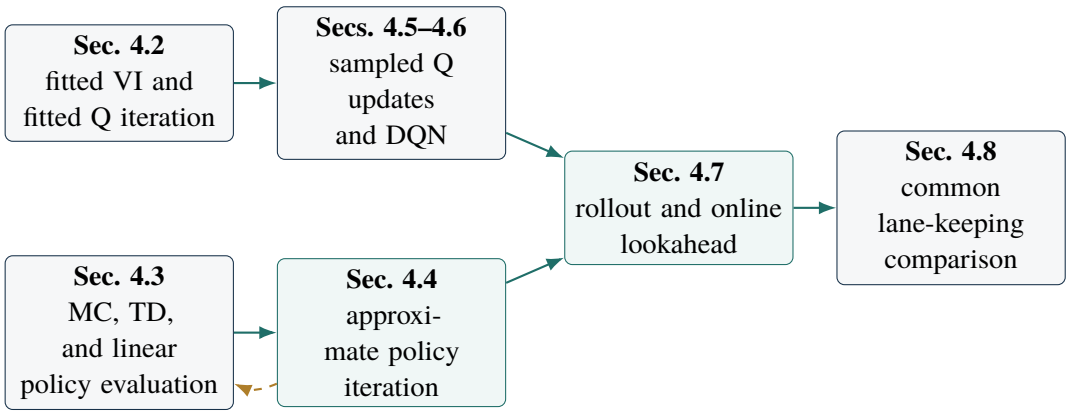


Figure 4.2: Original chapter map. The upper branch approximates the optimality operator directly. The lower branch makes policy evaluation and policy improvement explicit. Both branches can supply a terminal value or a base policy for online improvement.

This map is also a reading guide. Section 4.3 does not interrupt the path to the main idea in Section 4.4: it supplies the critic required by approximate policy iteration. Section 4.8 then revisits both branches on one model so that algorithm names are not confused with differences in the physical task.

4.2 Fitted Value Iteration: Bellman Backup Followed by Projection

4.2.1 The generic fitted-VI step

Retain the finite discounted stochastic model of assumption 2.3. Exact value iteration applies $J^{(i+1)} = TJ^{(i)}$ at every state. Suppose instead that values must lie in a class

$$\mathcal{F} = \{\tilde{J}(\cdot;\theta) : \theta \in \Theta\}. \tag{4.1}$$

Let Π_{ν_i} denote a fitting operation using a training-state distribution ν_i. A generic fitted or approximate VI step is

$$\tilde{J}^{(i+1)} = \Pi_{\nu_i} T\tilde{J}^{(i)}. \tag{4.2}$$

The notation is compact, but it represents two distinct computations.

First, construct a Bellman target. Its deterministic and stochastic forms are

$$y_i^{\mathrm{det}}(x) = \min_{u\in\mathscr{U}(x)} \left\{ g(x,u) + \alpha\tilde{J}^{(i)}(f(x,u)) \right\}, \tag{4.3}$$

$$y_i^{\mathrm{sto}}(x) = \min_{u\in\mathscr{U}(x)} \left\{ \bar{g}(x,u) + \alpha \sum_{x'} p(x' \mid x,u)\tilde{J}^{(i)}(x') \right\}. \tag{4.4}$$

Second, fit the target on samples $x^{(s)} \sim \nu_i$:

$$\theta_{i+1} \in \arg\min_{\theta} \sum_{s=1}^{q_i} \omega_i^{(s)} \left| \tilde{J}(x^{(s)};\theta) - y_i(x^{(s)}) \right|^2 + \lambda_i \mathscr{R}(\theta). \tag{4.5}$$

The target depends on the previous approximator, so this is not a static supervised-learning problem. Approximation error can be regenerated at every iteration and can alter the greedy action used in the next target.

Equations (4.3)– (4.4) are *model-based target constructions*. The deterministic form requires the successor map f; the stochastic form requires the transition probabilities and expected stage cost. Both also require minimizing over the feasible actions at every sampled training state. A learned model or a simulator may replace an analytic model, but then model-estimation or Monte Carlo error must be added to the regression error. The sample-only alternatives that observe one successor without forming the expectation are developed in Sections 4.3 and 4.5.

Algorithm lens: Fitted value iteration

Choose an initial approximator and a training-state design. At iteration i, evaluate or sample an optimality Bellman target on the selected states; fit a new approximator; then check parameter change, Bellman residual, held-out target error, and the induced policy. A small change between successive fits does not imply a small Bellman residual: the projected iteration may have reached a fixed point of $\Pi_\nu T$ rather than of T.

4.2.2 Approximate Q-value iteration

The Q-factor version uses the operator introduced in Section 2.8.5. For a finite stochastic model,

$$(T_Q Q)(x,u) = \bar{g}(x,u) + \alpha \sum_{x'} p(x' \mid x,u) \min_{v \in \mathscr{U}(x')} Q(x',v). \tag{4.6}$$

Fitted Q iteration is

$$\widetilde{Q}^{(i+1)} = \Pi_{Q,\rho_i} T_Q \widetilde{Q}^{(i)}, \tag{4.7}$$

where ρ_i is a state–control sampling distribution. With a finite action set, the stored approximation immediately yields

$$\widetilde{\mu}_i(x) \in \arg\min_u \widetilde{Q}^{(i)}(x,u). \tag{4.8}$$

This execution advantage explains why Q-factor approximation is prominent in model-free RL. It does not remove the action minimization for a continuous control set; Chapter 6 introduces a separate actor for that case.

4.2.3 How one-step approximation errors accumulate

The next proposition separates Bellman contraction from implementation error. It is an elementary manuscript derivation obtained by perturbing the standard discounted VI contraction, rather than a separately named theorem. The conceptual source is the approximate-VI development in Bertsekas (2019, Sec. 4.4) and Bertsekas (2025, Sec. 3.3.1). It is not a convergence guarantee for an arbitrary regression algorithm.

Proposition 4.1 (Perturbed value-iteration bound). *Suppose*

$$\tilde{J}^{(i+1)} = T\tilde{J}^{(i)} + e_i \quad \text{and} \quad \|e_i\|_\infty \le \varepsilon_i \tag{4.9}$$

under assumption 2.3. Then

$$\left\|\tilde{J}^{(n)} - J^*\right\|_\infty \le \alpha^n \left\|\tilde{J}^{(0)} - J^*\right\|_\infty + \sum_{i=0}^{n-1} \alpha^{n-1-i} \varepsilon_i. \tag{4.10}$$

If $\varepsilon_i \le \bar{\varepsilon}$, the second term is at most $\bar{\varepsilon}(1-\alpha^n)/(1-\alpha)$.

Proof. The contraction property gives

$$\left\|\tilde{J}^{(i+1)} - J^*\right\|_\infty \leq \alpha \left\|\tilde{J}^{(i)} - J^*\right\|_\infty + \varepsilon_i.$$

Recursive substitution proves the stated bound. □

The bound uses a uniform error measured over all states. A least-squares projection usually controls a distribution-weighted norm instead. Rare but control-critical states can therefore have large errors even when the training loss is small. Moreover, $\Pi_\nu T$ need not be a contraction in the fitted norm. These are structural limitations, not merely optimizer tuning issues. The fitted and approximate DP development here follows the value-space viewpoint in Bertsekas (2019) and Bertsekas (2025).

4.3 Policy Evaluation from Sampled Trajectories

Value iteration changes the policy implicitly through the minimum in every target. Policy evaluation asks a different question: for a fixed policy μ, estimate

$$J_\mu(x) = \mathbb{E}_\mu\left[\sum_{t=0}^{\infty} \alpha^t g(x_t, u_t, w_t) \,\middle|\, x_0 = x\right]. \tag{4.11}$$

This is the critic step of policy iteration. The data distribution is induced by the initial-state distribution, the policy, and the dynamics; it is not automatically the distribution required for safe policy improvement.

4.3.1 Monte Carlo returns

For a sampled trajectory, an N-step return with a terminal estimate is

$$G_k^{(N)} = \sum_{t=0}^{N-1} \alpha^t g_{k+t} + \alpha^N \hat{J}_T(x_{k+N}). \tag{4.12}$$

Here $\hat{J}_T$ is a declared terminal evaluator; the subscript prevents it from being confused with the policy value currently being learned. With $\hat{J}_T = 0$, a sufficiently long trajectory approximates the full discounted return. With $\hat{J}_T \approx J_\mu$, shorter rollouts can be used, at the price of terminal-target bias. A tabular every-visit update is

$$\tilde{J}_\mu(x_k) \leftarrow \tilde{J}_\mu(x_k) + \eta_k(G_k^{(N)} - \tilde{J}_\mu(x_k)). \tag{4.13}$$

For a parameterized value, replace the tabular coordinate by the gradient $\nabla_\theta \tilde{J}(x_k; \theta)$ as in Chapter 3.

The return uses actual sampled costs and transitions, so neither MC nor the TD methods below require an analytic transition model. In this precise backup sense they are *model free*. They can still consume trajectories produced by a model-based simulator, and a later greedy improvement from a state-value critic may again require a model.

The plain return is on-policy: it evaluates the policy that generated the actions. If trajectories were generated by a behavior policy μ_{beh} but the target is another stochastic policy μ, a trajectory likelihood ratio is

$$\varrho_{k:k+N-1} = \prod_{t=k}^{k+N-1} \frac{\mu(u_t \mid x_t)}{\mu_{\text{beh}}(u_t \mid x_t)}. \tag{4.14}$$

One simple off-policy regression weights $G_k^{(N)}$ by this ratio, or equivalently uses $\varrho_{k:k+N-1}$ as a sample weight. This is *importance weighting*: it changes averages under the behavior distribution into averages under the target distribution. It requires support—$\mu_{\text{beh}}(u \mid x) > 0$ whenever $\mu(u \mid x) > 0$—and products of many ratios may have enormous variance. Per-decision ratios, clipping, or specialized off-policy estimators trade variance against bias; they should be declared rather than hidden inside the word "replay." Long returns reduce bootstrapping bias but can have high variance and delay updates.

4.3.2 One-step temporal difference

TD(0) replaces the unknown continuation value by the current approximator:

$$\underbrace{y_k^{\text{TD}}}_{\text{one-step TD target}} = g_k + \alpha \tilde{J}_\mu(x_{k+1}; \theta_k), \tag{4.15}$$

$$\delta_k = y_k^{\text{TD}} - \tilde{J}_\mu(x_k; \theta_k), \tag{4.16}$$

$$\theta_{k+1} = \theta_k + \eta_k \delta_k \nabla_\theta \tilde{J}_\mu(x_k; \theta_k). \tag{4.17}$$

Here k is both a physical sample time and an update index because one update is made from each transition. The update is often called a semi-gradient: the bootstrap target depends on θ_k, but its derivative is not followed in Equation (4.17). TD can update before a trajectory terminates and often has lower target variance than a long return, but it introduces bootstrap bias and can propagate errors from successor states.

Like MC, this update uses the observed tuple (x_k, u_k, g_k, x_{k+1}) rather than $p(x' \mid x, u)$, so on-policy TD evaluation is model free in its target construction. Unlike MC, it bootstraps. Unlike Q-learning, it does not contain an optimality minimum and therefore does not improve a policy by itself.

Worked Example 4.1 (A common trajectory viewed by MC and TD). Consider a fixed policy on two states with deterministic transitions $x^0 \to x^1$ at cost 1 and $x^1 \to x^1$ at cost 2, with $\alpha = 0.9$. Its exact values are

$$J_\mu(x^1) = \frac{2}{1 - 0.9} = 20, \qquad J_\mu(x^0) = 1 + 0.9(20) = 19.$$

Suppose the current critic has $\tilde{J}_\mu(x^1) = 10$. The one-step TD target from x^0 is $1 + 0.9(10) = 10$. A three-step MC return with zero terminal value is

$$1 + 0.9(2) + 0.9^2(2) = 4.42,$$

which is strongly biased downward by artificial truncation. Using the exact terminal value

after those three costs gives

$$1 + 0.9(2) + 0.9^2(2) + 0.9^3(20) = 19.$$

The same observed trajectory therefore supports different targets depending on the bootstrap and terminal convention. TD needs no transition matrix, but its target inherits the current critic error. Truncated MC also needs no transition matrix, but its target inherits terminal-value error. Neither target changes the policy until an improvement step is added.

4.3.3 The continuum between MC and TD(0)

An n-step TD target is

$$G_k^{(n)} = \sum_{t=0}^{n-1} \alpha^t g_{k+t} + \alpha^n \tilde{J}_\mu(x_{k+n}; \theta). \tag{4.18}$$

Small n emphasizes bootstrapping; large n approaches a Monte Carlo return. The forward-view λ-return combines all depths,

$$G_k^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_k^{(n)}, \qquad 0 \le \lambda < 1, \tag{4.19}$$

with an episodic terminal modification when the trajectory ends. Eligibility traces provide an online backward-view implementation. For a differentiable critic, an accumulating trace and its semi-gradient update are

$$z_k = \alpha\lambda z_{k-1} + \nabla_\theta \tilde{J}_\mu(x_k; \theta_k), \qquad z_{-1} = 0, \tag{4.20}$$

$$\theta_{k+1} = \theta_k + \eta_k \delta_k z_k. \tag{4.21}$$

The trace assigns the current TD error backward to recently active features. At $\lambda = 0$, z_k contains only the current gradient, so Equation (4.21) is exactly TD(0). As λ approaches one in a terminating task, the forward target approaches the Monte Carlo return, subject to the chosen terminal convention. Thus TD(0) is not a separate family from TD(λ); it is its one-step endpoint.

There is also an operator interpretation. Define

$$T_\mu^{(\lambda)} J = (1-\lambda) \sum_{n=0}^{\infty} \lambda^n T_\mu^{n+1} J. \tag{4.22}$$

Linear TD(λ), LSTD(λ), and LSPE(λ) can be viewed as different numerical routes toward the projected equation $\Phi r = \Pi_\nu T_\mu^{(\lambda)}(\Phi r)$. Larger λ reduces the amount of bootstrapping and can improve the projected-operator geometry, but commonly raises sampling variance and trace length. The MC, TD, and trace formulations follow the standard development in Sutton and Barto (2018, Chaps. 6–7); the signs here use cost minimization rather than reward maximization. The projected-equation view and the comparison of TD, LSTD, and LSPE follow Bertsekas (2019, Sec. 4.9).

Terminology note

"Bias versus variance" is useful but incomplete. A one-step target can have low conditional variance and still be systematically wrong because its bootstrap value is wrong. A long Monte Carlo return can be unbiased for the behavior policy yet irrelevant to a different target policy. Coverage, partial observability, truncation, and policy mismatch must be reported along with target variance.

4.3.4 Linear TD, LSTD, and projected equations

For $\tilde{J}_\mu(x;r) = \phi(x)^\top r$, there are two conceptually different fitting routes. A *direct* method treats sampled returns as regression labels,

$$r_{\mathrm{MC}} \in \arg\min_r \sum_{k=0}^{q-1} \left(\phi(x_k)^\top r - G_k^{(N)}\right)^2. \tag{4.23}$$

An *indirect* TD method instead tries to make the approximation consistent with its own one-step continuation. Incremental TD(0) becomes

$$r_{k+1} = r_k + \eta_k \phi(x_k)\left[g_k + \alpha\phi(x_{k+1})^\top r_k - \phi(x_k)^\top r_k\right]. \tag{4.24}$$

This equation is the simple critic update that is easily obscured by a deep-RL software stack: compute one scalar TD error, multiply by the active feature vector, and add the result to the parameters.

Under declared on-policy sampling and regularity conditions, the associated projected fixed-point question is

$$\Phi r = \Pi_\nu T_\mu(\Phi r). \tag{4.25}$$

The empirical least-squares TD (LSTD) normal equation is

$$A_q = \sum_{k=0}^{q-1} \phi_k(\phi_k - \alpha\phi_{k+1})^\top + \lambda_{\mathrm{reg}} I, \qquad b_q = \sum_{k=0}^{q-1} \phi_k g_k, \tag{4.26}$$

$$r_{\mathrm{LSTD}} = A_q^{-1} b_q, \tag{4.27}$$

where $\phi_k = \phi(x_k)$. Here the symbol in the equation is $\lambda_{\mathrm{reg}} \geq 0$, distinct from the trace parameter in TD(λ). In code one solves the linear system rather than explicitly forming A_q^{-1}. LSTD solves an empirical projected Bellman equation; it is not ordinary least squares on fixed independent labels because the successor feature appears in the system matrix.

Least-squares policy evaluation (LSPE) instead performs a preconditioned projected Bellman iteration. One batch form is

$$C_q = \sum_{k=0}^{q-1} \phi_k \phi_k^\top + \lambda_{\mathrm{reg}} I, \tag{4.28}$$

$$r_{i+1} = r_i + \eta_i C_q^{-1} \sum_{k=0}^{q-1} \phi_k \left[g_k + \alpha\phi_{k+1}^\top r_i - \phi_k^\top r_i\right]. \tag{4.29}$$

Unlike LSTD, LSPE retains an iteration index and can warm-start the evaluation of a nearby policy. Unlike incremental TD, it uses an accumulated feature covariance to scale the residual. All three use the same one-step policy Bellman relation; they differ in how much sampled information is accumulated before a parameter update.

Table 4.2: Four linear policy-evaluation routes for the same feature class.

Method	Equation targeted	Main computation	Characteristic tradeoff
MC regression	projection of a sampled return	least squares on $G_k^{(N)}$	no bootstrap; return variance and truncation
TD(0)	stochastic projected fixed point	one rank-one sample update	low memory; stepsize and ordering sensitivity
LSTD(0)	empirical projected fixed point	solve $A_q r = b_q$	no learning-rate schedule; matrix storage and conditioning
LSPE(0)	projected Bellman iteration	scaled batch residual steps	warm start and iteration control; repeated linear solves

These methods preserve the separation learned in Chapter 3: the architecture is linear, while the target mechanism is MC or TD. For $\lambda > 0$, the same distinction remains after eligibility-weighted samples replace the one-step statistics. Detailed convergence claims require the sampling chain, feature rank, weighting distribution, and on/off-policy setting to be stated; the equations alone do not supply them.

4.4 Approximate Policy Iteration

Exact PI in Chapter 2 alternated an exact solution of $J_{\mu_i} = T_{\mu_i} J_{\mu_i}$ and an exact greedy improvement. That algorithm is often impractical for a large or unknown system: solving for all policy values is expensive, and a newly improved policy may not have a compact representation. Approximate policy iteration (API) retains the same two-step logic while allowing either step to be imperfect:

$$\text{evaluation:} \quad \tilde{J}_{\mu_i} \approx J_{\mu_i} \quad \text{or} \quad \widetilde{Q}_{\mu_i} \approx Q_{\mu_i}, \tag{4.30}$$

$$\text{improvement:} \quad \mu_{i+1}(x) \in \arg\min_u \left\{ \bar{g}(x,u) + \alpha \sum_{x'} p(x' \mid x,u) \tilde{J}_{\mu_i}(x') \right\}, \tag{4.31}$$

or, with a Q-factor critic,

$$\mu_{i+1}(x) \in \arg\min_u \widetilde{Q}_{\mu_i}(x,u). \tag{4.32}$$

Evaluation targets can come from model equations, Monte Carlo returns, TD updates, or combinations of them. If the improved policy is too large to store, actions from Equation (4.31) can become labels for a parameterized actor. That last regression is approximation in *policy space*; it is developed in Chapter 6.

This explains the placement of Section 4.3. MC and TD are not competing controllers; they are alternative implementations of the evaluation block inside API. A complete sample-based API loop is

1. collect trajectories under μ_i or a declared behavior policy;
2. fit $\tilde{J}_{\mu_i}$ or $\widetilde{Q}_{\mu_i}$ using MC, TD, LSTD, or another evaluator;
3. form an improved policy from a model-based J minimization, a direct Q minimization, or a fitted actor;
4. independently audit the candidate and either accept it or collect more data.

The model classification can therefore differ between blocks. For example, TD evaluation is sample based and model free, while the improvement in Equation (4.31) explicitly uses a transition model. Q-factor API can make both blocks sample based because the critic already stores action comparisons.

4.4.1 Evaluation error, improvement error, and the API error region

Two different inaccuracies must not be collapsed into one training loss. Assume a uniform evaluation error

$$\left\|\tilde{J}_{\mu_i} - J_{\mu_i}\right\|_\infty \le \delta \tag{4.33}$$

and an approximate-greedy error

$$T_{\mu_{i+1}}\tilde{J}_{\mu_i} \le T\tilde{J}_{\mu_i} + \epsilon\mathbf{1}. \tag{4.34}$$

Exact minimization gives $\epsilon = 0$. A restricted maneuver search, an incompletely optimized continuous action, or a fitted actor can make $\epsilon > 0$ even with a perfect critic.

Proposition 4.2 (Discounted API performance region)**.** *For the finite discounted model of assumption 2.3, suppose Equations* (4.33) *and* (4.34) *hold at every API iteration. Then the generated policies satisfy*

$$\limsup_{i\to\infty} \left\|J_{\mu_i} - J^*\right\|_\infty \le \frac{\epsilon + 2\alpha\delta}{(1-\alpha)^2}. \tag{4.35}$$

If the policy sequence eventually becomes constant, with policy $\bar{\mu}$, the stronger bound

$$\left\|J_{\bar{\mu}} - J^*\right\|_\infty \le \frac{\epsilon + 2\alpha\delta}{1-\alpha} \tag{4.36}$$

holds.

Proof idea. Insert $\tilde{J}_{\mu_i}$ between J_{μ_i} and the two Bellman operators. Contraction contributes at most $\alpha\delta$ on each side of the greedy comparison, while approximate minimization contributes ϵ. Resolving the resulting policy-loss recursion produces one factor $1/(1-\alpha)$; allowing policies to keep changing produces a second factor. If the policy becomes constant, its fixed-policy equation closes the argument without that second accumulation. A full derivation and worst-case discussion are given in Bertsekas (2019, Sec. 4.6.3); the notation here has been converted to the conventions of this book. □

The first bound describes an *error region*, not convergence of the policies. Even with a finite policy set, API may alternate among policies whose evaluation errors reverse a few close action comparisons. This is one reason a low critic RMSE does not imply monotone policy improvement. The second bound explains why an observed stable policy is materially different from an oscillating sequence, although stability alone still does not prove optimality. The factors involving $(1-\alpha)^{-1}$ also show why long-horizon problems demand either small uniform errors or additional structure; the bound is intentionally conservative and may be much larger than empirical loss.

There is also a local interpretation. If $\bar{\mu}$ is exactly greedy with respect to $\tilde{J}_\mu$ and $\|\tilde{J}_\mu - J_\mu\|_\infty \leq \delta$, then

$$T_{\bar{\mu}} J_\mu \leq J_\mu + 2\alpha\delta\mathbf{1}, \tag{4.37}$$

$$J_{\bar{\mu}} \leq J_\mu + \frac{2\alpha\delta}{1-\alpha}\mathbf{1}. \tag{4.38}$$

Unlike exact policy improvement, these inequalities do not guarantee $J_{\bar{\mu}} \leq J_\mu$. They quantify how evaluation error can consume the improvement margin. A candidate should therefore be evaluated independently rather than accepted because its critic loss decreased.

4.4.2 Why API is not merely fitted VI with different labels

Fitted VI repeatedly approximates the optimality map $\tilde{J}_{i+1} \approx T\tilde{J}_i$. API instead freezes a policy long enough to approximate the affine policy map T_{μ_i}, and only then changes that policy. The distinction has practical consequences.

Neither method uniformly dominates the other. Policy evaluation can be easier than global optimality approximation because it follows one policy's trajectories, but improvement needs reliable comparisons for actions that the current policy may rarely take. Approximate VI avoids an explicit outer loop, but projection is applied to a moving optimality target and the projected map need not be a contraction. API makes the control step visible, which is useful when a mobility controller must restrict maneuvers, preserve a baseline, or audit every proposed policy.

4.4.3 The geometric connection to policy iteration and online play

The geometry developed in Chapter 5 gives a second way to read these equations. Each policy operator $T_\mu J$ is an affine surface in value space, and the optimality operator TJ is their lower envelope. VI moves from a current value to that envelope. Exact PI selects an active policy surface

Table 4.3: Direct approximate VI and approximate PI compared.

Question	Fitted/approximate VI	Approximate PI
Target object	optimal value J^* or Q^*	current-policy value J_{μ_i} or Q_{μ_i}
Target operator	nonlinear minimum in every backup	fixed-policy operator during evaluation
Data focus	states/actions needed by the optimality backup	occupancy and alternatives relevant to μ_i
Control update	implicit in every target	explicit improvement after evaluation
Typical risk	projection error is re-injected every optimality sweep	evaluation and improvement errors can cause policy oscillation
Useful structure	stable projected optimality update or rich coverage	accurate policy-specific evaluator and meaningful improvement margin

and jumps to its fixed point; this Newton-like step is the source of its often rapid improvement. API perturbs both operations: an approximate critic does not lie exactly at the selected policy fixed point, and an approximate actor may select the wrong active surface.

This view also explains why API is closely connected to rollout. A critic $\tilde{J}_{\mu_i}$ predicts the distant cost under the current policy. A one-step or multistep online minimization locally selects a better policy surface before the action is executed. If those improved actions are stored as labels for a new actor, online play becomes the improvement mechanism of an API loop. Chapter 5 develops this geometry extensively for VI, PI, LQR, and MPC; the present section supplies the stochastic and approximation-error interpretation needed to use it here.

4.4.4 Optimistic and simulation-based API

Full policy evaluation can be more expensive than another improvement. An optimistic API scheme uses a finite number of TD, Monte Carlo, or Bellman evaluation steps before updating the policy. This is the sample-based analogue of modified PI in Chapter 2. Three timescales then interact:

1. physical transitions generate data;
2. critic updates partially evaluate the current or behavior policy; and
3. policy updates change the distribution that will generate future data.

Fast policy changes can make critic targets stale; slow changes can waste data on a poor policy. The term "optimistic" here refers to incomplete policy evaluation in the Bertsekas terminology, not to optimistic initialization or upper-confidence exploration.

Approximate VI can be viewed as an extreme optimistic-PI construction: take only a small number of evaluation-style updates before changing the greedy policy again. SARSA and actor–critic implementations make a related timescale choice through the relative frequencies and

stepsizes of critic and actor updates. Stating this connection prevents "modified PI," "optimistic PI," and "online actor–critic" from appearing as unrelated algorithm families.

4.5 Sampled Q Updates: Q-Learning and SARSA

Q-factors absorb the one-step model comparison into a state–control function. Given a sampled transition (x_k, u_k, g_k, x_{k+1}), tabular Q-learning applies

$$y_k^Q = g_k + \alpha \min_v Q_k(x_{k+1}, v), \tag{4.39}$$

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \eta_k \left[y_k^Q - Q_k(x_k, u_k) \right], \tag{4.40}$$

and leaves all unvisited entries unchanged. The expected target conditioned on (x_k, u_k) equals $(T_Q Q_k)(x_k, u_k)$. Q-learning is therefore an asynchronous, sampled version of Q-value iteration (Watkins and Dayan 1992).

It is also a *temporal-difference control method*. Its TD optimality error is

$$\delta_k^Q = g_k + \alpha \min_v Q_k(x_{k+1}, v) - Q_k(x_k, u_k), \tag{4.41}$$

and Equation (4.40) is simply $Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \eta_k \delta_k^Q$. The word TD is not reserved for fixed-policy state-value evaluation: it denotes bootstrapping from a later estimate. What distinguishes Q-learning is that its bootstrap uses an *optimality* minimum rather than the action actually selected by the behavior policy.

SARSA samples the next action u_{k+1} from the current behavior policy and uses

$$y_k^{\mathrm{SARSA}} = g_k + \alpha Q_k(x_{k+1}, u_{k+1}), \tag{4.42}$$

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \eta_k \left[y_k^{\mathrm{SARSA}} - Q_k(x_k, u_k) \right]. \tag{4.43}$$

Expected SARSA replaces the sampled continuation action by an expectation under the behavior policy,

$$y_k^{\mathrm{ExpSARSA}} = g_k + \alpha \sum_v \mu_{\mathrm{beh}}(v \mid x_{k+1}) Q_k(x_{k+1}, v). \tag{4.44}$$

Thus SARSA is on-policy when the same evolving policy generates the data and defines the continuation action. Q-learning is off-policy relative to its exploratory behavior data because it targets a greedy continuation regardless of which next action the behavior policy takes. "Off-policy" does not mean "without data from a policy"; it means that the target policy and behavior policy differ. The distinction is summarized in Table 4.4.

AI/RL terminology bridge

Q-learning is called *model free* because its update needs a sampled next state and cost rather than an explicit transition matrix. This does not imply that no model influenced the simulator, feature design, data collection, or safety filter. Conversely, model-based fitted Q iteration and model-free Q-learning can use the same Q-network architecture;

Table 4.4: Sampled Q updates under the cost-minimization convention.

Method	Continuation target	Policy relation	Main interpretation
Q-learning	$\min_v Q(x', v)$	Off-policy relative to exploratory data	Sampled optimality backup
SARSA	$Q(x', u'), u' \sim \mu_{\text{beh}}$	On-policy	Sampled policy-evaluation backup while policy changes
Expected SARSA	$\sum_v \mu_{\text{beh}}(v \mid x') Q(x', v)$	On-policy expectation	Lower action-sampling variance with known behavior probabilities

they differ in how the Bellman target is evaluated.

4.5.1 What tabular convergence does and does not say

For a stationary finite discounted MDP with bounded one-stage costs, tabular Q-learning converges under standard stochastic-approximation conditions when every state–control pair is visited infinitely often and its stepsizes obey

$$\sum_{k \in \mathcal{K}(x,u)} \eta_k = \infty, \qquad \sum_{k \in \mathcal{K}(x,u)} \eta_k^2 < \infty. \tag{4.45}$$

Here $\mathcal{K}(x, u)$ is the set of update times for the pair (x, u). These conditions explain why persistent exploration and pair-specific visit counts appear in elementary implementations.

The statement does not transfer automatically to nonlinear function approximation. One parameter update changes many state–control estimates; data are correlated; targets move with the same parameters; and off-policy bootstrapping can be unstable. The combination of function approximation, bootstrapping, and off-policy learning is often called the *deadly triad*. It is a diagnostic warning, not a theorem that every such method must diverge.

4.6 Deep Q-Networks as Stabilized Fitted Q Iteration

A deep Q-network (DQN) replaces a table by $\widetilde{Q}(x, u; \theta)$ while retaining a finite action set. For a replayed transition (x, u, g, x'), the basic squared target loss is

$$y^{\text{DQN}} = g + \alpha \min_v \widetilde{Q}(x', v; \theta^-), \tag{4.46}$$

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,u,g,x') \sim \mathcal{D}} \left[\left(\widetilde{Q}(x, u; \theta) - y^{\text{DQN}} \right)^2 \right]. \tag{4.47}$$

The replay buffer $\mathcal{D}$ reuses past transitions and weakens their temporal ordering. The target parameters θ^- are held fixed for a declared interval or moved slowly, so the regression label does not change with every online-network update. These devices were central to the DQN implementation of Mnih et al. (2015); they improve optimization behavior but do not recreate the contraction proof for arbitrary neural training.

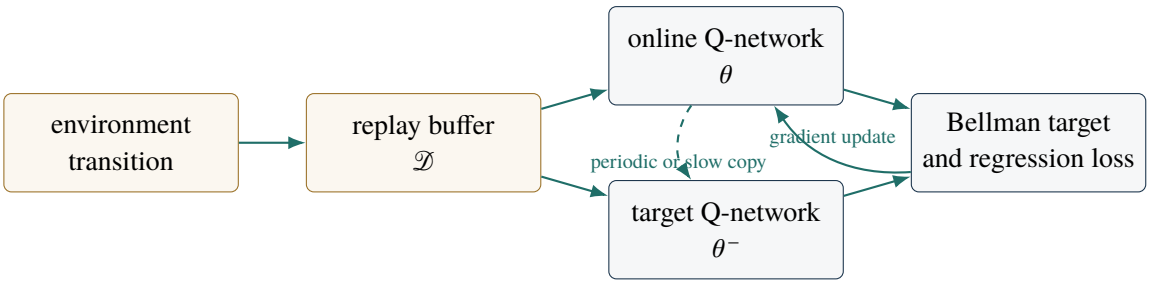


Figure 4.3: Original DQN computation diagram. Replay changes the data stream; the target network changes the label timescale. Neither changes the finite- action Q-Bellman equation being approximated.

4.6.1 Double and dueling modifications

In reward-maximizing notation, using the same estimates to select and evaluate a maximum can create positive maximization bias. Under the present cost-minimization convention, the analogous reuse can create a downward minimization bias. Double DQN separates selection and evaluation:

$$v^* \in \arg\min_v \widetilde{Q}(x', v; \theta), \tag{4.48}$$

$$y^{\text{Double}} = g + \alpha \widetilde{Q}(x', v^*; \theta^-). \tag{4.49}$$

The online network selects the action and the target network evaluates it (Hasselt et al. 2016).

The two lines have deliberately different jobs. In Equation (4.48), the online parameters answer the discrete decision question "which continuation action currently looks best?" In Equation (4.49), the delayed parameters attach a numerical cost to that already selected action. Basic DQN instead uses $\min_v \widetilde{Q}(x', v; \theta^-)$, so the same noisy target-network outputs both choose and score the minimum. Separation reduces, but does not eliminate, selection bias. It also does not create two independently trained optimal Q-functions; the target network remains a delayed copy.

For example, if the online outputs at x' are $(4, 3, 5)$ and the target outputs are $(2, 6, 7)$, Double DQN selects the second action and evaluates it as 6, whereas basic DQN uses the target-network minimum 2 from the first action. The discrepancy is not an algebra error: it is exactly the selection– evaluation separation. Under cost minimization, noisy minima tend to be too low; under the reward-maximizing convention, noisy maxima tend to be too high.

A dueling network changes the output architecture, not the Bellman target. It represents a state value and relative action advantages, for example

$$\widetilde{Q}(x,u) = V(x) + A(x,u) - \frac{1}{|\mathcal{U}|}\sum_{v} A(x,v). \tag{4.50}$$

The centering resolves the non-identifiability of adding a constant to V and subtracting it from every advantage. This architecture can share state-value information across actions when many actions have similar consequences (Wang et al. 2016).

To interpret the word *advantage* under cost minimization, define for a policy μ

$$A_\mu(x,u) = Q_\mu(x,u) - J_\mu(x). \tag{4.51}$$

It measures the excess cost of taking u once and then following μ, relative to following μ immediately. For a deterministic policy, $A_\mu(x,\mu(x)) = 0$; a negative value identifies a one-step improvement over the current policy. For the optimal functions,

$$A^*(x,u) = Q^*(x,u) - J^*(x) \geq 0, \qquad \min_u A^*(x,u) = 0. \tag{4.52}$$

Only differences across actions affect the greedy decision. This motivates approximating Q differences or advantages when a large state-dependent offset is harder to learn than the action ordering, an idea developed as advantage updating in Bertsekas (2025, Sec. 3.3.6) and in the earlier Q-factor treatment of Bertsekas (2019).

The dueling-network stream $A(x,u)$ should nevertheless be interpreted with care. Mean centering in Equation (4.50) is an identifiability choice; it does not enforce the policy identity $A_\mu(x,\mu(x)) = 0$ or the optimal identity $\min_u A^*(x,u) = 0$. Dueling DQN and advantage updating are related through their focus on action differences, but they are not identical algorithms.

Control viewpoint

DQN is natural for a finite maneuver or mode library: lane change left, keep lane, lane change right; charge, hold, discharge; or a finite set of motion primitives. A steering wheel or motor torque is physically continuous. A DQN controller for that variable either discretizes it or selects a finite primitive. Keeping a continuous action requires an inner minimization of a structured Q-function or a policy-space actor such as those developed in Chapter 6.

4.7 Online Improvement with a Terminal Value

The previous methods usually execute a stored greedy policy. A different use of a value approximation is to place it at the leaves of an online planning problem. Given a state x_k, an ℓ-step stochastic lookahead computes

$$\min_{\mu_0,\dots,\mu_{\ell-1}} \mathbb{E}\left[\sum_{t=0}^{\ell-1} \alpha^t g(x_{k+t}, \mu_t(x_{k+t}), w_{k+t}) + \alpha^\ell \tilde{J}(x_{k+\ell}) \,\middle|\, x_k\right] \tag{4.53}$$

and applies only the first control. In operator form its root value is $T^{\ell}\tilde{J}$. The terminal approximation compresses the distant future; the online Bellman steps recompute near-term actions using the current state and model.

Proposition 4.3 (Greedy policy induced by an approximate value). *Under assumption 2.3, suppose* $\|\tilde{J} - J^*\|_\infty \le \varepsilon$. *Let* $\tilde{\mu}$ *be greedy with respect to* $\tilde{J}$. *Then*

$$\|J_{\tilde{\mu}} - J^*\|_\infty \le \frac{2\alpha}{1-\alpha}\varepsilon. \tag{4.54}$$

If $\tilde{\mu}_\ell$ *is the first-stage policy from exact* ℓ*-step lookahead with terminal* $\tilde{J}$, *then*

$$\|J_{\tilde{\mu}_\ell} - J^*\|_\infty \le \frac{2\alpha^\ell}{1-\alpha}\varepsilon. \tag{4.55}$$

Proof. The greedy relation is $T_{\tilde{\mu}}\tilde{J} = T\tilde{J}$. Add and subtract the two operators evaluated at J^* to obtain

$$\|T_{\tilde{\mu}}J^* - J^*\|_\infty \le 2\alpha\varepsilon.$$

The residual bound for the contraction $T_{\tilde{\mu}}$ gives Equation (4.54). For ℓ-step lookahead, the first action is greedy with respect to $T^{\ell-1}\tilde{J}$, whose distance from J^* is no greater than $\alpha^{\ell-1}\varepsilon$. Substitution into the one-step result proves the second bound. □

The powers of α explain why exact lookahead can attenuate terminal- value error. They do not account for an approximate model, restricted action search, Monte Carlo tree error, or real-time truncation. Each of those adds a separate error source. Nor does the bound say that deeper search is always worth its computation.

4.7.1 One horizon notation for lookahead, rollout, and a terminal critic

Three depths are often mixed in prose. It is clearer to separate them:

1. the first ℓ stages are optimized online;
2. the next m stages follow a declared base policy μ_{base}; and
3. a learned or designed terminal evaluator $\hat{J}_T$ represents everything beyond those simulated stages.

The effective continuation value for the root decision is

$$\bar{J}_{\ell,m} = T^{\ell-1}T^m_{\mu_{\text{base}}}\hat{J}_T, \tag{4.56}$$

and the executed policy satisfies

$$T_{\mu_{\ell,m}}\bar{J}_{\ell,m} = T\bar{J}_{\ell,m}. \tag{4.57}$$

This compact operator expression covers several familiar designs without pretending that their computation is identical.

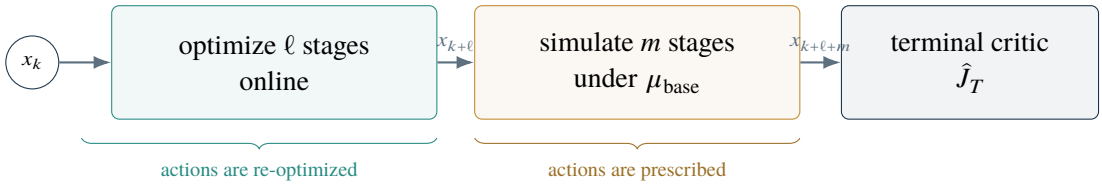


Figure 4.4: Original horizon decomposition for online value-space control. Pure lookahead, rollout, and actor-plus-critic search differ in which blocks are present and how their expectations are computed.

For a discounted policy operator, truncating the base-policy continuation has a transparent terminal-error relation:

$$\left\| T^m_{\mu_{\text{base}}} \hat{J}_T - J_{\mu_{\text{base}}} \right\|_\infty \leq \alpha^m \left\| \hat{J}_T - J_{\mu_{\text{base}}} \right\|_\infty . \tag{4.58}$$

Thus a longer base rollout attenuates a bounded terminal-critic error, but its simulation cost and stochastic variance grow. Additional optimized lookahead can then attenuate the remaining value error as in Equation (4.55). These statements assume exact operators; finite Monte Carlo samples, model mismatch, and pruned action sets contribute additional errors.

Table 4.5: Common online uses of a value-space approximation.

Variant	Root evaluator	What is optimized online	Main caution
One-step greedy	$\hat{J}_T$	current action only	state-value critic still requires successor prediction
ℓ-step lookahead	$T^{\ell-1}\hat{J}_T$	first ℓ actions/policies	branching and model error grow with depth
Pure rollout	$J_{\mu_{\text{base}}}$	current action against exact base value	exact infinite-horizon value is usually theoretical
Truncated rollout	$T^m_{\mu_{\text{base}}}\hat{J}_T$	current action; then m base steps	terminal bias and Monte Carlo variance must be reported
Combined search	$T^{\ell-1}T^m_{\mu_{\text{base}}}\hat{J}_T$	ℓ optimized stages; m base stages	computation, approximation, and search errors coexist

4.7.2 Rollout as policy improvement

Let μ_{base} be a base policy. Pure one-step rollout uses its exact value as the terminal evaluator:

$$\mu_{\text{ro}}(x) \in \arg\min_{u} \left\{ \bar{g}(x,u) + \alpha \sum_{x'} p(x' \mid x,u) J_{\mu_{\text{base}}}(x') \right\}. \tag{4.59}$$

By the exact policy-improvement theorem, $J_{\mu_{\text{ro}}} \leq J_{\mu_{\text{base}}}$. If $J_{\mu_{\text{base}}}$ is replaced by a finite Monte Carlo estimate or a learned critic, the exact monotonic guarantee is lost and Equation (4.38) becomes the relevant qualitative warning.

The exact $J_{\mu_{\text{base}}}$ in Equation (4.59) is an infinite-horizon mathematical object. It can be available analytically in a small finite model or for a structured controller such as LQR, but one cannot generally obtain it by literally simulating an infinite trajectory. A practical rollout therefore truncates after m base-policy steps and uses

$$\widehat{Q}_m(x,u) = \mathbb{E}\left[g_0 + \sum_{t=1}^{m} \alpha^t g_t + \alpha^{m+1} \hat{J}_T(x_{m+1}) \;\middle|\; x_0 = x, u_0 = u, u_t = \mu_{\text{base}}(x_t) \right]. \tag{4.60}$$

For deterministic dynamics, one trajectory evaluates each candidate action. For stochastic dynamics, the expectation is normally estimated with multiple rollouts. Using common disturbance samples across candidate actions can reduce the variance of their *differences*, which matters because the controller selects an ordering rather than an absolute value (Bertsekas 2025, Secs. 2.7.3–2.7.4).

Rollout is therefore finite horizon in its online implementation even when it approximates policy improvement for an infinite-horizon problem. The terminal critic closes the truncated horizon, and the controller repeats the calculation at the next physical state. This receding-horizon interpretation is the direct bridge to MPC.

Rollout provides a direct connection between offline RL and online control: the base policy supplies a default continuation, the critic scores leaf states, and an online model-based search improves the action actually sent to the system. Chapter 5 develops the geometry extensively and shows how the same relation appears in LQR and MPC.

Algorithm lens: Truncated stochastic rollout at one state

For each admissible root action, simulate the root transition and then follow μ_{base} for m steps. Add the discounted terminal critic, average repeated stochastic trajectories, and select the action with the smallest estimated cost. Apply only that root action. Record the candidate set, horizon, number of samples, terminal evaluator, random-number coupling, worst-case runtime, and fallback action. Without those declarations, "rollout" does not specify an executable controller.

4.8 Mobility Case: A Common Lane-Keeping Comparison

This section returns to the stationary stochastic lane-keeping MDP of Section 2.9. Reusing the same nine states, three steering actions, disturbance distribution, stage cost, and discount factor

Table 4.6: Offline and online roles of a value-space object.

Use	Offline requirement	Online requirement
Stored Q policy	Train $\widetilde{Q}$ and validate action ordering	Evaluate all finite-action outputs and take arg min
One-step value greedy	Train $\tilde{J}$	Evaluate model successors and minimize one-step cost plus $\tilde{J}$
Multistep lookahead	Train terminal $\tilde{J}$ and possibly a base policy	Build or sample a tree, optimize root action, meet a time budget
Parameterized actor	Train or distill $\widetilde{\mu}$	One forward pass, plus declared safety or constraint layer

is deliberate. Chapter 2 used the model to compare exact algorithms. Here the physical problem is held fixed while the representation and target mechanism change.

The normalized variables are

$$e \in \{-4, \dots, 4\}, \qquad u \in \{-1, 0, 1\}, \qquad \Pr(w = -1, 0, 1) = (0.10, 0.60, 0.30), \tag{4.61}$$

with

$$e_{k+1} = \mathrm{clip}(e_k + u_k + w_k, -4, 4), \qquad g(e, u) = e^2 + 0.20u^2, \qquad \alpha = 0.95. \tag{4.62}$$

The exact transition matrix is retained by the author as an audit model. It is visible to exact and fitted VI, but it is not supplied to the MC, TD, or Q-learning updates.

4.8.1 Six computations, one declared model and data source

The experiment performs the following computations.

1. **Exact VI reference.** Synchronous model-based VI computes all nine values and all 27 state–action comparisons.
2. **Exact PI reference.** Starting from the zero-steering base policy, exact linear-system evaluation and exact model-based greedy improvement are alternated until the policy stops changing. This exposes the evaluation–improvement loop that exact VI hides inside every backup.
3. **Fitted VI.** The value is compressed to four coefficients,

$$\tilde{J}(e; r) = r_0 + r_1 z + r_2 z^2 + r_3 z^3, \qquad z = e/4, \tag{4.63}$$

   and each exact Bellman target is projected by ridge-regularized least squares. This is representation approximation with a known-model target.
4. **MC and TD(0) policy evaluation.** Both methods first evaluate the same myopic zero-steering policy from sampled transitions. Thirty seeds, uniformly randomized episode

starts, 250 episodes, and 100 transitions per episode are used. MC processes one long return from each randomized episode start; TD(0) updates after every transition. These curves compare critics, not controllers.

5. **One-step and repeated approximate PI.** Each final MC or TD critic is passed through one exact-model greedy improvement. This deliberately mixed design has model-free evaluation and model-based improvement. The resulting candidate policies are evaluated independently by the audit model over all initial states. A second experiment repeats this loop for four improvement rounds, using 500 episodes per evaluation and 20 independent seeds. It separates an evaluator's data error from the number of policy-improvement cycles.
6. **Tabular Q-learning.** Thirty seeds each use 30,000 sampled transitions, decaying ϵ-greedy exploration, occasional state resets for coverage, and pair-specific diminishing stepsizes. There is no function approximation here: the approximation is asynchronous and sample based.

No DQN is trained on this nine-state problem. A neural Q-network would add optimization and hyperparameter effects where a 27-entry table is already an exact representation. The small experiment is used to verify the Bellman and sampling logic; DQN becomes meaningful when the state observation is too large or structured for a table.

This design makes two orthogonal axes visible. Fitted VI approximates the representation but uses complete model backups. Tabular Q-learning has an exact table representation for the finite MDP but approximates the Bellman expectation through samples. The MC/TD branch adds a third distinction: policy evaluation can be model free even when the subsequent J-based improvement is model based.

The four panels of Figure 4.7 answer different questions. The upper-left panel checks whether two sampled evaluators recover the value of the *same fixed base policy*; it is not an optimal-control comparison. The upper-right plots that evaluation RMSE against sampled transitions, with one-standard-deviation bands over seeds. The lower-left measures Q-learning's numerical error over all state–action entries. The lower-right measures what matters after taking an arg min: action agreement and the true closed-loop cost gap. The last two quantities can reach their useful values before every Q entry is highly accurate.

Likewise, Figure 4.5 separates representation from control. Its upper-left panel compares exact tabular values with a four-coefficient polynomial; the upper-right compares the induced actions. The lower-left shows that iterate change and Bellman projection residual are not the same diagnostic. The lower-right evaluates frozen policies under one common audit model. Reading only one panel would support a narrower claim than the experiment actually tests.

4.8.2 What the numerical result establishes

Table 4.7 reports true discounted policy costs from the audit model. Fitted VI has a value RMSE of 19.94 relative to J^*, yet its greedy policy agrees with the exact policy at all nine states. This is a small case-specific example of the action-ordering distinction developed in Chapter 3; it is not evidence that a large value error is generally harmless.

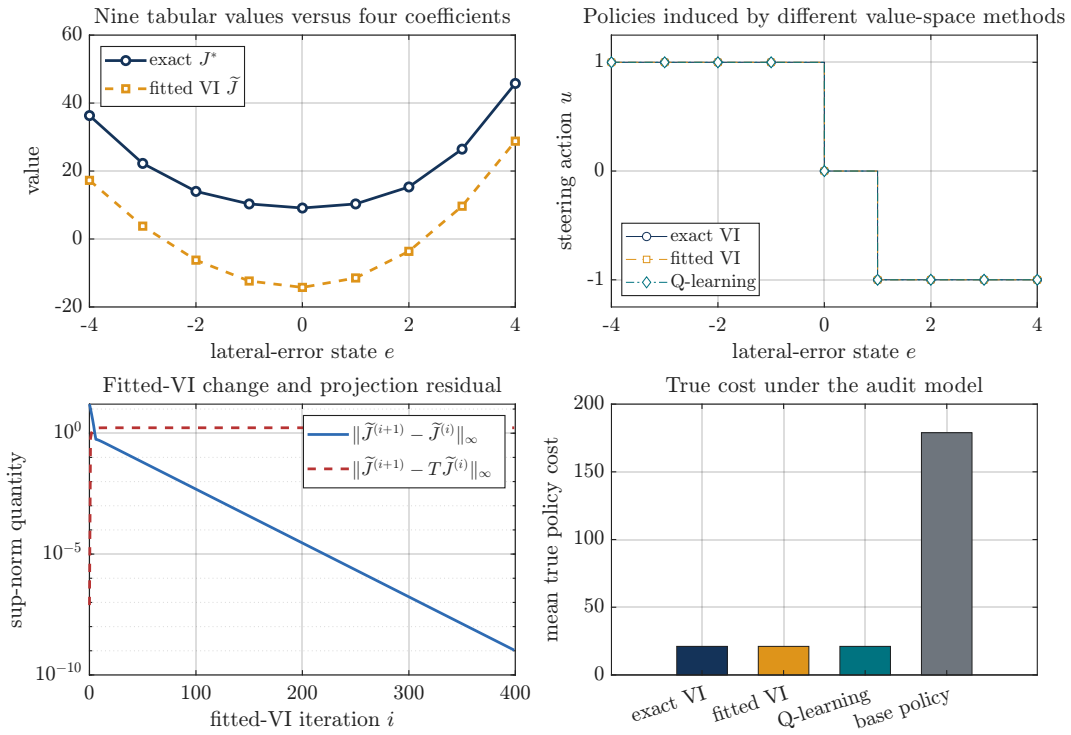


Figure 4.5: Original value-space comparison on the lane-keeping MDP. Fitted VI represents nine state values with four coefficients. Its iterate change can vanish while a projection residual remains. Policies are evaluated by the same exact audit model after training; that audit does not retroactively make the sample-based Q-learning update model based.

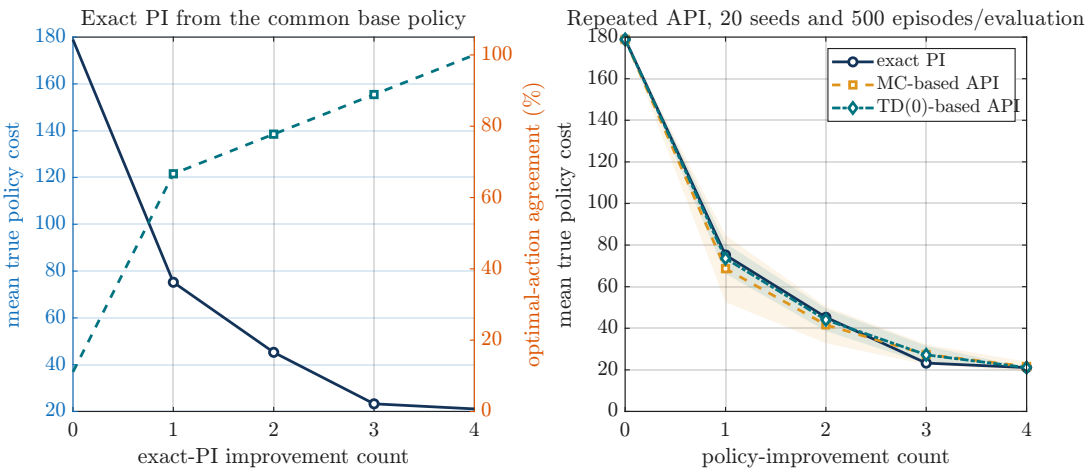


Figure 4.6: Original policy-iteration comparison from the common zero-steering base policy. The left panel reports exact PI after each accepted improvement. The right panel repeats sample-based evaluation and model-based greedy improvement for the same four rounds. Curves are means over 20 seeds and shaded bands are one standard deviation. A single improvement should be compared with the first exact-PI point, not with converged PI.

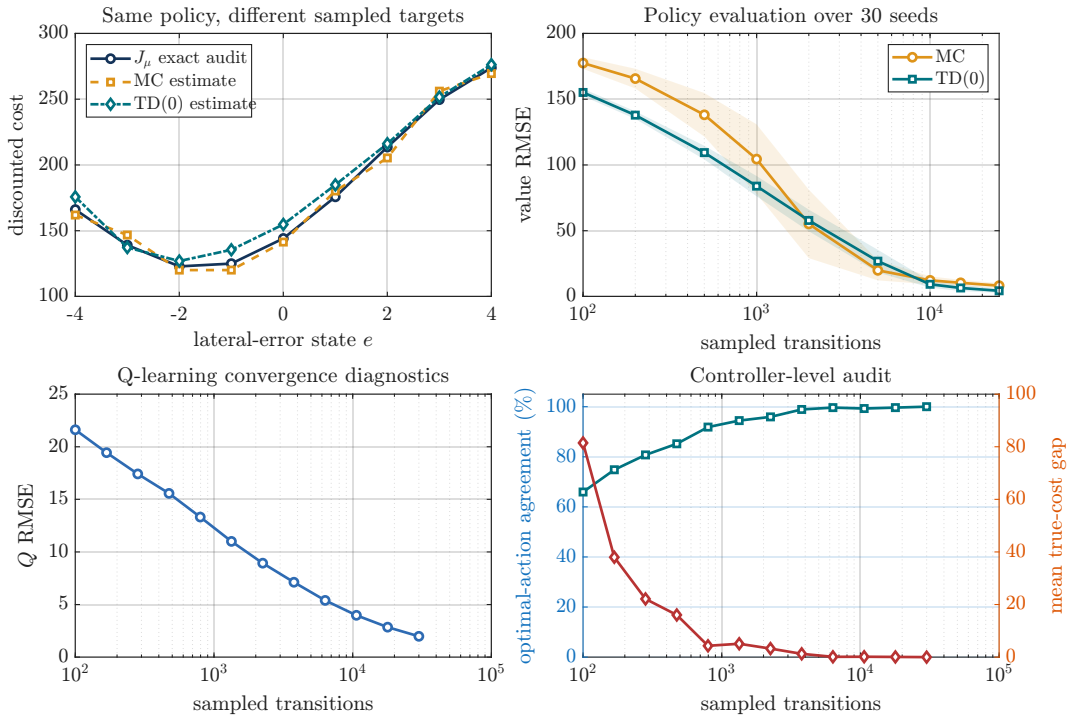


Figure 4.7: Original sample-based evaluation and Q-learning results. MC and TD(0) evaluate the same fixed myopic policy, whereas Q-learning applies a sampled optimality target and changes the greedy policy. Curves report means over 30 seeds; shaded evaluation bands show one standard deviation.

Table 4.7: Verified lane-keeping value-space results. Policy cost is averaged uniformly over the nine initial states.

Method	Stored DOF	Action agreement	Mean policy cost
Exact VI	9 values	100%	21.097
Exact PI, one improvement	9 policy values	66.7%	75.225
Exact PI, four improvements	9 policy values	100%	21.097
Fitted VI	4 coefficients	100%	21.097
MC-critic API, 30-seed mean	9 policy values	68.5%	68.971
TD-critic API, 30-seed mean	9 policy values	66.7%	75.225
MC-based API, four-round mean	9 policy values	98.9%	21.612
TD-based API, four-round mean	9 policy values	100%	21.097
Q-learning, representative seed	27 Q entries	100%	21.097
Myopic base policy	9 actions	11.1%	178.884

Exact PI reaches the same policy as exact VI after four improvements from the base policy. Its *first* improvement has cost 75.225. The one-step TD-based candidate has exactly the same cost and action agreement in this experiment: the apparently high number is therefore not primarily a TD failure. It is the expected intermediate point after improving a deliberately poor base policy only once. The lower one-step MC mean does not imply a more accurate evaluator; sampling error causes some seeds to jump to a different policy that happens to have lower audited cost.

When evaluation and improvement are repeated four times, TD-based API reaches the exact policy in all 20 runs. MC-based API reaches 98.9% mean action agreement and mean cost 21.612, close to the optimum 21.097. Thus more episodes can reduce critic variance, but tuning the evaluator alone cannot make one exact PI step equal converged PI. The structurally relevant changes are to repeat policy improvement or to perform deeper online lookahead, as developed in Sections 4.4 and 4.7. Approximation is not intrinsically responsible for the gap shown after one cycle.

Across the 30 Q-learning runs, final action agreement is 100% and the mean true-cost gap is numerically zero at the reported precision. Q RMSE remains nonzero because all 27 numerical entries converge more slowly than the action ordering required by the policy. For evaluation of the myopic policy, the final mean RMSEs are approximately 8.19 for MC and 4.30 for TD(0) with the declared finite data budget. The policies after one model-based improvement have mean costs 68.97 and 75.22, respectively. In this run the MC critic has worse value RMSE but produces the better one-step candidate. This is not a contradiction: policy improvement depends on action ordering through successor values, not on uniform numerical accuracy alone. Both candidates substantially improve on the base cost 178.88, while neither should be read as the final performance of a repeated approximate-policy-iteration method.

This finite experiment does not establish that MC or TD is uniformly better: the ranking depends on trajectory length, stepsize, initialization, mixing, terminal truncation, and the improvement rule. It does establish a more modest point that is central to the chapter: critic evaluation, policy improvement, and final controller audit are separate computations and should be reported separately.

Implementation note

The base-MATLAB script `generate_figures.m` is stored in the Chapter 4 code directory. It records seeds and writes the plotted statistics to CSV. The audit model is used to calculate reference values and true policy costs after training. Students can remove the audit calls and still run MC, TD(0), and Q-learning from the transition sampler, but they will then lose exact numerical verification.

4.9 A Linear-Programming View of Value Approximation

Bellman iteration is not the only way to characterize J^*. For a finite discounted cost problem, the inequalities

$$J(x) \le \bar{g}(x,u) + \alpha \sum_{x'} p(x' \mid x,u) J(x'), \qquad \text{for every } (x,u), \tag{4.64}$$

imply $J \le TJ$. Monotonicity and repeated application of T then imply $J \le J^*$. Consequently, for any strictly positive weights $c(x)$, the exact value is the solution of

$$\max_{J} \quad \sum_{x} c(x)J(x) \tag{4.65}$$

$$\text{subject to} \quad J(x) \le \bar{g}(x,u) + \alpha \sum_{x'} p(x' \mid x,u)J(x') \quad \forall (x,u). \tag{4.66}$$

The maximization selects the largest Bellman-inequality lower bound.

For a linear architecture $\tilde{J}(x;r) = \phi(x)^\top r$, approximate linear programming reduces the number of decision variables but not automatically the number of constraints:

$$\max_{r} \quad \sum_{x \in \widetilde{\mathscr{X}}} c(x)\phi(x)^\top r, \tag{4.67}$$

$$\text{subject to} \quad \phi(x)^\top r \le \bar{g}(x,u) + \alpha \sum_{x'} p(x' \mid x,u)\phi(x')^\top r \quad (x,u) \in \widetilde{\mathscr{C}}. \tag{4.68}$$

Representative state–control constraints $\widetilde{\mathscr{C}}$ may be sampled and then enriched where violations are found. The method makes global Bellman inequalities explicit, but constraint coverage becomes its analogue of data coverage. If important constraints are omitted, feasibility on the sampled set is not a global lower-bound certificate. This subsection retains the LP connection from the prior course notes while keeping fitted and sample-based Bellman methods as the chapter's main path.

The weights $c(x) > 0$ do not change the exact finite-model optimizer, but after restricting J to an approximation architecture they specify where a large lower bound is preferred. They therefore play a role analogous to a state-relevance distribution. A practical constraint-generation loop starts with a manageable set $\widetilde{\mathscr{C}}$, solves the approximate LP, searches a larger audit set for the most violated Bellman inequality, and adds that state–action pair before resolving. Termination with no sampled violations is an empirical audit, not a global certificate unless the search itself is exhaustive or has a separate bound.

Approximate LP is included because it exposes a qualitatively different control approximation: fitted VI minimizes a regression discrepancy, whereas ALP preserves selected one-sided Bellman inequalities. That structure can be valuable when lower bounds or dual occupancy interpretations matter. It is not developed into a primary learning algorithm here because very large or continuous spaces move the difficulty from value variables to constraint generation. The detailed course path therefore returns to sampled critics, API, and online improvement after establishing this connection.

4.10 Value Space and Policy Space Are Complementary Axes

Chapter 2 introduced approximation in value space and policy space as two different placements of limited computation. They should not be treated as competing names for the same method. A controller may approximate either object, both, or neither.

DQN primarily occupies the value-only row: its finite-action Q-network defines an implicit policy. PPO primarily improves a parameterized stochastic policy while also using a value critic

Table 4.8: Where approximation enters and what remains at execution.

Design	Approximate objects	Execution mechanism
Value only	$\tilde{J}$ or finite-action $\widetilde{Q}$	Model-based minimization for J, direct finite arg min for Q
Policy only	$\widetilde{\mu}(x)$	Direct actor output; no critic is required at execution
Actor–critic	$\widetilde{\mu}$ and $\widetilde{Q}_\mu$ or $\tilde{J}_\mu$	Actor executes; critic trains or evaluates the actor
Critic plus lookahead	terminal $\tilde{J}$ or $\widetilde{Q}$ and optionally a base actor	Online model-based or sampled planning improves the root action
Exact online control	No learned value or actor	Analytic minimization or numerical MPC from a declared model

during training. TD3 and SAC approximate both a continuous actor and Q critics. From this book's viewpoint, all can be located by answering three questions:

1. Which policy is being evaluated by the critic target?
2. How is the new policy improved and represented?
3. Is another improvement performed online through a model, lookahead, or safety layer?

Chapter 6 develops the second question. Chapters 5 and 8 develop the third. The first question is answered in Sections 4.3 and 4.4: MC and TD evaluate a declared policy, API changes that policy between evaluation phases, and Q-learning bypasses fixed-policy evaluation by targeting Q^*.

Appendix C provides self-contained algorithm cards for DQN, PPO, TD3, and SAC, including their actor, critic target, data relation, and cost/reward translation. Chapter 1 will later retain only a short orientation to these names. This division preserves Chapter 1's main purpose—motivating learning-based control—while giving a reader without prior DRL coursework a nearby technical reference. It also preserves the DP structure while connecting it to the algorithm names students encounter in a standard deep-RL course.

4.11 Conditions, Failure Modes, and Deployment Checks

Value-space approximation can fail even when the code runs and the training loss decreases. The checks below identify which claim is at risk.

Conditions to check

1. **Object:** Is the target J_μ, Q_μ, J^*, or Q^*? A behavior-policy return is not an optimality target.

2. **Markov information:** Does the critic input contain the state information needed for the declared Bellman equation? A frame, short history, or RNN hidden state must be justified rather than assumed sufficient.
3. **Coverage:** Which states and actions are visited? Greedy data collection can leave alternative actions unevaluated exactly where policy improvement needs them.
4. **Backup:** Is the expectation computed from a model, one sampled successor, or a replay distribution? State what "model free" means for that computation.
5. **Bootstrapping:** How are target parameters delayed or separated from online parameters? Check for moving-target instability.
6. **Off-policy correction:** Does the target correspond to the behavior policy, a target policy, or an optimality operator? Reuse is not automatically unbiased.
7. **Decision quality:** Report action agreement, decision regret, true or simulated closed-loop cost, and constraint violations in addition to value RMSE or TD loss.
8. **Continuous action:** Identify whether action discretization, numerical minimization, or a separate actor produces the actual command.
9. **Online budget:** For rollout or lookahead, report depth, branching, sample count, worst-case computation, and fallback action.
10. **Safety claim:** A Bellman loss, replay buffer, or target network supplies no constraint guarantee by itself.

Control viewpoint

The mobility engineer should maintain two evaluation paths. The learning path generates critic targets from the available data. The audit path evaluates the frozen induced policy under independent trajectories, shifted conditions, and constraint checks. When a trusted model exists, it may be used in the audit even if the learned backup was model free. That is verification, not a change to the training algorithm's classification.

4.12 Chapter Summary

- Chapter 3's approximators become value-space methods only after a DP or RL target-generation mechanism and a policy-improvement mechanism are attached.
- Fitted VI alternates a Bellman optimality backup and a projection. Contraction of T alone does not guarantee contraction of an arbitrary projected or trained update.
- MC estimates policy returns from long samples; TD bootstraps from a current value; multistep and λ returns interpolate between these endpoints.
- Q-learning is sampled asynchronous Q-value iteration. SARSA performs on-policy Q evaluation and improvement. Their targets differ even when they use the same table or network.

- DQN retains the finite-action Q-Bellman target and adds replay, a target network, and neural approximation. Double and dueling variants alter target evaluation and output structure, respectively.
- An approximate terminal value can be used by a stored greedy policy or by online lookahead. Exact lookahead attenuates terminal-value error under strong assumptions and connects value learning to MPC.
- Value-space and policy-space approximation are complementary. Modern actor–critic algorithms approximate both and may still be combined with online model-based improvement.

4.13 Exercises

Exercise 4.1 (Fitted VI error propagation). Starting from Equation (4.9), prove Equation (4.10) by induction. Evaluate the steady upper bound when $\alpha = 0.95$ and $\bar{\varepsilon} = 0.1$. Explain why the factor $1/(1-\alpha)$ makes small per-iteration errors important in long-horizon problems.

Exercise 4.2 (Projection fixed point versus Bellman fixed point). Construct a two-dimensional value vector and a one-dimensional approximation subspace for which $\tilde{J} = \Pi T\tilde{J}$ but $\tilde{J} \neq T\tilde{J}$. Identify the iteration-change residual and the Bellman residual.

Exercise 4.3 (MC and TD targets). For a three-transition trajectory with costs $2, 1, 4$, discount 0.9, and terminal estimate $\hat{J}_T(x_3) = 5$, calculate the three-step return from x_0 and the TD(0) target when the current estimate at x_1 is 6. State which target depends on the current approximator.

Exercise 4.4 (Linear TD and LSTD). Using two-dimensional features, collect five transitions from a declared fixed policy. Form the empirical matrix and vector in Equation (4.27). Compare the LSTD solution with 100 incremental TD(0) passes under two stepsizes. Discuss conditioning and ordering effects.

Exercise 4.5 (Q-learning versus SARSA). Write one numerical update of Equation (4.40) and one of Equation (4.43) for the same sampled transition. Choose a next exploratory action that is not greedy and show exactly where the targets differ.

Exercise 4.6 (Double target under cost minimization). Suppose the online next-state Q outputs are $(4, 3, 5)$ and the target-network outputs are $(2, 6, 7)$. Compute the DQN and Double-DQN continuation values. Explain why the direction of selection bias is reversed when moving from reward maximization to cost minimization.

Exercise 4.7 (Dueling identifiability). Show that $Q(x,u) = V(x) + A(x,u)$ is unchanged if a scalar is added to V and subtracted from every $A(x,u)$. Verify that the mean-centering term in Equation (4.50) removes this degree of freedom.

Exercise 4.8 (Reproduce the lane-keeping comparison). Run the Chapter 4 MATLAB script. Plot each of the 30 Q-learning policy-cost gaps rather than only their mean. Then remove random state resets and explain the change through state–control visitation counts. Do not use the audit transition matrix inside the learning update.

Exercise 4.9 (Terminal value and lookahead depth). For $\alpha = 0.95$ and $\varepsilon = 2$, calculate the bounds in Equation (4.55) for $\ell = 1, 2, 5, 10$. If computation grows as $|\mathcal{U}|^{\ell}$, explain why the bound alone cannot choose a real-time horizon.

Exercise 4.10 (Value versus policy approximation for slalom). Consider a slalom task with a finite preview window and continuous steering. Propose (a) a value-space design with online lookahead, (b) a policy-space actor, and (c) an actor–critic plus lookahead design. For each, state the Markov state, training target, action-generation mechanism, model use, and shifted validation conditions.

Exercise 4.11 (Approximate linear programming). Write the exact Bellman-inequality LP for the nine-state lane-keeping MDP. Then substitute the four-feature architecture in Equation (4.63) and retain only half of the state–action constraints. Check the omitted constraints afterward and explain why sampled feasibility is not a global certificate.

5 LQR and MPC Through the DP Lens

Learning objectives

After studying this chapter, the reader should be able to:

1. derive finite-horizon LQR by backward dynamic programming;
2. interpret the algebraic Riccati equation as Bellman's fixed-point equation restricted to quadratic value functions;
3. explain why an exact infinite-horizon terminal value makes finite lookahead exact;
4. distinguish an approximate terminal value from the true cost of the policy it induces;
5. explain the geometric and Newton-type interpretation of greedy policy improvement;
6. interpret receding-horizon MPC as online multistep lookahead;
7. state standard terminal-cost and terminal-set conditions for recursive feasibility and stability; and
8. connect the analysis to a linearized vehicle lateral-control problem.

Chapter roadmap

LQR supplies a setting in which every DP object can be written explicitly. The chapter begins with finite- and infinite-horizon Riccati equations, then uses terminal approximations to introduce lookahead. A geometric section compares VI, PI, and rollout before the same ideas are carried to a scalar laboratory, lateral regulation, and constrained MPC.

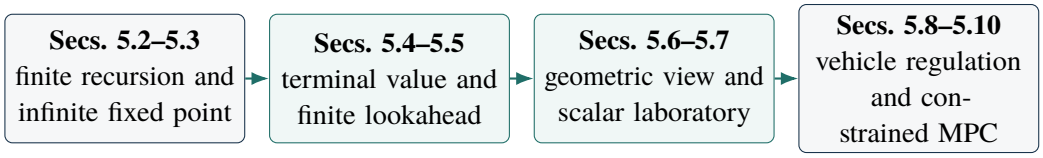


Figure 5.1: Original map of Chapter 5. The terminal-value/lookahead block is the bridge from offline value approximation to online MPC-style improvement.

5.1 Why Return to a Linear–Quadratic Problem?

The preceding chapters move from exact dynamic programming to value-function approximation. Returning to LQR at this point is not a retreat to an easier subject. It is a controlled experiment. Bellman operators act on functions and are therefore difficult to draw. In LQR, the relevant functions are quadratic, so the Bellman operation becomes a Riccati operation on matrices. In a scalar problem it becomes a curve on a plane. Approximate values, greedy policies, policy costs, value iteration, and lookahead can then be compared without hiding the mechanism inside a neural network or a nonlinear solver.

This solvable structure also exposes the central distinction of the book:

> An approximation may be computed offline, while the action actually applied to the system is obtained by a fresh online minimization.

The distinction is central to the value-space interpretation developed by Bertsekas (2019), the online-play viewpoint of Bertsekas (2022), and the unified discussion in Bertsekas (2025). Our derivations and numerical figures are original to this chapter; the cited sources provide the broader conceptual and technical context.

Control viewpoint

LQR is usually introduced as a closed-form optimal-control design. Here it also serves as a transparent DP model: the Riccati recursion is value iteration on a structured function class, and a finite-horizon controller is a lookahead policy.

5.2 Finite-Horizon LQR Is Backward Dynamic Programming

Consider the time-invariant discrete-time system

$$x_{k+1} = Ax_k + Bu_k, \qquad x_k \in \mathbb{R}^n, \quad u_k \in \mathbb{R}^m, \tag{5.1}$$

and the finite-horizon cost

$$J_0(x_0; u_0, \dots, u_{N-1}) = x_N^\mathsf{T} S x_N + \sum_{k=0}^{N-1} \left(x_k^\mathsf{T} Q x_k + u_k^\mathsf{T} R u_k \right). \tag{5.2}$$

Assumption 5.1 (Finite-horizon weights). The matrices satisfy $Q \succeq 0$, $S \succeq 0$, and $R \succ 0$.

Let $J_k^*(x)$ denote the optimal cost from stage k to N. Bellman's recursion is

$$J_k^*(x) = \min_u \left\{ x^\mathsf{T} Q x + u^\mathsf{T} R u + J_{k+1}^*(Ax + Bu) \right\}, \qquad J_N^*(x) = x^\mathsf{T} S x. \tag{5.3}$$

Suppose $J_{k+1}^*(x) = x^\mathsf{T} P_{k+1} x$. Completing the square in u gives

$$K(P) = \left(R + B^\mathsf{T} P B \right)^{-1} B^\mathsf{T} P A, \tag{5.4}$$

$$\mathscr{F}(P) = Q + A^\mathsf{T} P A - A^\mathsf{T} P B \left(R + B^\mathsf{T} P B \right)^{-1} B^\mathsf{T} P A. \tag{5.5}$$

The minimizing control and the backward recursion are therefore

$$u_k^* = -K(P_{k+1})x_k, \qquad P_k = \mathscr{F}(P_{k+1}), \qquad P_N = S. \tag{5.6}$$

Proposition 5.1 (Quadratic closure under the Bellman operation). *Under assumption 5.1, if the terminal value is quadratic, then every finite-horizon value function is quadratic. In particular, $J_k^*(x) = x^\mathsf{T} P_k x$ with P_k generated by Equation (5.6).*

Proof. The terminal statement holds with $P_N = S$. If the statement holds at $k+1$, the expression minimized in Equation (5.3) is a strictly convex quadratic function of u because $R + B^\mathsf{T} P_{k+1} B \succ 0$. Its unique minimizer is Equation (5.4), and substitution produces Equation (5.5). Backward induction completes the proof. □

This is exact DP, but its tractability comes from structure. For a generic nonlinear system, the Bellman operation does not preserve a finite-dimensional family of functions. LQR is special because the quadratic family is closed under the Bellman operation. Standard treatments include Anderson and Moore (2007) and Bertsekas (2019).

5.3 Infinite-Horizon LQR as a Fixed-Point Problem

Now consider the undiscounted cost

$$J(x_0; \mu) = \sum_{k=0}^{\infty} \left(x_k^\mathsf{T} Q x_k + u_k^\mathsf{T} R u_k \right), \qquad u_k = \mu(x_k). \tag{5.7}$$

Assumption 5.2 (Stabilizing infinite-horizon solution). The pair (A, B) is stabilizable, and the state penalty detects all unstable uncontrolled modes; a standard sufficient statement is that $(Q^{1/2}, A)$ is detectable. Also $Q \succeq 0$ and $R \succ 0$.

Under these conditions, the stabilizing solution $P_\infty \succeq 0$ of the discrete algebraic Riccati equation (DARE)

$$P_\infty = \mathscr{F}(P_\infty) \tag{5.8}$$

defines

$$J^*(x) = x^\mathsf{T} P_\infty x, \qquad u = -K_\infty x, \qquad K_\infty = K(P_\infty). \tag{5.9}$$

The closed-loop matrix $A - BK_\infty$ is stable.

Equation (5.8) is Bellman's equation restricted to quadratic value functions. Consequently, the matrix iteration

$$P^{(i+1)} = \mathscr{F}\left(P^{(i)}\right) \tag{5.10}$$

is value iteration within that family. The algorithmic index i is deliberately different from the physical time k.

Policy iteration is visible as a different pair of matrix operations. Given a stabilizing feedback $u = -K^{(i)}x$, policy evaluation solves the Lyapunov equation

$$P^{(i)} = Q + K^{(i)\top} R K^{(i)} + (A - BK^{(i)})^\top P^{(i)} (A - BK^{(i)}), \tag{5.11}$$

and policy improvement sets

$$K^{(i+1)} = K(P^{(i)}). \tag{5.12}$$

Terminology note

The matrix Q in LQR is a state-cost weight. A Q-factor in RL is a function of state and action. The shared letter is historical and context dependent; the two objects should never be identified.

5.4 Terminal Values and Finite Lookahead

The terminal matrix S in a finite-horizon problem is not merely a numerical boundary condition. It represents everything that the planner believes about the cost after its explicit horizon. This interpretation yields an exact and important result.

Indeed, repeated application of Equation (5.6) gives

$$P_k = \mathscr{F}^{N-k}(S), \qquad u_k^* = -K\big(\mathscr{F}^{N-k-1}(S)\big)\, x_k, \quad k = 0, \dots, N-1. \tag{5.13}$$

This equation is the explicit link between finite-horizon LQR and the fixed-point statement that follows.

Proposition 5.2 (Exact terminal value reproduces the infinite-horizon policy)**.** *Under assumption 5.2, choose $S = P_\infty$ in the finite-horizon problem (5.2). For every horizon $N \geq 1$,*

$$P_k = P_\infty, \qquad K(P_{k+1}) = K_\infty, \qquad k = 0, \dots, N-1. \tag{5.14}$$

Thus the first finite-horizon action is the infinite-horizon optimal action, regardless of the horizon length.

Proof. Because $P_\infty = \mathscr{F}(P_\infty)$, the backward Riccati recursion remains at the fixed point when initialized with $P_N = P_\infty$. Equation (5.4) then gives the same feedback gain at every stage. □

The result is the LQR specialization of Bellman's fixed-point principle; see the rollout and terminal-value discussions in Bertsekas (2022) and Bertsekas (2025).

The same principle is not specific to LQR. If the exact infinite-horizon value J^* is used at the leaf of an ℓ-step lookahead calculation, then Bellman's fixed-point relation gives

$$T^\ell J^* = J^*. \tag{5.15}$$

The first action of the lookahead solution is optimal. The practical problem is that J^* is precisely the object we rarely know.

5.5 Approximate Terminal Values and Their Induced Policies

Let

$$\tilde{J}(x) = x^{\mathsf{T}}\widetilde{P}x \tag{5.16}$$

be an approximation of J^*. One-step lookahead minimizes

$$x^{\mathsf{T}}Qx + u^{\mathsf{T}}Ru + (Ax + Bu)^{\mathsf{T}}\widetilde{P}(Ax + Bu), \tag{5.17}$$

and yields the feedback

$$\widetilde{\mu}(x) = -K(\widetilde{P})x. \tag{5.18}$$

Two matrices must now be kept separate:

- $\widetilde{P}$ describes the terminal value used to choose the action;
- $P_{K(\widetilde{P})}$ describes the true infinite-horizon cost of the resulting policy, when that policy is stabilizing.

The second matrix solves

$$P_{\widetilde{K}} = Q + \widetilde{K}^{\mathsf{T}}R\widetilde{K} + (A - B\widetilde{K})^{\mathsf{T}}P_{\widetilde{K}}(A - B\widetilde{K}), \qquad \widetilde{K} = K(\widetilde{P}). \tag{5.19}$$

Confusing $\widetilde{P}$ with $P_{\widetilde{K}}$ is a common conceptual error. The former is an evaluation heuristic; the latter is achieved closed-loop performance.

Proposition 5.3 (Local quadratic sensitivity of LQR policy cost). *Suppose assumption 5.2 holds and $\widetilde{P}$ is sufficiently close to P_∞ that $A - BK(\widetilde{P})$ remains stable. Then, locally,*

$$\left\|K(\widetilde{P}) - K_\infty\right\| = O\left(\left\|\widetilde{P} - P_\infty\right\|\right), \tag{5.20}$$

$$\left\|P_{K(\widetilde{P})} - P_\infty\right\| = O\left(\left\|\widetilde{P} - P_\infty\right\|^2\right). \tag{5.21}$$

Proof. The gain map in Equation (5.4) is smooth near P_∞ because $R + B^{\mathsf{T}}P_\infty B \succ 0$, giving Equation (5.20) in any finite-dimensional matrix norm. Let $A_K = A - BK$ and $H_\infty = R + B^{\mathsf{T}}P_\infty B$. For every stabilizing K, subtracting the DARE from the policy-evaluation equation and using the optimality condition for K_∞ gives the exact identity

$$P_K - P_\infty = A_K^{\mathsf{T}}(P_K - P_\infty)A_K + (K - K_\infty)^{\mathsf{T}}H_\infty(K - K_\infty). \tag{5.22}$$

In a sufficiently small stabilizing neighborhood of K_∞, the inverse of the discrete Lyapunov operator $X \mapsto X - A_K^{\mathsf{T}}XA_K$ is uniformly bounded. Hence $\|P_K - P_\infty\| = O(\|K - K_\infty\|^2)$. Combining this bound with Equation (5.20) proves Equation (5.21). □

This local result is a matrix form of the LQR performance-difference argument used in modern policy-optimization analyses; see, for example, Fazel et al. (2018).

For ℓ-step lookahead, backward DP first transforms the terminal matrix to

$$P_{\text{leaf}} = \mathscr{F}^{\ell-1}(\widetilde{P}), \tag{5.23}$$

and the applied gain is $K(P_{\text{leaf}})$. Thus additional lookahead can move the terminal approximation toward the Riccati fixed point before the first action is chosen. It does not magically erase all errors: convergence depends on the Riccati iteration, and constraints or model mismatch change the analysis.

5.6 Geometric View of Policy Improvement and Lookahead

The algebra above separates an approximate terminal value from the cost of the policy it induces. Operator geometry shows why the two objects differ and why online minimization can improve an offline approximation. We first use the discounted finite-state model of Chapter 2, where the geometry is especially transparent, and then realize the same construction exactly with the scalar Riccati map.

5.6.1 The Bellman envelope and a touching policy operator

For a stationary policy μ, write its Bellman operator as

$$T_\mu J = g_\mu + \alpha P_\mu J, \qquad 0 < \alpha < 1. \tag{5.24}$$

It is affine in the value vector. The optimality operator is the pointwise lower envelope

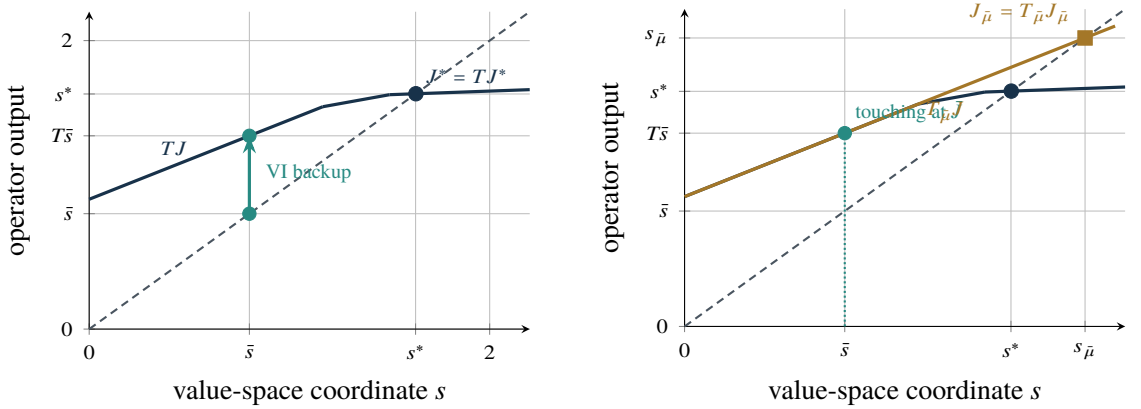
$$TJ = \min_\mu T_\mu J. \tag{5.25}$$

Given an evaluator $\bar{J}$, a greedy policy $\bar{\mu}$ selects an active member of this envelope:

$$T_{\bar{\mu}} \bar{J} = T\bar{J}. \tag{5.26}$$

The equality is local to the value used for improvement. It does not say that $T_{\bar{\mu}} J = TJ$ for other values, nor that $\bar{J}$ is the cost of $\bar{\mu}$.

Figure 5.2 draws a one-dimensional slice through value space. For an n-state model, J is a vector in $\mathbb{R}^n$, so the curve is a schematic slice rather than a literal plot of the full operator. The slice is useful because it preserves the three relations that matter: the Bellman envelope, the touching policy operator, and their distinct fixed points.



(a) Value iteration: one Bellman backup

(b) Greedify, then evaluate the selected policy

Figure 5.2: Original schematic of Bellman-operator geometry on a one-dimensional slice of value space. VI stops after the vertical move $\bar{J} \mapsto T\bar{J}$. Greedy improvement selects a policy operator touching the lower envelope at $\bar{J}$; exact policy evaluation follows that affine operator to its own fixed point $J_{\bar{\mu}}$. The latter need not equal either $\bar{J}$ or J^*.

5.6.2 VI, PI, and rollout are different moves

The geometric distinction can be written without a picture:

$$\text{VI:} \qquad J^+ = TJ, \tag{5.27}$$

$$\text{greedy selection:} \qquad T_{\bar{\mu}}\bar{J} = T\bar{J}, \tag{5.28}$$

$$\text{exact evaluation:} \qquad J_{\bar{\mu}} = T_{\bar{\mu}} J_{\bar{\mu}}. \tag{5.29}$$

VI performs the first line repeatedly. PI starts from the exactly evaluated cost of its current policy, applies the second line, and then solves the third. Approximate PI may replace either the evaluator or the policy-evaluation solve.

This also explains the Newton terminology. To solve $F(J) = J - TJ = 0$, choose the active affine operator $T_{\bar{\mu}}$ at $\bar{J}$ and solve the local fixed-point model

$$J - T_{\bar{\mu}} J = 0. \tag{5.30}$$

The solution is $J_{\bar{\mu}}$. For a differentiable scalar envelope this is an ordinary tangent-based Newton construction. For a piecewise-affine or nonsmooth Bellman operator, *Newton-type* is the more accurate term: the greedy policy supplies an active supporting operator rather than a unique classical derivative. This interpretation follows the operator development in Bertsekas (2022).

Rollout is one policy-improvement step whose evaluator is the exactly evaluated cost J_μ of a base policy. Thus

$$T_{\bar{\mu}} J_\mu = TJ_\mu \le T_\mu J_\mu = J_\mu. \tag{5.31}$$

The inequality supplies a genuine improvement guarantee in the discounted finite-state setting: monotonicity of $T_{\bar{\mu}}$ implies $J_{\bar{\mu}} \le J_\mu$. Starting from a generic learned $\tilde{J}$ is different; without an error bound or a suitable stability condition, the same ordering need not hold.

5.6.3 Multistep lookahead moves the Newton starting point

An ℓ-step calculation with terminal evaluator $\tilde{J}$ first forms the effective evaluator

$$\bar{J} = T^{\ell-1}\tilde{J} \tag{5.32}$$

and then selects a policy μ_ℓ satisfying

$$T_{\mu_\ell}\bar{J} = T\bar{J} = T^{\ell}\tilde{J}. \tag{5.33}$$

The repeated backups and the final greedy minimization have different jobs. The backups improve the point at which the policy operator is selected; only the last minimization determines the first action applied to the plant.

The rollout guarantee extends to this construction when the terminal evaluator is the exact cost of a base policy.

Proposition 5.4 (Multistep rollout improvement). *For a discounted finite-state model, let J_μ be the exact cost of a base policy. Set $\bar{J} = T^{\ell-1}J_\mu$ and choose μ_ℓ by Equation (5.33). Then*

$$J_{\mu_\ell} \le \bar{J} \le J_\mu \tag{5.34}$$

componentwise.

Proof. Because $TJ_\mu \le T_\mu J_\mu = J_\mu$ and T is monotone, its successive iterates from J_μ are nonincreasing. Hence $\bar{J} \le J_\mu$ and $T_{\mu_\ell}\bar{J} = T\bar{J} \le \bar{J}$. Repeated application of the monotone contraction T_{μ_ℓ} decreases monotonically from $\bar{J}$ to its fixed point J_{μ_ℓ}, proving the first inequality. □

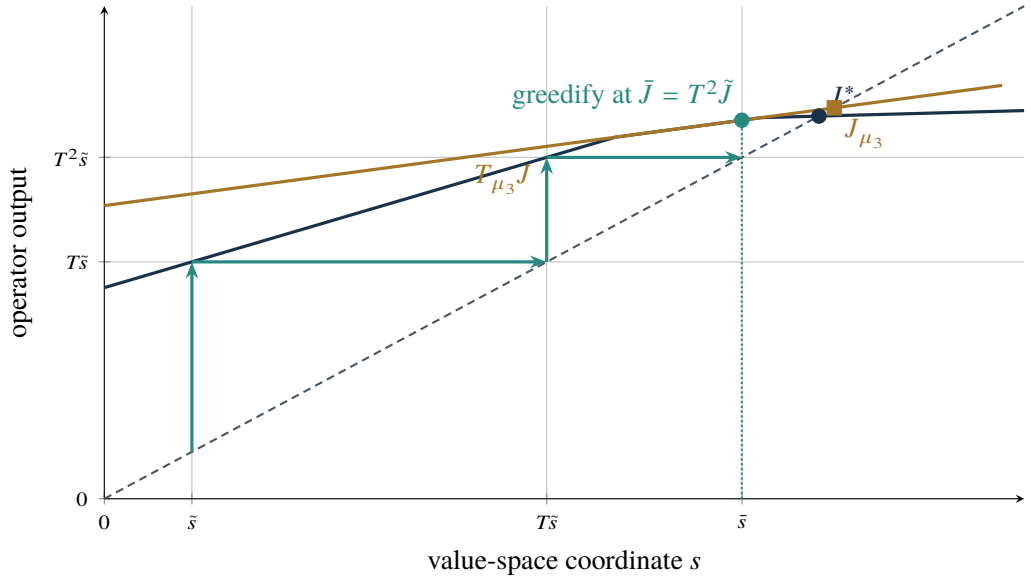


Figure 5.3: Original schematic of three-step lookahead on the same one-dimensional slice. Two Bellman backups move the terminal approximation $\tilde{J}$ to the effective evaluator $\bar{J} = T^2\tilde{J}$; the final minimization chooses the touching policy operator. Exact evaluation of that policy reaches J_{μ_3}, which is distinct from the leaf evaluator and need not equal J^*.

Control viewpoint

An MPC controller with horizon ℓ and terminal value V_f applies the first action of $T^\ell V_f$ and repeats the calculation after the next measurement. Rollout uses $V_f = J_\mu$ for a base policy. A learned critic may instead supply $V_f \approx J^*$, while an actor supplies a base policy or a warm start for the online minimization. AlphaZero-style online play uses the analogous learned-value plus online-search pattern. These methods share an improvement architecture, not identical assumptions or guarantees.

For discounted tables, contraction makes every policy fixed point well defined. For undiscounted LQR, the corresponding policy must be stabilizing; otherwise its infinite-horizon cost is not a finite quadratic form. For constrained MPC, recursive feasibility and stability require terminal ingredients or other verifiable conditions. The geometry explains the update mechanism but does not replace these checks.

5.7 Scalar LQR: Riccati Iteration and Lookahead

Consider

$$x_{k+1} = ax_k + bu_k, \qquad J = \sum_{k=0}^{\infty} (qx_k^2 + ru_k^2), \tag{5.35}$$

with $b \neq 0$, $q > 0$, and $r > 0$. For $J(x) = px^2$, the Riccati map is

$$\mathscr{F}(p) = q + \frac{a^2 rp}{r + b^2 p}, \tag{5.36}$$

and the positive-feedback magnitude in $u = -k(p)x$ is

$$k(p) = \frac{abp}{r + b^2 p}. \tag{5.37}$$

If $|a - bk| < 1$, the actual cost coefficient of a fixed gain k is

$$p_k = \frac{q + rk^2}{1 - (a - bk)^2}. \tag{5.38}$$

The DARE reduces to a quadratic equation,

$$b^2 p^2 + \left(r - qb^2 - a^2 r\right) p - qr = 0, \tag{5.39}$$

whose positive solution is p_∞.

5.7.1 Riccati geometry of value and policy iteration

The generic geometry of Section 5.6 is exact, not schematic, on the scalar quadratic family. For a fixed feedback $u = -kx$, the policy-specific Bellman operation maps the coefficient p to

$$\mathscr{F}_k(p) = q + rk^2 + (a - bk)^2 p. \tag{5.40}$$

This is an affine function of p. Minimizing over the feedback gain recovers the Riccati map

$$\mathscr{F}(p) = \min_k \mathscr{F}_k(p), \tag{5.41}$$

so $\mathscr{F}$ is the lower envelope of the policy lines. At a candidate coefficient $\bar{p}$, the greedy gain $\bar{k} = k(\bar{p})$ satisfies

$$\mathscr{F}_{\bar{k}}(\bar{p}) = \mathscr{F}(\bar{p}). \tag{5.42}$$

The line $\mathscr{F}_{\bar{k}}$ touches the Riccati curve at the coefficient used for improvement. Its intersection with the 45-degree line solves $p = \mathscr{F}_{\bar{k}}(p)$ and is the true infinite-horizon cost coefficient $p_{\bar{k}}$ when $\bar{k}$ is stabilizing. The intersection of $p = \mathscr{F}(p)$ is the optimal coefficient p_∞.

Thus scalar LQR makes the Newton-type interpretation algebraic: choose the supporting policy line at the current coefficient and solve its linear fixed-point equation. Value iteration instead takes the single update $p^+ = \mathscr{F}(p)$. Policy iteration alternates a greedy gain calculation with the exact policy-cost calculation in Equation (5.38).

Multistep lookahead changes the coefficient at which the supporting line is chosen. With terminal coefficient $\tilde{p}$, it first forms

$$\bar{p} = \mathscr{F}^{\ell-1}(\tilde{p}) \tag{5.43}$$

and then applies $k(\bar{p})$. The generic sequence $\tilde{J} \mapsto T^{\ell-1}\tilde{J} \mapsto \mu_\ell$ therefore becomes an explicit Riccati iteration followed by one feedback-gain calculation.

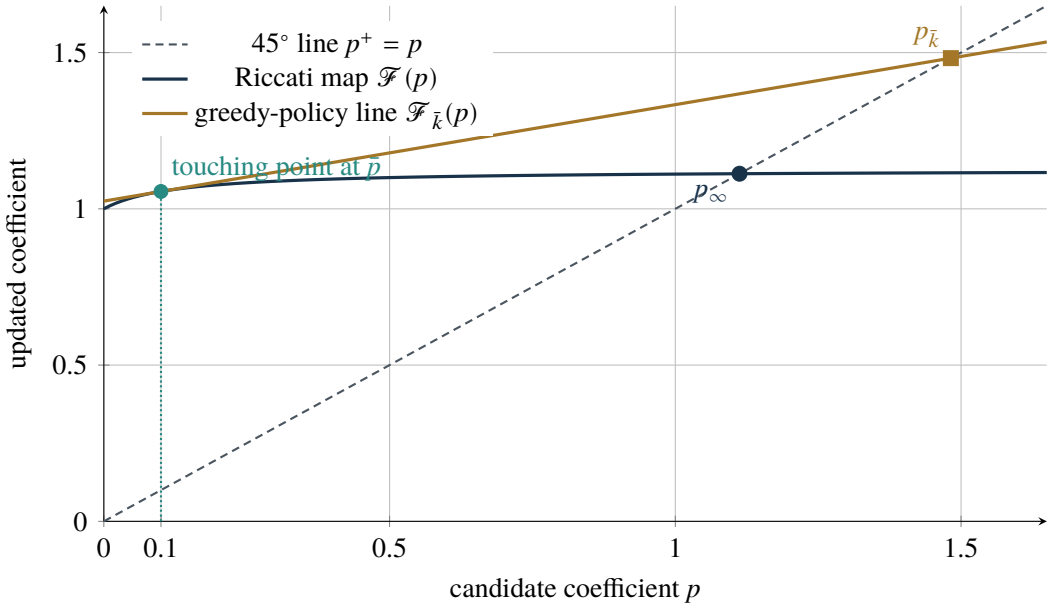


Figure 5.4: Original scalar realization of the operator geometry for $a = 1$, $b = 2$, $q = 1$, and $r = 0.5$, illustrated at $\bar{p} = 0.1$. Greedy improvement selects the policy line touching the Riccati lower envelope at $\bar{p}$; exact evaluation of that policy moves to its fixed point $p_{\bar{k}}$. The optimal Riccati fixed point is p_∞.

Worked Example 5.1 (Lookahead corrects an imperfect terminal coefficient)**.** Let

$$a = 1, \qquad b = 2, \qquad q = 1, \qquad r = 0.5.$$

Then

$$p_\infty = \frac{2 + \sqrt{6}}{4} \approx 1.1124. \tag{5.44}$$

Suppose the terminal coefficient is only $\tilde{p} = 0.2p_\infty$. For a lookahead depth ℓ, the first feedback gain is

$$k_\ell = k\big(\mathscr{F}^{\ell-1}(\tilde{p})\big). \tag{5.45}$$

The terminal coefficient is initially poor, but one Riccati update already moves it close to the positive fixed point. The actual policy loss is smaller than the coefficient error because the online minimization removes the first-order policy error near the optimum.

Figures 5.5 and 5.6 should be read with two cautions. First, the result is not "longer horizon is always better" without conditions. It relies on a well-behaved Riccati iteration and exact online optimization. Second, the actual online action is improved even though the terminal coefficient itself remains approximate. This is the behavior the book will seek in nonlinear mobility problems, where closed-form analysis is no longer available.

Table 5.1: Verified scalar lookahead results for $\tilde{p} = 0.2p_\infty$. The policy loss evaluates the infinite-horizon cost of the feedback actually induced by the lookahead calculation.

ℓ	$\mathscr{F}^{\ell-1}(\tilde{p})$	k_ℓ	$a - bk_\ell$	$p_{k_\ell} - p_\infty$
1	0.222474	0.320131	0.359739	9.5135×10^{-2}
2	1.080033	0.448134	0.103732	9.1938×10^{-6}
3	1.112034	0.449476	0.101048	9.5748×10^{-10}
4	1.112369	0.449490	0.101021	9.9698×10^{-14}

Implementation note

The script writes the plotted coefficients, gains, closed-loop eigenvalues, and policy costs to `figures/generated/scalar_horizon_data.csv`. The figure is therefore an output of an auditable numerical experiment rather than a manually edited illustration.

5.8 Vehicle Example: Linearized Lateral-Error Regulation

Consider a kinematic bicycle traveling at constant longitudinal speed v near a reference path. Define e_y as signed lateral displacement and e_ψ as the vehicle heading minus the path-tangent angle. For small e_ψ, small steering angle δ, and slowly varying reference curvature κ_{ref}, the continuous-time error kinematics are locally

$$\dot{e}_y = v \sin e_\psi \approx v e_\psi, \qquad \dot{e}_\psi \approx \frac{v}{L}\delta - v\kappa_{\text{ref}}. \tag{5.46}$$

Choose the nominal feedforward steering $\delta_{\text{ff}} \approx L\kappa_{\text{ref}}$ and define the feedback correction $\nu = \delta - \delta_{\text{ff}}$. Forward-Euler discretization with sampling time T_s then gives

$$\begin{bmatrix} e_{y,k+1} \\ e_{\psi,k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & vT_s \\ 0 & 1 \end{bmatrix}}_{A} \begin{bmatrix} e_{y,k} \\ e_{\psi,k} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ vT_s/L \end{bmatrix}}_{B} \nu_k. \tag{5.47}$$

Thus Equation (5.47) is not a new vehicle model: it is the local small-angle, constant-speed, forward-Euler approximation of the kinematic bicycle error equations. See Rajamani (2012) for the underlying vehicle-modeling framework.

Mobility case: Linear lateral-regulation baseline

The pilot simulation uses

$$v = 12\,\text{m/s}, \quad T_s = 0.1\,\text{s}, \quad L = 2.8\,\text{m}, \quad Q = \text{diag}(4, 1), \quad R = 0.3,$$

from initial error $e_y(0) = 0.15\,\text{m}$ and $e_\psi(0) = 2°$. It compares:

1. infinite-horizon LQR;

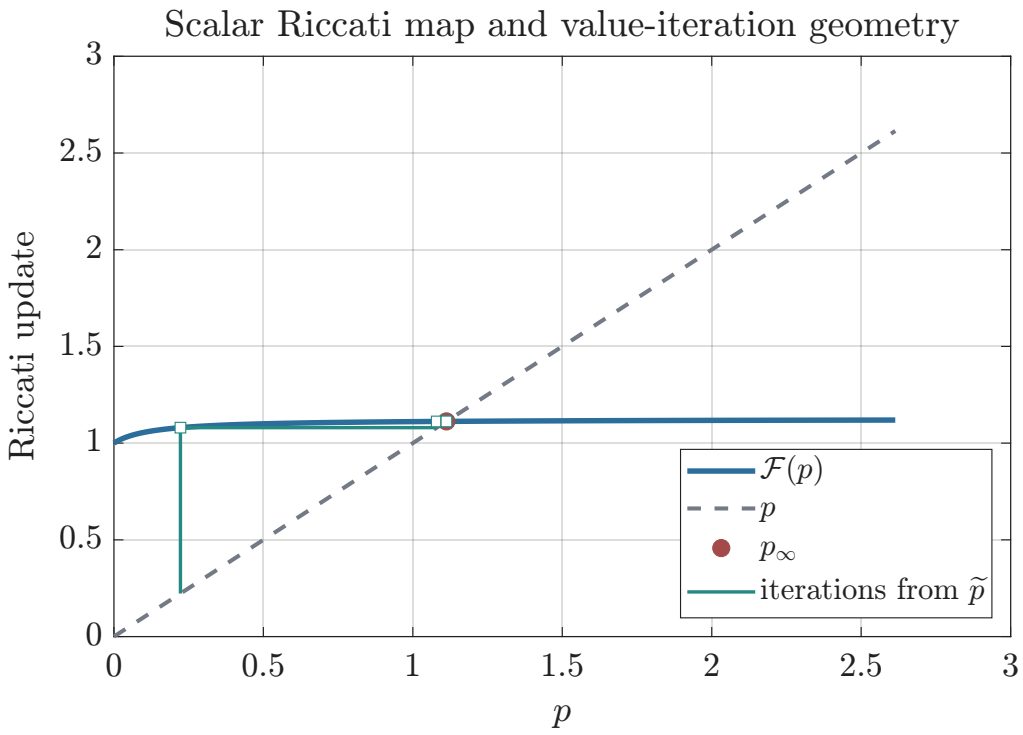


Figure 5.5: An original numerical visualization of the scalar Riccati map in worked example 5.1. Stair steps show value-iteration updates from the imperfect terminal coefficient. The figure is generated by `code/pilot_lqr/generate_figures.m`.

2. a two-step receding-horizon controller with zero terminal cost; and
3. a two-step controller with terminal matrix P_∞.

The third controller reproduces infinite-horizon LQR, as predicted by proposition 5.2. The zero-terminal-cost controller is a different stationary feedback because its short lookahead undervalues the remaining lateral error.

This model is deliberately modest. The later slalom case will add speed dynamics, steering and tire limits, actuator lag, collision geometry, and model mismatch. Its educational purpose here is to establish a result that remains visible after those complications are added: terminal information determines what a short online horizon can see.

5.9 From Receding-Horizon LQR to Constrained MPC

The scalar and vehicle examples above used unconstrained quadratic problems so that every object could be computed analytically. MPC keeps the same finite-lookahead logic while allowing nonlinear dynamics and explicit state and input constraints. For a deterministic system

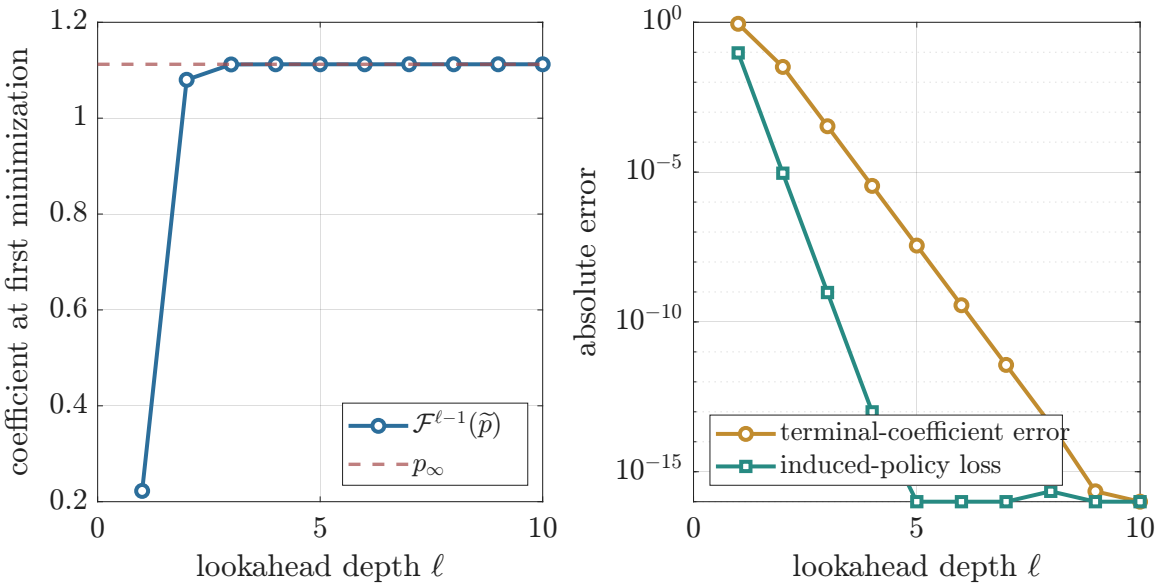


Figure 5.6: Effect of lookahead depth for the scalar example. The left panel shows the coefficient presented to the first minimization; the right panel separates terminal-coefficient error from the infinite-horizon loss of the induced feedback policy.

$x_{k+1} = f(x_k, u_k)$, the online problem at measured state x_k is

$$V_\ell(x_k) = \min_{u_{k|k},\dots,u_{k+\ell-1|k}} \quad \sum_{j=0}^{\ell-1} g(x_{k+j|k}, u_{k+j|k}) + V_f(x_{k+\ell|k}) \tag{5.48a}$$

$$\begin{aligned} \text{subject to} \quad & x_{k|k} = x_k, \\ & x_{k+j+1|k} = f(x_{k+j|k}, u_{k+j|k}), \\ & x_{k+j|k} \in \mathcal{X}, \quad u_{k+j|k} \in \mathcal{U}, \quad j = 0, \dots, \ell-1, \\ & x_{k+\ell|k} \in \mathcal{X}_f. \end{aligned}$$

Only $u^*_{k|k}$ is applied; the optimization is rebuilt from the next measured state. In the unconstrained LQR case, this reduces exactly to the backward Riccati calculation already studied.

Without constraints, the lookahead value is $T^\ell V_f$. This statement does not mean that dynamic programming is inherently an online algorithm: an exact DP policy may be computed and stored offline. Online computation appears here because the Bellman minimizations are deliberately re-evaluated at the measured state.

AI/RL terminology bridge

In planning terminology, MPC searches forward from the current state and uses V_f as a leaf evaluator. A learned value network can supply this leaf estimate without being updated online. The controller then performs online policy improvement, but not necessarily online learning.

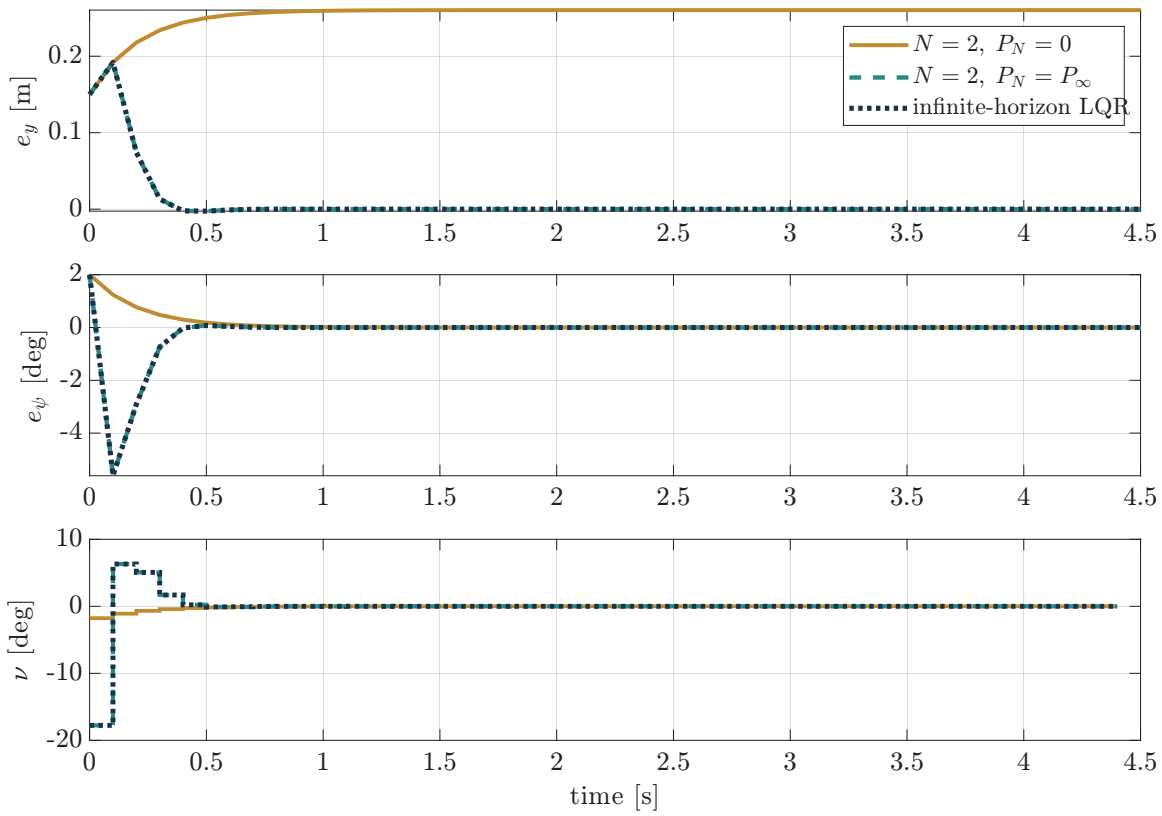


Figure 5.7: Linearized vehicle lateral-error regulation. Exact terminal value makes the two-step lookahead controller coincide with infinite-horizon LQR; zero terminal value produces a different short-horizon feedback. This is a regulation example, not yet the nonlinear slalom benchmark.

5.9.1 Terminal ingredients and the shift argument

For unconstrained LQR, P_∞ alone is an exact terminal weight. With constraints, a standard nominal MPC design uses three mutually supporting objects (Mayne et al. 2000; Rawlings et al. 2017):

1. a local terminal policy $\kappa_f(x)$, often $-K_\infty x$;
2. a terminal set $\mathcal{X}_f$ that is invariant under κ_f and respects the constraints; and
3. a nonnegative terminal value V_f that decreases under κ_f.

For every $x \in \mathcal{X}_f$, require

$$\kappa_f(x) \in \mathcal{U}, \qquad f(x, \kappa_f(x)) \in \mathcal{X}_f, \tag{5.49}$$

$$V_f(f(x, \kappa_f(x))) - V_f(x) \le -g(x, \kappa_f(x)). \tag{5.50}$$

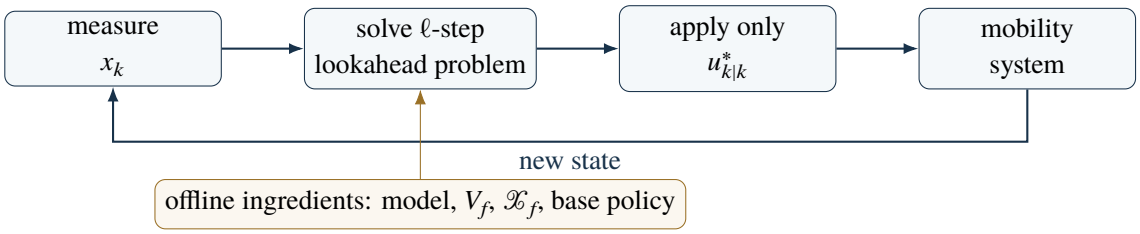


Figure 5.8: Receding-horizon control as online lookahead supported by offline ingredients. Resolving changes the action at each measured state even when the offline model and terminal approximation remain fixed.

Theorem 5.1 (Recursive feasibility and value decrease by shifting). *Suppose Equation* (5.48) *is feasible at x_k, its deterministic model is exact, and the optimizer returns a global optimum. If Equations* (5.49) *and* (5.50) *hold, then the shifted input sequence is feasible at x_{k+1}. Consequently, the receding-horizon controller is recursively feasible and*

$$V_\ell(x_{k+1}) - V_\ell(x_k) \le -g(x_k, u_k). \tag{5.51}$$

Proof. Let the optimal predicted input sequence at x_k be $(u^*_{k|k}, \dots, u^*_{k+\ell-1|k})$. After applying its first element, shift the remaining elements one position and append $\kappa_f(x^*_{k+\ell|k})$. Exact prediction identifies the next measured state with the first predicted successor. Terminal invariance and constraint admissibility make the shifted sequence feasible. Relative to the previous optimal cost, its cost removes $g(x_k, u_k)$ and adds

$$g(x^*_{k+\ell|k}, \kappa_f(x^*_{k+\ell|k})) + V_f(x^*_{k+\ell+1|k}) - V_f(x^*_{k+\ell|k}),$$

which is nonpositive by Equation (5.50). The new optimum cannot exceed the cost of this feasible candidate, proving Equation (5.51). □

If $g(x, u)$ lower-bounds a positive-definite function of the state and the usual continuity and boundedness conditions make V_ℓ a Lyapunov function, Equation (5.51) yields the corresponding asymptotic stability result. Those extra conditions matter: recursive feasibility and value decrease should not be promoted to a blanket nonlinear stability claim.

Likewise, an arbitrary learned terminal value is not automatically a safe MPC terminal ingredient. It may improve performance while violating Equation (5.50); model error or an inexact solve can also invalidate the shift proof.

5.10 Terminology Across DP, MPC, and Model-Based RL

The table translates computational roles; it does not assert that the methods are identical. Exact DP may compile a policy offline and require no online optimization. MPC emphasizes constraints and continuous optimization, while RL often emphasizes learning the policy, value, or model from samples.

Table 5.2: A compact terminology map. The DP/rollout column describes an online lookahead implementation, not dynamic programming as a whole.

Object	DP / rollout	Model predictive control	Model-based RL / planning
Tail information	terminal value approximation	terminal cost and terminal set	leaf evaluator or critic
Online decision	optional Bellman minimization at the current state	constrained finite-horizon optimization	model-based search or rollout
Applied action	first minimizer	first element of the optimal sequence	root action
Repeated use	rollout/lookahead policy	receding horizon	replanning or online play

5.11 Design Conditions and Limitations

Conditions to check

1. **Check closed-loop stability after value approximation.** A small fitting error in a global norm does not ensure that $A - BK(\widetilde{P})$ is stable.
2. **Match the horizon and solver to the control deadline.** A theoretical improvement from lookahead is lost if the optimizer misses the control deadline or returns a poor local solution.
3. **Quantify prediction-model error.** Replanning corrects the state estimate at the next step, but it does not automatically protect constraints between updates.
4. **Verify terminal ingredients as a set.** A terminal matrix copied from an unconstrained LQR design does not by itself establish recursive feasibility for a constrained nonlinear model.
5. **Treat constraints inside the optimization.** Clipping an LQR command changes the feedback law without solving the constrained horizon problem and can destroy the claimed analysis.
6. **Separate empirical evidence from guarantees.** No observed violations over a test set are useful evidence, but they are not equivalent to an invariant-set or robust-feasibility result.

5.12 Reproducible Implementation Pattern

The essential finite-horizon computation requires only a backward Riccati loop. The complete script used for this chapter is `code/pilot_lqr/generate_figures.m`. Its core

operation is equivalent to:

Listing 5.1: Backward Riccati recursion in Python-like pseudocode.

```
P = terminal_matrix
for k in reversed(range(N)):
    K[k] = solve(R + B.T @ P @ B, B.T @ P @ A)
    P = Q + A.T @ P @ A - A.T @ P @ B @ K[k]

u0 = -K[0] @ x0
```

For MPC, the same conceptual loop is embedded in a constrained optimizer and rerun after each state measurement. Later chapters will separate three error sources that coincide in an ideal LQR calculation: value approximation error, model error, and online optimization error.

The current script is suitable for reproducing the chapter figures and for a guided in-class demonstration. It is not yet a release-ready graded lab: that version still needs a student starter file, explicit tasks and checks, a constrained-QP extension, and cross-platform testing. The run instructions and safe parameter changes are documented in `code/pilot_lqr/README.md`.

5.13 Summary

- Finite-horizon LQR is exact backward DP because quadratic values are closed under the Bellman operation.
- The DARE is Bellman's fixed-point equation on quadratic functions, and Riccati iteration is structured value iteration.
- An exact infinite-horizon terminal value makes any positive lookahead depth recover the optimal first action.
- An approximate terminal value is not the same as the cost of its induced policy. Near the optimum, online minimization can turn first-order value error into second-order policy loss.
- VI takes a Bellman backup, whereas PI selects a touching policy operator and evaluates its fixed point. Rollout is this improvement step applied to the exact value of a base policy; multistep lookahead first moves the effective evaluator through repeated Bellman backups.
- In scalar LQR, the Riccati map is the lower envelope of affine policy-specific maps. Greedy improvement selects a touching policy line, and policy evaluation is a Newton-type fixed-point step.
- MPC repeats this lookahead after every measurement and incorporates constraints, but stability requires more than a plausible terminal cost.
- The linearized lateral-control example establishes the unconstrained regulation baseline for the later constrained slalom benchmark.

5.14 Exercises

Exercise 5.1 (Bellman-envelope geometry). Consider the three affine maps used in Figure 5.2:

$$T_1(s) = 0.9 + 0.55s, \qquad T_2(s) = 1.25 + 0.25s,$$
$$T_3(s) = 1.55 + 0.05s, \qquad T(s) = \min_i T_i(s).$$

Compute the envelope breakpoints and the fixed point of T. Starting from $\bar{s} = 0.8$, compute one VI update, identify the touching policy map, and solve its fixed point. Then start from $\tilde{s} = 0.2$ and reproduce the three-step construction in Figure 5.3. Explain why the leaf evaluator and the achieved policy cost are different objects.

Exercise 5.2 (Scalar Riccati equation). Starting from Equation (5.35), derive Equations (5.36), (5.37) and (5.39). Identify the interval of p for which the one-step lookahead gain is stabilizing.

Exercise 5.3 (Terminal-value invariance). Prove proposition 5.2 directly from the finite-horizon Bellman recursion, without invoking the Riccati fixed-point notation. Explain why the proof also applies when the minimizing action is not unique.

Exercise 5.4 (Value error versus policy loss). For worked example 5.1, reproduce the horizon sweep for $\tilde{p}/p_\infty \in \{0, 0.2, 0.5, 1.5, 3\}$. Plot separately:

1. $|\mathscr{F}^{\ell-1}(\tilde{p}) - p_\infty|$;
2. $|k_\ell - k_\infty|$; and
3. $p_{k_\ell} - p_\infty$.

Mark horizons that generate an unstable feedback instead of assigning them a finite cost.

Exercise 5.5 (Policy iteration). Choose a stabilizing scalar gain $k^{(0)}$. Use Equation (5.38) for policy evaluation and Equation (5.37) for policy improvement. Compare the number of PI and VI iterations required to reach the same tolerance in p.

Exercise 5.6 (Lateral-control terminal value). Reproduce Figure 5.7. Repeat the experiment for horizons $N \in \{1, 2, 3, 5, 10\}$ and initial speeds from 5 m/s to 25 m/s. Explain why fixing a discrete-time gain while changing speed is not the same as redesigning the controller at each operating point.

Exercise 5.7 (Constrained MPC). Add the constraints $|\delta| \le 0.35$ rad and $|e_y| \le 1.2$ m to the lateral model. Compare a genuine constrained QP with clipped LQR. Report feasibility, maximum lateral error, control effort, and per-step solve time.

Exercise 5.8 (Challenge: toward the slalom benchmark)**.** Replace the straight-path regulation model by a sinusoidal reference curvature. Introduce an unknown steering lag and fit a residual model from closed-loop data. Compare:

1. nominal MPC;
2. MPC with the learned residual model; and
3. the same two controllers with a learned terminal value.

Separate model prediction error from closed-loop cost and constraint violations.

6 Approximation in Policy Space

Chapter 4 approximated a value or Q-factor and then asked how that object produces a control. This chapter stores the control law itself:

$$u = \mu_\theta(x) \quad \text{or} \quad u \sim \pi_\theta(\cdot \mid x). \tag{6.1}$$

The resulting actor is fast to execute and natural for continuous actions, but it does not remove the need for evaluation. A policy-gradient or actor–critic algorithm still needs returns, a critic, or both to judge which parameter change is an improvement. For this reason, PPO, TD3, and SAC will be treated as variants of approximate policy iteration rather than as isolated acronyms.

The second theme is equally important. A learned actor need not be the last decision maker. It can be a base policy for rollout, a warm start or local center for MPC, and a data generator for a terminal critic. Online lookahead then adapts the stored policy to the state, constraints, and model information available at execution time.

Learning objectives

After studying this chapter, the reader should be able to:

1. distinguish deterministic and stochastic policy architectures and enforce basic action structure in their outputs;
2. construct a policy by supervised imitation, rollout distillation, gradient optimization, or derivative-free search;
3. derive the stochastic policy-gradient identity and explain the roles of occupancy, Q-factors, baselines, and advantages;
4. interpret actor–critic training as approximate policy evaluation followed by restricted policy improvement;
5. derive cost-form PPO, DDPG/TD3, and entropy-regularized SAC objectives;
6. distinguish on-policy and off-policy data use and identify the policy evaluated by each critic target;
7. select among DQN, PPO, TD3, SAC, and structured policy search using action space, data relation, and deployment needs;
8. use a learned actor as a base policy, optimizer warm start, or local action-search center and a critic as a terminal value;
9. explain when an online improvement guarantee survives approximation and when it becomes an empirical claim; and

10. audit an actor, critic, and online improvement layer separately under model shift and constraints.

Chapter roadmap

The chapter first defines the policy object and the ways it can be fitted. It then derives policy-gradient and actor–critic machinery before locating PPO, TD3, and SAC within that common loop. Derivative-free search provides a second route through policy space. The last third of the chapter combines the actor and critic with online lookahead and verifies the architecture on a continuous mobility surrogate.

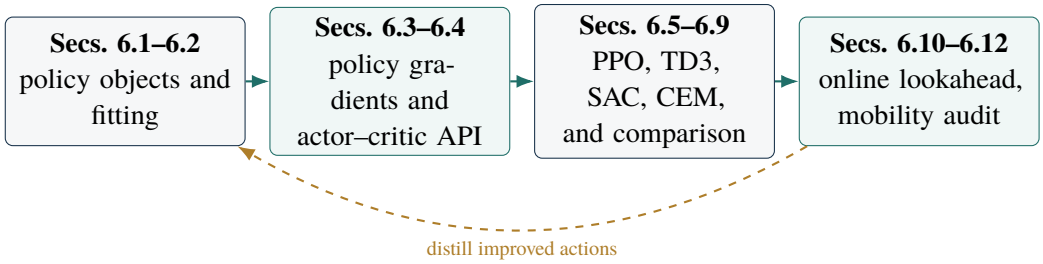


Figure 6.1: Original map of Chapter 6. Offline policy approximation supplies a fast actor; online improvement can use that actor and return improved actions to the next offline training cycle.

6.1 Why Approximate a Policy Directly?

Suppose a value approximator $\tilde{J}(x; r)$ is available. A control still has to solve

$$\tilde{\mu}(x) \in \arg\min_{u \in \mathcal{U}(x)} \left\{ g(x,u) + \alpha \mathbb{E}[\tilde{J}(x') \mid x, u] \right\}. \tag{6.2}$$

This is attractive when the minimization is small or when its online solution enforces current constraints. It can be expensive when u is continuous, the model is nonlinear, or the action must be produced at high rate. A policy approximator replaces the repeated optimization by a forward evaluation.

Policy approximation also answers a different representation question. A Q-table naturally defines a finite-action actor by an arg min, but a continuous steering or torque command cannot be recovered by enumerating an unrestricted action space. A differentiable map $\mu_\theta(x)$ represents that continuous decision directly. This is one reason actor–critic methods are prominent in continuous-control RL.

The convenience comes with three losses of information.

1. A compressed actor may not represent the greedy action at every state.

2. The actor alone does not report how undesirable its action is or how close a competing action may be; a critic or online optimizer supplies that comparison.
3. A frozen actor does not automatically adapt to changed constraints, dynamics, or objectives.

Thus value and policy space are complementary axes. A critic supports evaluation and lookahead; an actor amortizes an optimization into a fast map.

Table 6.1: Roles of stored value- and policy-space objects.

Stored object	Direct output	Main advantage	Missing layer
$\tilde{J}(x)$	future-cost estimate	compact terminal evaluator; supports changing actions and constraints	model or samples plus an online action comparison
$\widetilde{Q}(x,u)$	state–action comparison	finite-action greedy control; critic for continuous actor	continuous inner minimization unless an actor is added
$\mu_\theta(x)$	deterministic action	low execution cost; natural continuous output	uncertainty/exploration model and explicit action comparison
$\pi_\theta(u \mid x)$	action distribution	exploration and multi-modal/randomized policies	declared execution statistic and safety layer

Control viewpoint

The phrase "policy network" describes a function architecture, not how the controller was obtained. The same network may imitate MPC, maximize a PPO surrogate, minimize a TD3 critic, or be optimized by random search. The target mechanism and deployment layer must be stated separately.

6.2 Deterministic and Stochastic Policy Architectures

A deterministic actor is a map

$$\mu_\theta : \mathcal{X} \to \mathcal{U}, \qquad u = \mu_\theta(x). \tag{6.3}$$

It is suitable when repeatable execution is desired and exploration can be handled during training. For box constraints $u_i \in [\underline{u}_i, \overline{u}_i]$, a neural output $z_\theta(x)$ can be transformed as

$$\mu_{\theta,i}(x) = \frac{\overline{u}_i + \underline{u}_i}{2} + \frac{\overline{u}_i - \underline{u}_i}{2} \tanh z_{\theta,i}(x). \tag{6.4}$$

This enforces the actuator box, but it does not enforce state constraints or coupled limits such as a friction ellipse. Those may require a structured output, projection, control barrier/safety filter, or online constrained optimization.

A stochastic actor represents a conditional distribution

$$u \sim \pi_\theta(\cdot \mid x). \tag{6.5}$$

For continuous actions, a common choice is a Gaussian whose mean and log standard deviation are network outputs, followed by a squashing transform. For a finite maneuver library, a softmax output supplies categorical probabilities. Stochasticity supports exploration and can represent genuine randomization, but deployment must state whether it applies a sample, mean, mode, or a separately filtered action.

The state representation matters as much as the output. An MLP is appropriate for a fixed vector; CNNs can process spatial observations and RNNs can maintain a latent summary of observation history. These are architectural choices from Chapter 3. An RNN does not make a partially observed problem Markov; it attempts to construct a useful information state.

6.2.1 Policy fitting from action labels

If an expert, optimizer, or rollout supplies pairs $(x_i, u_i^{\mathrm{tar}})$, a deterministic actor can be fitted by

$$\hat{\theta} \in \arg\min_\theta \sum_{i=1}^{M} w_i \|\mu_\theta(x_i) - u_i^{\mathrm{tar}}\|_{R_i}^2 + \lambda \|\theta\|_2^2. \tag{6.6}$$

The first term asks the actor to reproduce the supplied actions. The second term is not another control cost: it selects a preferred parameterization when several actors fit the labels similarly. Chapter 3, especially Section 3.4.2, distinguishes ℓ_2, sparsity, smoothness, early stopping, dropout, and control-sensitivity regularization. A larger λ can make the actor smoother or smaller but can also move it farther from an important expert action. It cannot repair a poor expert, missing state coverage, or a biased rollout label. This includes behavioral cloning and MPC policy approximation. When several actions have nearly equal Q-factor, a Q-weighted loss can reduce the importance of harmless label discrepancies. Conversely, a small action error near an active constraint may be control critical. Validation should therefore report closed-loop cost and violations, not only action RMSE.

For a stochastic actor, action labels can define the negative log-likelihood loss

$$\mathscr{L}_{\mathrm{NLL}}(\theta) = -\sum_{i=1}^{M} w_i \log \pi_\theta(u_i^{\mathrm{tar}} \mid x_i). \tag{6.7}$$

If a teacher supplies an entire action distribution, cross-entropy or KL divergence preserves more information than one sampled label.

6.2.2 Rollout and optimizer distillation

The teacher need not be a human. A base policy and online lookahead can create an improved action $\hat{u}(x)$ at selected states; the actor then fits $\hat{u}(x)$. This is rollout or planning distillation

(Bertsekas 2019; Bertsekas 2022). It shifts computation from repeated online search to offline data generation. The resulting actor should not be called the rollout policy without qualification: regression error and distribution shift can make it different away from the sampled states.

An effective loop alternates:

$$\text{actor rollout} \longrightarrow \text{visited states} \longrightarrow \text{online improved labels} \longrightarrow \text{actor refit.} \tag{6.8}$$

Collecting new labels under the refitted actor reduces the mismatch between a fixed offline data distribution and the states generated by the deployed policy. It does not by itself certify safety on unvisited states.

6.3 Policy Objectives and the Policy-Gradient Identity

Let the initial state have distribution ρ_0 and let π_θ be a differentiable stochastic policy. Its discounted objective is

$$\mathcal{J}(\theta) = \mathbb{E}_{x_0 \sim \rho_0,\, \pi_\theta}\left[\sum_{k=0}^{\infty} \alpha^k g(x_k, u_k)\right]. \tag{6.9}$$

Equation (6.9) turns controller design into a finite-dimensional parameter optimization. For each sampled x_0, the policy randomizes $u_k \sim \pi_\theta(\cdot \mid x_k)$, the plant generates the next state, and the inner sum records the resulting discounted physical cost. The expectation averages all three sources of variation: the initial-state distribution, policy sampling, and stochastic transitions. Thus ρ_0 declares where the policy is to perform well, while α declares how strongly delayed costs matter. This scalar objective is neither a Bellman equation nor a pointwise guarantee for every initial state. Two studies with different ρ_0 are solving different policy-fitting problems even if the dynamics and stage cost are unchanged. Define the normalized discounted state-occupancy distribution

$$d_{\pi_\theta}(x) = (1-\alpha)\sum_{k=0}^{\infty} \alpha^k \Pr_{\pi_\theta}(x_k = x). \tag{6.10}$$

For continuous states this notation denotes a density or measure rather than a finite probability table.

Proposition 6.1 (Stochastic policy-gradient identity). *Assume the policy is differentiable, its support does not change with θ, and interchange of differentiation and expectation is valid. Then the cost gradient satisfies*

$$\nabla_\theta \mathcal{J}(\theta) = \frac{1}{1-\alpha}\mathbb{E}_{\substack{x \sim d_{\pi_\theta} \\ u \sim \pi_\theta(\cdot \mid x)}}\left[Q_{\pi_\theta}(x,u)\nabla_\theta \log \pi_\theta(u \mid x)\right]. \tag{6.11}$$

Direct differentiation of Equation (6.9) appears to require the derivatives of every future state with respect to θ. That route is difficult for a stochastic plant, a black-box simulator, or a sampled physical system. The policy-gradient theorem replaces it by an equivalent likelihood-ratio form.

For a finite trajectory $\zeta = (x_0, u_0, \dots, x_T)$,

$$p_\theta(\zeta) = \rho_0(x_0) \prod_{t=0}^{T-1} \pi_\theta(u_t \mid x_t) P(x_{t+1} \mid x_t, u_t), \tag{6.12}$$

and the log-derivative identity gives

$$\nabla_\theta \mathbb{E}_{p_\theta}[F(\zeta)] = \mathbb{E}_{p_\theta}[F(\zeta) \nabla_\theta \log p_\theta(\zeta)], \tag{6.13}$$

$$\nabla_\theta \log p_\theta(\zeta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(u_t \mid x_t). \tag{6.14}$$

The transition factors in Equation (6.12) do not depend on θ, so their log gradients are zero. Their effect has not been discarded: they still determine which trajectories occur, the occupancy d_{π_θ}, and Q_{π_θ}. Causality then removes costs incurred before action u_t from that action's expected score, leaving the future-cost factor $Q_{\pi_\theta}(x_t, u_t)$ in Equation (6.11). This is the hard-to-differentiate trajectory objective rewritten as an expectation of quantities that can be sampled from interaction. The result is the policy-gradient theorem of Sutton, McAllester, et al. (2000) and is developed from the same likelihood-ratio viewpoint in Bertsekas (2025, Sec. 3.5).

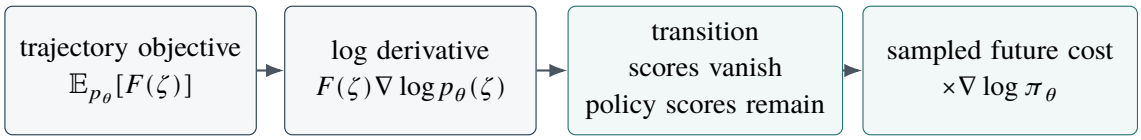


Figure 6.2: Original likelihood-ratio route to the stochastic policy gradient. The equivalent score form avoids differentiating the transition model, while the model still shapes sampled trajectories and Q-factors.

For any state-dependent baseline $b(x)$,

$$\mathbb{E}_{u \sim \pi_\theta}\left[b(x) \nabla_\theta \log \pi_\theta(u \mid x)\right] = 0. \tag{6.15}$$

For a finite action set, the reason is explicit:

$$\sum_u \pi_\theta(u \mid x) b(x) \nabla_\theta \log \pi_\theta(u \mid x) = b(x) \sum_u \nabla_\theta \pi_\theta(u \mid x) = b(x) \nabla_\theta 1 = 0. \tag{6.16}$$

The density version replaces the sum by an integral and requires the usual interchange conditions and parameter-independent support. An action- dependent baseline is not covered by this cancellation unless an additional correction is included. Therefore Q_{π_θ} in Equation (6.11) may be replaced by the cost advantage

$$A_{\pi_\theta}(x, u) = Q_{\pi_\theta}(x, u) - J_{\pi_\theta}(x). \tag{6.17}$$

The baseline does not change the expected gradient but can greatly reduce its variance. At a difficult state, every action may have a large future cost; that common state-level magnitude is

irrelevant to which action is locally preferable but multiplies every noisy score sample. Subtracting $J_\pi(x) = \mathbb{E}_{u\sim\pi}[Q_\pi(x,u)]$ centers the action comparison at that state and leaves the relative signal. It is a standard, convenient baseline, not always the mathematically variance-minimizing one for every parameter coordinate. The sign convention matters: a negative cost advantage means the action is better than the policy's average action at that state.

With sampled advantages $\widehat{A}_t$, a basic gradient step is

$$\theta^+ = \theta - \eta \frac{1}{M} \sum_{t=1}^{M} \widehat{A}_t \nabla_\theta \log \pi_\theta(u_t \mid x_t). \tag{6.18}$$

The theorem is an identity, not a promise that a finite, noisy step improves the closed loop. The stepsize, critic error, data distribution, and parameter class all affect the realized update.

6.3.1 Deterministic policy gradient

For a differentiable deterministic actor $u = \mu_\theta(x)$, a small parameter perturbation first changes the action and then changes the Q-factor:

$$d\mu_\theta(x) = \nabla_\theta \mu_\theta(x)\, d\theta, \qquad dQ_\mu(x, \mu_\theta(x)) = \nabla_u Q_\mu(x,u)|^{\mathsf{T}}_{u=\mu_\theta(x)} d\mu_\theta(x). \tag{6.19}$$

Averaging this chain rule over the discounted state distribution yields the deterministic policy-gradient theorem (Silver et al. 2014). In an off-policy implementation with replay distribution ν, the actor direction used by DDPG and TD3 is approximated as

$$\nabla_\theta \mathcal{J}_\nu(\theta) \approx \mathbb{E}_{x\sim\nu}\left[\nabla_\theta \mu_\theta(x) \nabla_u Q_{\mu_\theta}(x,u)|_{u=\mu_\theta(x)}\right]. \tag{6.20}$$

The product has two distinct factors. The actor Jacobian says how each parameter moves the commanded action; the critic action gradient says which local action direction decreases future cost. The state-distribution effect is accounted for by the policy-gradient theorem rather than being naively ignored in Equation (6.19). The replay form optimizes an off-policy surrogate weighted by ν; replacing the target occupancy by arbitrary replay data is exact only under the objective and coverage conditions stated by the deterministic policy-gradient result.

The method differentiates through the learned critic, not through the unknown physical dynamics. This is computationally attractive, but it transfers a strong responsibility to the critic's *action slope*. A small Q-value RMSE can coexist with a wrong $\nabla_u \widetilde{Q}$, which can point the actor toward an artificial low-cost region. Twin critics, delayed updates, and target smoothing in TD3 address parts of this approximation problem; they do not eliminate the need for action coverage or controller-level validation.

6.3.2 Performance difference and the meaning of improvement

The policy-gradient theorem gives a local derivative. A complementary identity describes the finite difference between two policies. Let $\mathcal{J}(\pi) = \mathbb{E}_{x_0\sim\rho_0}[J_\pi(x_0)]$.

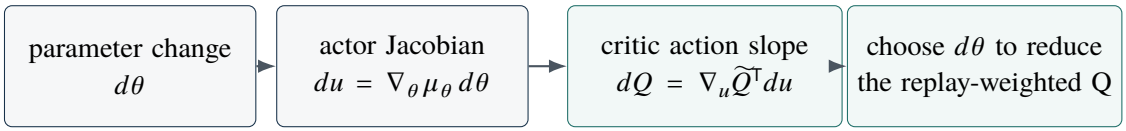


Figure 6.3: Original chain-rule interpretation of deterministic policy gradient. The actor supplies the parameter-to-action sensitivity and the critic supplies the action-to-cost sensitivity.

Proposition 6.2 (Discounted performance-difference identity). *For two stationary policies π and π', under the discounted assumptions of Chapter 2,*

$$\mathcal{J}(\pi') - \mathcal{J}(\pi) = \frac{1}{1-\alpha} \mathbb{E}_{\substack{x \sim d_{\pi'} \\ u \sim \pi'(\cdot|x)}} [A_\pi(x,u)]. \tag{6.21}$$

To see the identity, expand $A_\pi(x_t, u_t) = g_t + \alpha J_\pi(x_{t+1}) - J_\pi(x_t)$ along a trajectory of π'. The discounted value terms telescope, leaving the cost difference from the common initial distribution. If π' chooses nonpositive cost advantage at every state, it is no worse than π. Exact greedy PI enforces such a pointwise statement (Sutton and Barto 2018; Bertsekas 2025).

Parameterized improvement is weaker. A PPO or actor-gradient update controls an empirical surrogate under states sampled mainly from d_π, while the identity contains $d_{\pi'}$. A large policy change can alter the states that are visited and invalidate the local comparison. Ratio clipping, trust-region ideas, conservative steps, and repeated data collection manage this occupancy shift; they do not turn a fitted surrogate into a pointwise improvement proof.

The occupancy detail is not cosmetic. The new actor can enter a region that was rare under the old actor, where neither the critic nor the advantage estimates were accurate. Exact greedy PI avoids this particular sampling gap by checking an improvement inequality at every state. A clipped or trust- region update instead keeps the new policy close on sampled states, making the substitution $d_{\pi'} \approx d_\pi$ more plausible but not exact.

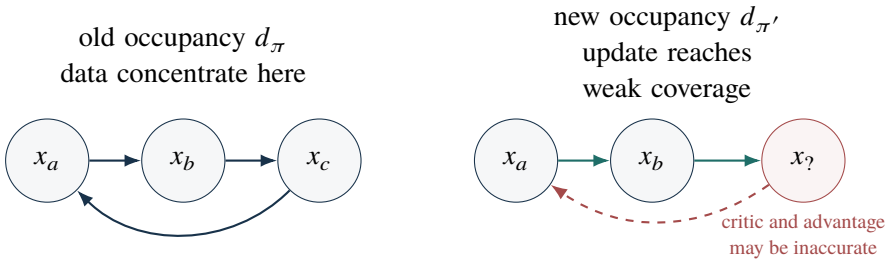


Figure 6.4: Original occupancy-shift interpretation of approximate policy improvement. A surrogate fitted under the old policy need not predict the states emphasized by the updated policy.

6.4 Actor–Critic as Approximate Policy Iteration

Approximate policy iteration (API) separates two questions:

$$\text{evaluation:} \qquad \varphi \leftarrow \text{FitCritic}(\pi_\theta), \tag{6.22}$$

$$\text{improvement:} \qquad \theta \leftarrow \text{ImproveActor}(\theta, \varphi). \tag{6.23}$$

An actor–critic method interleaves approximate versions of these operations. The critic may estimate J_{π_θ}, Q_{π_θ}, or an advantage; the actor makes a restricted move inside its parameter class. Unlike exact PI, neither stage normally converges before the other changes.

Chapter 4 already approximated the evaluation object while the policy-improvement rule was explicit. Here the policy is approximated as well. PPO represents a restricted stochastic improvement; DDPG and TD3 use a deterministic actor that descends a Q critic; SAC performs entropy-regularized stochastic improvement. These algorithms are therefore not detached from API: they choose different critic targets, policy classes, data distributions, and restrictions on the improvement step.

Advantage enters because policy improvement is an *action comparison at a fixed state*. Absolute future cost includes how difficult the state is; $A_\pi(x,u) = Q_\pi(x,u) - J_\pi(x)$ removes that state-level baseline and reports whether u is better or worse than the policy's usual action mixture. The critic need not have a separate advantage output. A temporal-difference residual supplies a sampled, bootstrapped action-comparison signal.

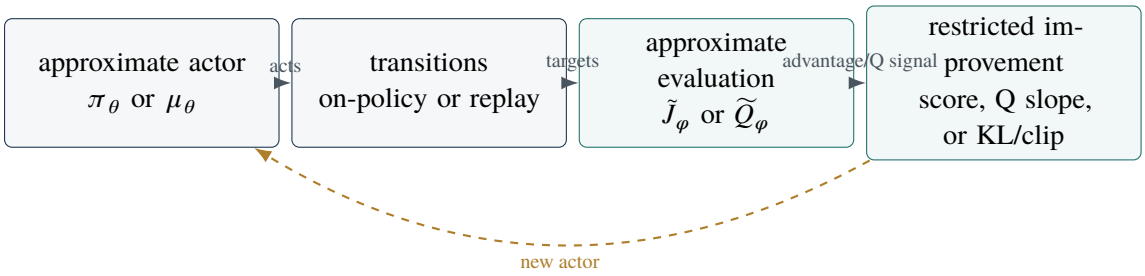


Figure 6.5: Original actor–critic reading as interleaved approximate policy evaluation and restricted approximate policy improvement.

6.4.1 From TD errors to generalized advantage estimates

For a cost critic $\tilde{J}(x;\varphi)$, define

$$\delta_t^\varphi = g_t + \alpha \tilde{J}(x_{t+1};\varphi) - \tilde{J}(x_t;\varphi). \tag{6.24}$$

The quantity δ_t^φ is the one-step *TD error*: observed stage cost plus the bootstrapped next value minus the current value estimate. If $\tilde{J} = J_\pi$ exactly, then

$$\mathbb{E}[\delta_t^\varphi \mid x_t = x, u_t = u] = Q_\pi(x,u) - J_\pi(x) = A_\pi(x,u). \tag{6.25}$$

Thus one TD error is a noisy *sample whose conditional mean is the advantage*; it is not literally the deterministic advantage in every sampled transition. With an approximate critic, its conditional mean also contains critic error. This distinction answers why TD error appears in the actor update: it is an inexpensive action-comparison estimate available from a single transition. The n-step cost-advantage estimate can be written

$$\widehat{A}_t^{(n)} = \sum_{j=0}^{n-1} \alpha^j \delta_{t+j}^{\varphi}. \tag{6.26}$$

The identity follows by expanding the TD errors: intermediate value terms cancel, leaving the n-step return $\sum_{j=0}^{n-1} \alpha^j g_{t+j} + \alpha^n \tilde{J}(x_{t+n})$ minus $\tilde{J}(x_t)$. It therefore interpolates between a one-step bootstrap and a long sampled return rather than introducing a new value definition. Generalized advantage estimation (GAE) forms the geometrically weighted sum

$$\widehat{A}_t^{\mathrm{GAE}(\lambda)} = \sum_{j=0}^{T-t-1} (\alpha\lambda)^j \delta_{t+j}^{\varphi}, \qquad 0 \le \lambda \le 1. \tag{6.27}$$

Small λ relies more strongly on the critic and usually has lower variance; large λ approaches a long return and reduces bootstrap bias at the cost of variance. GAE is therefore the policy-gradient counterpart of the TD(λ) continuum in Chapter 4 (Schulman, Moritz, et al. 2016).

6.4.2 Which distribution fits the critic?

Suppose the critic minimizes

$$\mathscr{L}_\nu(\varphi) = \mathbb{E}_{x\sim\nu}[(\tilde{J}(x;\varphi) - y(x))^2]. \tag{6.28}$$

A small loss is a statement weighted by ν. PPO makes ν close to the recent policy occupancy by recollecting trajectories. Replay methods use a mixture of older behavior distributions. Replay improves sample reuse and decorrelates minibatches, but the critic can be accurate on the buffer and poor at states newly preferred by the actor.

Let π_b denote the behavior policy that generated a transition and let π be the target policy being evaluated or improved. Provided $\pi_b(u \mid x) > 0$ whenever $\pi(u \mid x) > 0$, define the per-decision and trajectory ratios

$$\rho_t = \frac{\pi(u_t \mid x_t)}{\pi_b(u_t \mid x_t)}, \qquad \rho_{t:j} = \prod_{k=t}^{j} \rho_k. \tag{6.29}$$

The ratio is a change of measure: actions common under the target but rare under the behavior policy receive more weight. For an episodic return, ordinary trajectory importance sampling satisfies

$$\mathbb{E}_\pi[G_t \mid x_t] = \mathbb{E}_{\pi_b}[\rho_{t:T-1} G_t \mid x_t] \tag{6.30}$$

under the stated support and integrability conditions. Weighted importance sampling normalizes the observed weights; it is generally biased at finite sample size but often has lower variance. These are the off-policy Monte Carlo constructions developed in Sutton and Barto (2018, Ch. 5).

Bootstrapping changes which ratios are needed. A one-step semi-gradient off-policy TD update for a value critic can be written

$$\varphi^{+} = \varphi + \eta_{\varphi} \rho_t \delta_t^{\varphi} \nabla_{\varphi} \tilde{J}(x_t; \varphi). \tag{6.31}$$

The single action ratio corrects the one-step action distribution. A sampled stochastic-actor term has the analogous form

$$\hat{g}_{\theta} = \rho_t \widehat{A}_t \nabla_{\theta} \log \pi_{\theta}(u_t \mid x_t). \tag{6.32}$$

However, ρ_t alone does not change the behavior state occupancy d_{π_b} into d_{π}. Exact trajectory correction uses products such as $\rho_{0:t}$ or an appropriate state-density ratio; both can have severe variance and require coverage. Off-policy TD with function approximation and bootstrapping also has stability subtleties beyond the tabular identity.

Practical replay algorithms therefore combine short bootstrapped targets, target networks, restricted actor updates, and coverage heuristics rather than using unrestricted trajectory products. TD3 and SAC avoid an explicit importance ratio in their standard critic targets because they learn an off-policy Q-function from one-step transitions and evaluate the next action under the target actor; this is not permission to use arbitrary data without state–action coverage. PPO keeps ratios near one by collecting recent data and clipping the update. On-policy sampling is the special case $\pi_b = \pi$, for which every ratio equals one (Sutton and Barto 2018, Chs. 7 and 13).

Table 6.2: Three distributions that should not be conflated in actor–critic training.

Distribution	Role	Typical mismatch question
Behavior distribution	generates observed transitions	Were safety and exploration constraints respected during collection?
Critic-fitting distribution ν	weights evaluation error	Does replay cover states used by the current actor and online optimizer?
Deployment d_{π}	weights realized closed-loop performance	Did validation reproduce model shifts and rare states seen in use?

AI/RL terminology bridge

"On-policy" and "model-free" answer different questions. PPO is on-policy because its ratio objective uses trajectories from a recent policy. Its critic targets can be computed without an explicit transition model. TD3 and SAC are off-policy because replay data may come from older behavior policies. None of these labels says whether a simulator, learned model, or MPC safety layer is used elsewhere in the system.

6.5 PPO as Restricted On-Policy Improvement

PPO begins from a simple concern: an empirical policy-gradient direction is local, so changing the action probabilities too far with one noisy batch can invalidate the advantages and occupancy distribution estimated from that batch. PPO measures the probability change by a ratio and clips the *surrogate incentive* outside a neighborhood of one. The clipping does not clip the physical action, guarantee a KL bound, or enforce vehicle constraints; it restricts how much the sampled objective rewards a probability change.

The data and update flow is summarized visually in Figure C.2; the present section supplies the equations and the executable sequence in Algorithm 6.1.

Let a batch be collected by π_{old} and define

$$r_t(\theta) = \frac{\pi_\theta(u_t \mid x_t)}{\pi_{\text{old}}(u_t \mid x_t)}. \tag{6.33}$$

In this book's cost convention, a clipped PPO surrogate can be minimized as

$$\mathscr{L}^{\text{PPO}}_{\text{actor}}(\theta) = \widehat{\mathbb{E}}_t\big[\max\big(r_t(\theta)\widehat{A}^c_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A}^c_t\big)\big], \tag{6.34}$$

where $\widehat{A}^c_t$ is a cost advantage. The more familiar reward form maximizes a minimum; the reversal in Equation (6.34) follows from cost $= -$ reward. The hyperparameter $\epsilon \in (0, 1)$ is the clipping half-width, so the retained probability-ratio interval is $[1-\epsilon, 1+\epsilon]$. It is unrelated to exploration noise or the label error denoted by ε in Chapter 3. Implementations commonly negate physical costs and use the reward form instead.

The critic is fitted to returns or bootstrapped targets, for example

$$\mathscr{L}_{\text{critic}}(\varphi) = \widehat{\mathbb{E}}_t\left[(\tilde{J}(x_t;\varphi) - \widehat{G}_t)^2\right]. \tag{6.35}$$

Several minibatch epochs reuse the collected batch, after which new on-policy data are required. Entropy bonuses may preserve exploration. The clipping parameter restricts the incentive for a large probability-ratio change on the sampled data; it is not a hard trust region in state space and not a guarantee of monotonic closed-loop improvement (Schulman, Wolski, et al. 2017).

PPO is attractive when a stochastic policy is acceptable, simulation can supply fresh trajectories, and robustness to moderate optimizer choices is more important than replay efficiency. It is not automatically the right choice for an expensive physical platform: on-policy data are consumed quickly relative to off-policy replay methods.

Algorithm 6.1: Clipped PPO in cost form

Input: actor π_θ, value critic $\tilde{J}_\varphi$, discount α, GAE parameter λ, clip half-width ϵ, actor/critic stepsizes, minibatch size, and numbers of collection steps and optimization epochs.

1. Set $\theta_{\text{old}} \leftarrow \theta$. Collect a fresh batch with $\pi_{\theta_{\text{old}}}$ and store (x_t, u_t, g_t, x_{t+1}), termination flags, and old log probabilities.

2. Compute δ_t^{φ}, bootstrapped return targets $\widehat{G}_t$, and $\widehat{A}_t^{\mathrm{GAE}(\lambda)}$ backward along each trajectory.
3. For several shuffled-minibatch epochs, form $r_t(\theta)$ and descend the clipped actor loss (6.34); fit the critic with (6.35).
4. Optionally stop actor epochs early if an approximate KL threshold is exceeded. Record KL and clipping fraction as update diagnostics.
5. Report unnormalized physical cost and constraint metrics, then repeat from Step 1 with the updated actor.

Output: updated stochastic actor and value critic.

Advantage normalization is a numerical choice, not a change to the physical cost reported for the controller. The algorithm uses an on-policy *outer loop*: a batch may be reused for several epochs, but a materially updated actor requires fresh trajectories.

More epochs extract more optimization from a batch but also move the actor farther from the policy that generated it. Increasing ϵ permits larger ratio changes. Neither choice has a universal best value; both interact with stepsize, advantage scale, and sample count. A reproducible study should report these quantities and learning curves over multiple seeds.

6.6 DDPG and TD3: Deterministic Continuous-Action API

DDPG combines a deterministic actor, replay buffer, Q critic, and delayed target networks (Lillicrap et al. 2016). In cost notation, one critic target is

$$y_t^{\mathrm{DDPG}} = g_t + \alpha \widetilde{Q}(x_{t+1}, \mu_{\theta^-}(x_{t+1}); \varphi^-), \tag{6.36}$$

and the critic minimizes squared TD error. The actor minimizes

$$\mathscr{L}_{\mathrm{actor}}^{\mathrm{DDPG}}(\theta) = \widehat{\mathbb{E}}_{x\sim\mathscr{D}}[\widetilde{Q}(x, \mu_\theta(x); \varphi)]. \tag{6.37}$$

Exploration noise is added to behavior actions during data collection; the deployed actor can remain deterministic. The actor–twin-critic–target flow of TD3 is shown in Figure C.3; the distinctions among behavior noise, target noise, and delayed actor updates are made explicit below.

TD3 modifies this loop in three coupled ways (Fujimoto et al. 2018):

$$\xi \sim \mathrm{clip}(\mathcal{N}(0, \sigma_{\mathrm{tar}}^2 I), -c, c), \tag{6.38}$$

$$u' = \mathrm{clip}(\mu_{\theta^-}(x') + \xi, \mathscr{U}), \tag{6.39}$$

$$y_t^{\mathrm{TD3},c} = g_t + \alpha \max_{j\in\{1,2\}} \widetilde{Q}_j(x', u'; \varphi_j^-), \tag{6.40}$$

$$\theta^+ = \theta - \eta_\mu \nabla_\theta \widehat{\mathbb{E}}_x[\widetilde{Q}_1(x, \mu_\theta(x); \varphi_1)]. \tag{6.41}$$

Target-action noise smooths narrow critic artifacts, the conservative twin selection uses the smaller reward or equivalently larger cost estimate, and the actor/targets are updated less often

than the critics. These are controls on approximation error inside API. They do not prove that the critic is accurate outside replay support.

For mobility systems, the deterministic actor is attractive for bounded continuous commands and fast execution. The critic can also become a terminal value for online lookahead. That second use should be validated separately: a critic adequate for its actor-gradient samples may be inaccurate over the larger state region reached by an optimizer.

Here σ_{tar} is the target-smoothing standard deviation and $c > 0$ caps each noise component. This target noise is distinct from the behavior exploration noise used to collect data.

Algorithm 6.2: TD3 in cost form

Input: deterministic actor μ_θ, twin Q critics $\widetilde{Q}_{\varphi_1}, \widetilde{Q}_{\varphi_2}$, delayed copies $(\theta^-, \varphi_1^-, \varphi_2^-)$, replay buffer $\mathcal{D}$, discount α, target-noise parameters (σ_{tar}, c), policy delay d, Polyak coefficient $\kappa \in (0, 1)$, and stepsizes.

1. Apply $u_t = \text{clip}(\mu_\theta(x_t) + \epsilon_t^{\text{beh}}, \mathcal{U})$ and append (x_t, u_t, g_t, x_{t+1}) to $\mathcal{D}$.
2. Sample a replay minibatch. Draw clipped target noise ξ and form u' and the conservative cost target (6.40).
3. Fit both critics by descending $\widehat{\mathbb{E}}[(\widetilde{Q}_{\varphi_j}(x, u) - y^{\text{TD3},c})^2]$.
4. Every d critic updates, update the actor through (6.41), then set each delayed parameter $\omega^- \leftarrow \kappa\omega^- + (1 - \kappa)\omega$ for $\omega \in \{\theta, \varphi_1, \varphi_2\}$.

Output: deterministic actor and twin Q critics.

Exploration noise ϵ_t^{beh} changes data collection; target- smoothing noise ξ regularizes the critic target around the next actor action. Delayed actor updates describe training frequency, not the physical controller's sampling rate.

6.7 SAC as Entropy-Regularized Policy Iteration

SAC augments the physical cost with an entropy preference:

$$\mathcal{J}_{\text{soft}}(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \alpha^t (g(x_t, u_t) - \tau\,\mathcal{H}(\pi(\cdot \mid x_t)))\right]. \tag{6.42}$$

The coupled replay, twin-Q, stochastic-actor, and entropy flow appears in Figure C.4. The equations below explain why the entropy term must occur consistently in both evaluation and improvement. Here $\tau \geq 0$ is the temperature. The term $-\tau\mathcal{H}$ lowers the augmented cost of a broad policy, delaying premature concentration on one action and preserving alternative actions for off-policy learning. It is a preference in the optimization objective, not a claim that random control is physically robust. Setting $\tau = 0$ removes the entropy term, while automatic temperature tuning adjusts τ toward a declared target entropy. The soft state value associated with a cost Q-factor is

$$V_{\text{soft}}(x) = \mathbb{E}_{u\sim\pi_\theta}[Q_{\text{soft}}(x, u) + \tau \log \pi_\theta(u \mid x)], \tag{6.43}$$

because $\mathbb{E}[-\log \pi] = \mathscr{H}$. A twin-critic target is then formed from $g + \alpha V_{\text{soft}}(x')$. With cost critics, a conservative sampled target uses the larger of the twin cost estimates:

$$y_t^{\text{SAC},c} = g_t + \alpha \left[\max_{j \in \{1,2\}} \widetilde{Q}_{\varphi_j^-}(x_{t+1}, u') + \tau \log \pi_\theta(u' \mid x_{t+1}) \right], \quad u' \sim \pi_\theta(\cdot \mid x_{t+1}). \tag{6.44}$$

The stochastic actor minimizes

$$\mathscr{L}_{\text{actor}}^{\text{SAC}}(\theta) = \widehat{\mathbb{E}}_{\substack{x \sim \mathscr{D} \\ u \sim \pi_\theta}} [Q_{\text{soft}}(x, u) + \tau \log \pi_\theta(u \mid x)]. \tag{6.45}$$

Reparameterization permits gradients through sampled continuous actions. The temperature τ may be fixed or adapted toward a target entropy (Haarnoja et al. 2018).

In DP terms, SAC alternates approximate soft policy evaluation and soft policy improvement. As τ decreases, the entropy preference weakens, but the algorithm is not simply TD3 with noise. Its actor remains a distribution and its Bellman target contains the entropy term. For deployment, one must choose whether to sample or apply a deterministic statistic and then audit that choice. Entropy is not physical disturbance robustness, and random exploration is not a constraint-satisfaction mechanism.

Algorithm 6.3: SAC in cost form

Input: reparameterized stochastic actor π_θ, twin cost-Q critics and delayed copies, replay buffer $\mathscr{D}$, discount α, temperature τ (fixed or learned), target entropy if used, Polyak coefficient, and stepsizes.

1. Sample $u_t \sim \pi_\theta(\cdot \mid x_t)$, apply any declared safety layer, and append (x_t, u_t, g_t, x_{t+1}) to replay.
2. Sample a replay minibatch and reparameterized next actions. Construct the soft target (6.44) and fit both Q critics by squared TD error.
3. Draw reparameterized current-state actions and update the actor by descending (6.45).
4. If temperature is adaptive, update τ so the observed $-\log \pi_\theta(u \mid x)$ tracks the declared target entropy.
5. Polyak-average the delayed critics and repeat.

Output: stochastic actor, twin soft Q critics, and temperature.

The Q target, actor objective, and temperature update form one coupled design. Removing the entropy term only from the actor while retaining it in the critic changes the fixed point. Likewise, stochastic training return does not determine the performance of a mean-action deployment policy; both execution choices should be evaluated if either is contemplated.

6.8 Derivative-Free Search in Structured Policy Classes

Not every useful actor needs a policy gradient or a large neural network. Let μ_θ be a gain-scheduled, rule-based, or low-dimensional neural controller and define the simulation objective

$\mathcal{J}(\theta)$. This is *direct policy search*: complete closed-loop trials assign a cost to a parameter vector without first fitting a critic. It is useful when controller structure is trusted, a simulator or repeatable experiment is available, and the parameter dimension is modest. It does not mean that the system dynamics are absent; they are exercised inside each trial.

For directions Δ_i with $\mathbb{E}[\Delta_i \Delta_i^\top] = I$, a symmetric random-direction estimate is

$$\widehat{\nabla}\mathcal{J}(\theta) = \frac{1}{2cM} \sum_{i=1}^{M} (\widehat{\mathcal{J}}(\theta + c\Delta_i) - \widehat{\mathcal{J}}(\theta - c\Delta_i))\Delta_i. \tag{6.46}$$

The numerator measures the objective slope along Δ_i; multiplication by Δ_i maps that directional slope back into parameter coordinates. For a smooth objective, its conditional expectation approaches the gradient as $c \downarrow 0$, up to the direction-distribution convention. Very small c amplifies simulation noise, whereas large c produces finite-difference bias. Using the same initial states and disturbance seeds in each plus/minus pair—common random numbers—removes variation unrelated to the parameter perturbation. The estimate avoids differentiating the plant but needs two rollouts per direction and becomes noisy in a very large parameter vector.

The cross-entropy method (CEM) instead maintains a parameter distribution. For a Gaussian search law, one iteration is

$$\theta^{(j)} \sim \mathcal{N}(m_i, \Sigma_i), \qquad j = 1, \dots, M, \tag{6.47}$$

$$\mathcal{E}_i = \text{indices of the lowest-cost } M_e \text{ samples}, \tag{6.48}$$

$$m_{i+1} \leftarrow (1 - \beta) m_i + \beta \frac{1}{M_e} \sum_{j \in \mathcal{E}_i} \theta^{(j)}, \tag{6.49}$$

$$\Sigma_{i+1} \leftarrow (1 - \beta)\Sigma_i + \beta \operatorname{Cov}\{\theta^{(j)} : j \in \mathcal{E}_i\}. \tag{6.50}$$

The elite set is an empirical approximation to a low-cost region. Fitting the next Gaussian to those elites is the maximum-likelihood, or equivalently minimum-cross-entropy within the Gaussian family, update. Smoothing by $\beta \in (0, 1]$ prevents one noisy batch from replacing the search law, and a diagonal variance floor prevents irreversible collapse before the region has been audited (Boer et al. 2005).

Algorithm 6.4: CEM policy search

Input: policy class μ_θ, initial search mean m_0 and covariance Σ_0, scenario set, sample count M, elite count M_e, smoothing β, variance floor, and stopping rule.

1. Draw M parameter vectors from the current search law, enforcing any declared parameter bounds.
2. Evaluate every vector on the same training scenario protocol and rank the resulting closed-loop costs.
3. Select the M_e lowest-cost elites. Compute their sample mean and covariance and apply (6.49)– (6.50).
4. Apply covariance floors, record the best held-out candidate separately, and repeat until the evaluation budget or convergence rule is met.

Output: selected actor parameters and the search/audit history.

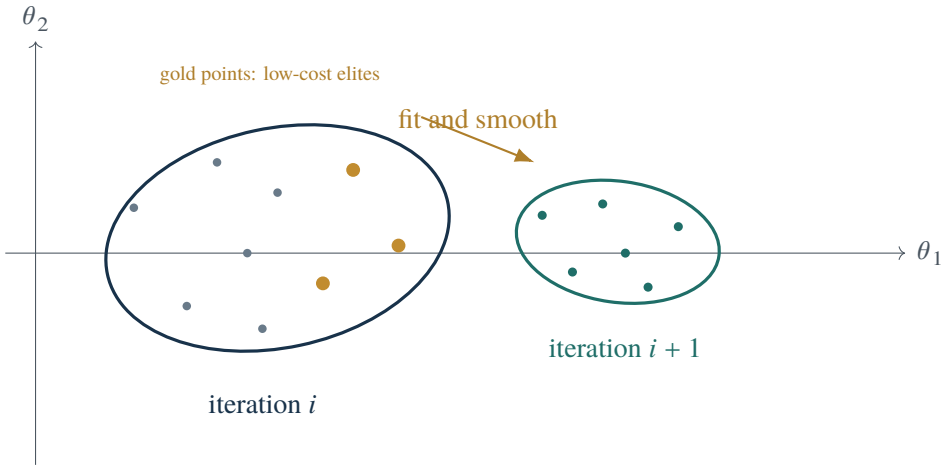


Figure 6.6: Original two-parameter CEM geometry. Samples are ranked by complete closed-loop cost; the next search distribution contracts toward the elite region while smoothing and variance floors control premature collapse.

CEM is transparent for a small structured controller and parallelizes across simulations. It does not require differentiability or a randomized deployed policy, but it is not guaranteed to find a global optimum. Its sample count typically grows difficult with parameter dimension, and the best training parameter can overfit the scenario set. Held-out disturbances and model variations are therefore required (Bertsekas 2025, Sec. 3.5).

6.9 One Comparison Map for Modern Algorithms

Table 6.3 compares methods by mathematical role rather than popularity. DQN is retained because it shows the boundary: its Q critic already defines a discrete actor, whereas the other methods learn an explicit policy. Concept diagrams for all four algorithms appear in Appendix C.

Three questions should accompany every algorithm name:

1. Which policy or optimality equation does the critic target?
2. How does the actor or greedy rule represent improvement?
3. Is the frozen result applied directly, or improved again online?

The third question is often absent from standard DRL comparisons and is the main connection to the control architecture of this book.

Table 6.3: Algorithm comparison through the book's DP and deployment axes.

Method	Action/policy	Critic target	Data relation	DP reading	Natural online use
DQN	finite, implicit greedy	optimality Q TD	off-policy replay	sampled approximate Q-VI	Q as leaf evaluator for discrete search
PPO	stochastic explicit	recent-policy value/advantage	on-policy batches	restricted API	actor base/warm start; critic terminal value
TD3	deterministic continuous	twin policy-Q TD	off-policy replay	deterministic API with bias controls	fast base actor and differentiable terminal Q
SAC	stochastic continuous	soft twin-Q TD	off-policy replay	entropy-regularized PI	sample/mean base policy plus soft critic
CEM	chosen structured class	no critic required	simulated or experimental policy trials	direct parameter optimization	optimized actor as local-search center

6.10 The Learned Actor Is a Starting Point for Online Improvement

Let μ_θ be a frozen actor and let $\tilde{J}_\varphi$ be a critic. Before listing complete execution architectures, distinguish two surrounding cases from the book's three core online-improvement mechanisms. Applying the actor directly is the no-improvement baseline. A safety filter is a protective wrapper that checks or modifies a proposed action; it can surround a direct actor or any improved controller and is not counted as a fourth policy- improvement mechanism. The three mechanisms studied below are: use the actor as a base policy for rollout, use it to center or initialize online optimization, and use the critic to compress the cost beyond the online horizon. They are building blocks and may be combined rather than mutually exclusive algorithms.

6.10.1 Actor as a base policy for rollout

For a sampled or known model, evaluate a candidate root action and then follow the actor for m steps:

$$\widehat{Q}_{\mathrm{roll}}(x,u) = g(x,u) + \mathbb{E}\left[\sum_{j=1}^{m} \alpha^j g(x_j, \mu_\theta(x_j)) + \alpha^{m+1}\tilde{J}_\varphi(x_{m+1})\right]. \tag{6.51}$$

The applied action minimizes this estimate. When the base-policy cost is evaluated exactly and the candidate set includes the base action, classical one-step rollout is no worse than the base policy. Finite Monte Carlo samples, terminal approximation, action discretization, and model mismatch weaken that statement; improvement then requires an audit or an uncertainty margin.

6.10.2 Actor as a warm start or local search center

For a continuous online optimizer, the actor supplies

$$u_{0:N-1}^{\mathrm{init}} = (\mu_\theta(x), \mu_\theta(\hat{x}_1), \ldots). \tag{6.52}$$

An SQP, shooting, sampling, or MPC solver then updates the sequence subject to the current model and constraints. A lower-cost initialization can reduce iterations, but a nonconvex solver may still converge to a different local solution. Warm-start benefit should be measured by computation and achieved cost at the allowed deadline.

If full optimization is too expensive, search a local action set

$$\mathcal{U}_{\mathrm{local}}(x) = \mathrm{clip}\{\mu_\theta(x) + \Delta u_j : j = 1, \ldots, M\}. \tag{6.53}$$

The actor centers a higher-resolution comparison near its preferred action while preserving a fixed real-time budget. A fallback global candidate or the actor action itself should be retained so that local search does not exclude a known feasible command.

6.10.3 Critic as a terminal value

An ℓ-step online problem uses

$$\min_{u_0,\dots,u_{\ell-1}} \mathbb{E}\left[\sum_{j=0}^{\ell-1} \alpha^j g(x_j, u_j) + \alpha^\ell \tilde{J}_\varphi(x_\ell)\right]. \tag{6.54}$$

The critic compresses the tail beyond the online horizon. In operator notation, a useful family is

$$T^\ell T^m_{\mu_\theta} \tilde{J}_\varphi, \tag{6.55}$$

where m actor-rollout steps refine the leaf and ℓ unrestricted lookahead steps improve the root. This is the policy-space counterpart of the geometric construction in Chapter 5. Increasing either depth spends more online computation but reduces reliance on a raw approximation in a different way.

6.10.4 From three mechanisms to five deployment patterns

The three mechanisms above describe how learned objects participate in online improvement. A deployed controller still needs a complete execution pattern: what proposes an action, what computation is permitted before the deadline, and what claim is made about the resulting command. The following five patterns retain that operational distinction.

Direct actor. The minimum-computation baseline applies

$$u = \mu_\theta(x), \tag{6.56}$$

or samples from $\pi_\theta(\cdot \mid x)$ for a stochastic policy. No model or online optimization is required after the actor has been trained. This is the natural use of a policy network when latency is the dominant constraint, but it also places the full burden on offline data coverage. Saturation may keep the command inside an actuator interval; it does not by itself establish constraint satisfaction, robustness, or low cost after a distribution shift. Those properties must be audited on the deployed state and disturbance range.

Actor with a safety filter. A filter treats the actor command as a proposal and returns the nearest command that passes a declared safety test. A representative one-step abstraction is

$$u = \operatorname*{arg\,min}_{v \in \mathcal{U}_{\text{safe}}(x)} \|v - \mu_\theta(x)\|_R^2, \tag{6.57}$$

where $\mathcal{U}_{\text{safe}}(x)$ may be constructed by projection, a quadratic program, a control-barrier condition, or a predictive backup test. The objective is to modify the learned action as little as possible while enforcing the filter's certificate; it is not, in general, a Bellman cost improvement. The relevant questions are whether the filter problem remains feasible, which model and uncertainty set support its claim, and how often its corrections move the system outside the actor's training distribution. The same wrapper can surround a direct actor, local search, rollout, or MPC.

Actor-warm-started MPC. Here the actor does not determine the final command. It produces the initial sequence in Equation (6.52), after which a finite- horizon optimizer incorporates the current reference, model, constraints, and measured state. Only the first optimized action is applied. Warm starting is valuable when it reduces iterations or supplies a useful feasible candidate within a fixed deadline; the comparison should therefore report solve time, deadline success, achieved cost, and constraint behavior against a cold start or another initialization. A certified fallback remains necessary when the solver terminates early or fails to return a feasible sequence.

Local actor search. When a full nonlinear program is too expensive, the actor can center the finite set in Equation (6.53). One-step search with a critic, for example, applies

$$u \in \operatorname*{arg\,min}_{v \in \mathcal{U}_{\text{local}}(x)} \left\{ g(x, v) + \alpha \, \mathbb{E}\left[\tilde{J}_{\varphi}(x^{+}) \mid x, v \right] \right\}. \tag{6.58}$$

This preserves a declared number of model or sample evaluations and gives finer resolution near the learned action. It is cheaper and more predictable than open-ended optimization, but it can miss an action outside the local neighborhood. Retaining the actor command, one or more coarse global candidates, and a feasible fallback makes that coverage limitation explicit.

Actor rollout with a critic leaf. This pattern uses the actor as the continuation policy and the critic as a compressed tail, as in Equation (6.51). Candidate root actions may come from the full finite action set, a sampled continuous set, or the actor-centered neighborhood. The online model or simulator propagates each candidate, the actor supplies intermediate actions, and the critic values the leaf. Increasing the rollout or lookahead depth reduces dependence on a raw leaf estimate but adds model, sampling, and computation error. Thus the claim is conditional on candidate coverage, model fidelity, sample count, terminal-value quality, and completion before the control deadline. A safety filter can still be applied to the selected root action.

The five rows of Table 6.4 expand these three mechanisms into deployable patterns. Warm-started MPC and local action search are two implementations of actor-centered optimization. Actor rollout with a critic leaf combines the first and third mechanisms. The direct actor is the baseline, and the safety-filter row is a wrapper that may be composed with any other row. This table is an authorial synthesis, not a standard five-category taxonomy quoted from one source. Its rollout and terminal-value interpretation follows Bertsekas (2019) and Bertsekas (2022); its MPC execution layer follows the finite-horizon framework of Rawlings et al. (2017); and its safety-filter row represents architectures such as Wabersich and Zeilinger (2021). The fixed-budget local-search row is the chapter's finite-candidate specialization of actor-centered online improvement.

Table 6.4: Execution patterns built from the same learned actor and critic. The rows synthesize the cited rollout, MPC, and safety-filter constructions with the chapter's actor-centered local-search pattern.

Pattern	Online computation	Main benefit	Claim that still needs verification
Direct actor	one network evaluation	minimum latency	shifted-model cost and constraint behavior
Actor + safety filter	actor followed by projection, QP, or filter	explicit local action modification	feasibility and interaction with the learned policy
Actor warm-started MPC	finite-horizon constrained optimization	current constraints and model; fewer solver iterations	deadline success and local-solver quality
Local actor search	finite declared candidate set	predictable compute budget and local resolution	candidate coverage outside the actor neighborhood
Actor rollout + critic leaf	sampled/model tree with terminal value	online improvement with a compressed tail	sampling, model, and terminal-value errors

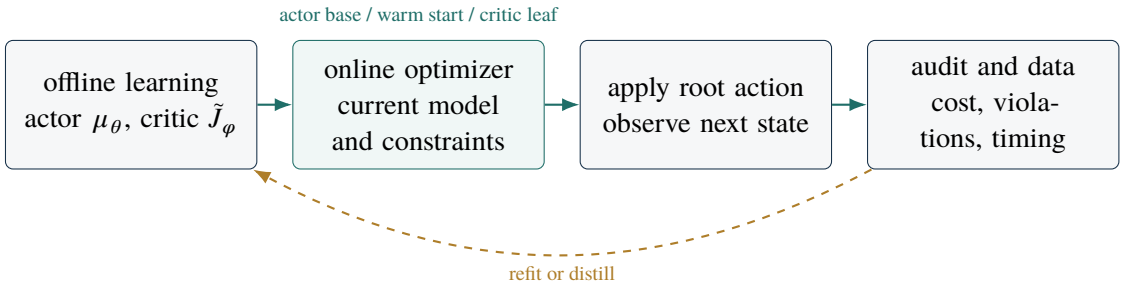


Figure 6.7: Original actor–critic–lookahead architecture. Learning, online optimization, and deployment audit are separate layers with separate claims.

Control viewpoint

The phrase "combining an actor with online lookahead and a critic with terminal cost" is the main bridge from modern DRL to this textbook, but it is not one specific algorithm. PPO, TD3, SAC, supervised MPC imitation, or CEM can supply the actor. MC, TD, fitted value iteration, or the same actor–critic training can supply the terminal value. The online layer determines how the stored objects are used at execution.

6.11 Mobility Case: A Policy Actor with Online Lookahead

The reproducible experiment isolates the architecture without requiring a deep learning toolbox. Consider the normalized lateral-error surrogate

$$x_{k+1} = 1.03x_k + bu_k + d + w_k, \qquad |u_k| \le 1.5, \tag{6.59}$$

with stage cost

$$g(x,u) = x^2 + 0.08u^2 + 0.01u^4, \qquad \alpha = 0.97. \tag{6.60}$$

The coefficient b represents control effectiveness, while d is a constant lateral bias that can stand for a normalized crosswind or road-curvature feedforward term. Offline training uses $(b,d) = (1,0)$. The online audit fixes $b = 0.75$ and varies the previously unseen bias over $d \in [0, 0.22]$. The lookahead model receives the current (b,d), while the frozen actor and critic are not refitted. The remaining disturbance is bounded Gaussian noise generated from fixed seeds. This scalar model is a teaching surrogate, not a validated vehicle model.

6.11.1 Offline actor search and critic fitting

The structured deterministic actor is

$$\mu_\theta(x) = 1.5 \tanh\left(-\frac{\theta_1 x + \theta_3 x^3}{1.5}\right). \tag{6.61}$$

CEM optimizes its two parameters over common initial states and disturbance sequences. The verified result is

$$\theta_1 = 0.8615, \qquad \theta_3 = 0.3400. \tag{6.62}$$

The actor's Monte Carlo cost is then fitted by the even polynomial

$$\tilde{J}_\varphi(x) = \varphi_0 + \varphi_2 x^2 + \varphi_4 x^4 + \varphi_6 x^6. \tag{6.63}$$

Its RMSE on the declared critic grid is 0.0225. This low number is a fitting diagnostic, not the online-control conclusion.

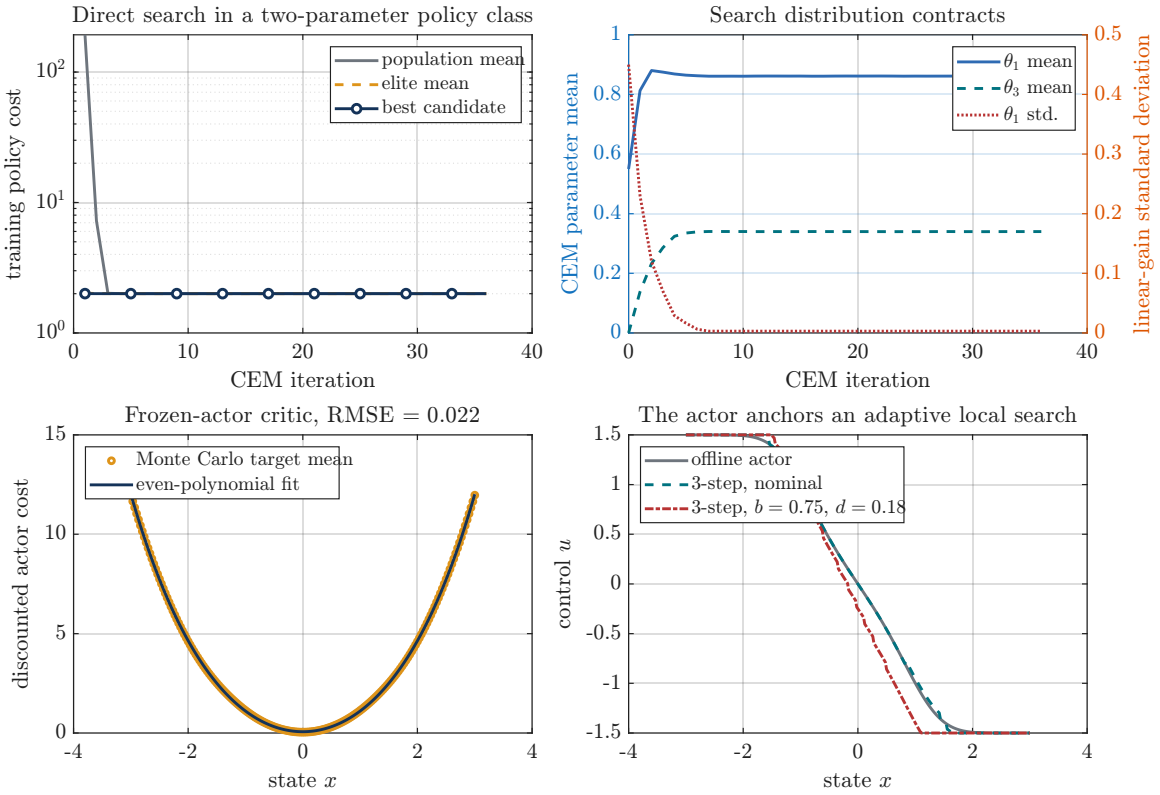


Figure 6.8: Original policy-space experiment. CEM contracts a distribution over two actor parameters, after which Monte Carlo actor costs are compressed into an even-polynomial critic. The lower-right panel shows that online three-step search changes the control law when effectiveness and the constant lateral bias differ from their offline values.

6.11.2 Actor-centered lookahead

At every state, local search compares 29 clipped candidates

$$u \in \text{clip}\{\mu_\theta(x) + \Delta u : \Delta u \in [-0.70, 0.70]\}. \tag{6.64}$$

One- and three-step backward recursions use the current values of (b, d) in the online model and terminate with Equation (6.63). A separate 61-action full-grid calculation audits whether the actor-centered candidate set misses an important action. The figure precomputes these policies on a dense state grid for efficiency; operationally, the same finite search would be performed only at the measured state.

At $(b, d) = (0.75, 0.18)$, the offline actor has mean discounted cost 4.5449. One-step local lookahead reduces it to 2.3430, and three-step lookahead gives 2.3175. The 61-action three-step audit gives 2.3159. The full-grid and local results are close, which supports the declared actor-centered candidate coverage in this scalar case. It does not prove coverage in a higher-dimensional action space.

Table 6.5: Verified shifted-model result at $(b, d) = (0.75, 0.18)$.

Controller	Candidate actions/state	Mean discounted cost
Offline actor	one forward output	4.5449
Local one-step lookahead	29	2.3430
Local three-step lookahead	29 per backup	2.3175
Full-grid three-step audit	61 per backup	2.3159

The larger separation has a specific cause. The offline odd-feedback actor was trained with $d = 0$, so it reacts only after a constant bias creates state error. The online model sees d and can apply anticipatory compensation while using the actor as its search center. This is a cleaner demonstration of online information use than deliberately weakening the offline actor. It is not a robustness theorem: the lookahead is advantaged by knowing the audit parameters exactly. In a real vehicle, estimation error in (b, d), state constraints, solver deadlines, and out-of-range bias would have to be included before making a deployment claim.

6.11.3 What is model based and what is sample based?

CEM uses sampled trajectories from a nominal simulator and does not differentiate its equations. The critic targets are Monte Carlo samples. The online lookahead is explicitly model based because it evaluates successor states with the current (b, d). Finally, the reported costs use a held-out audit simulation. Calling the entire pipeline either "model free" or "model based" would hide these different roles.

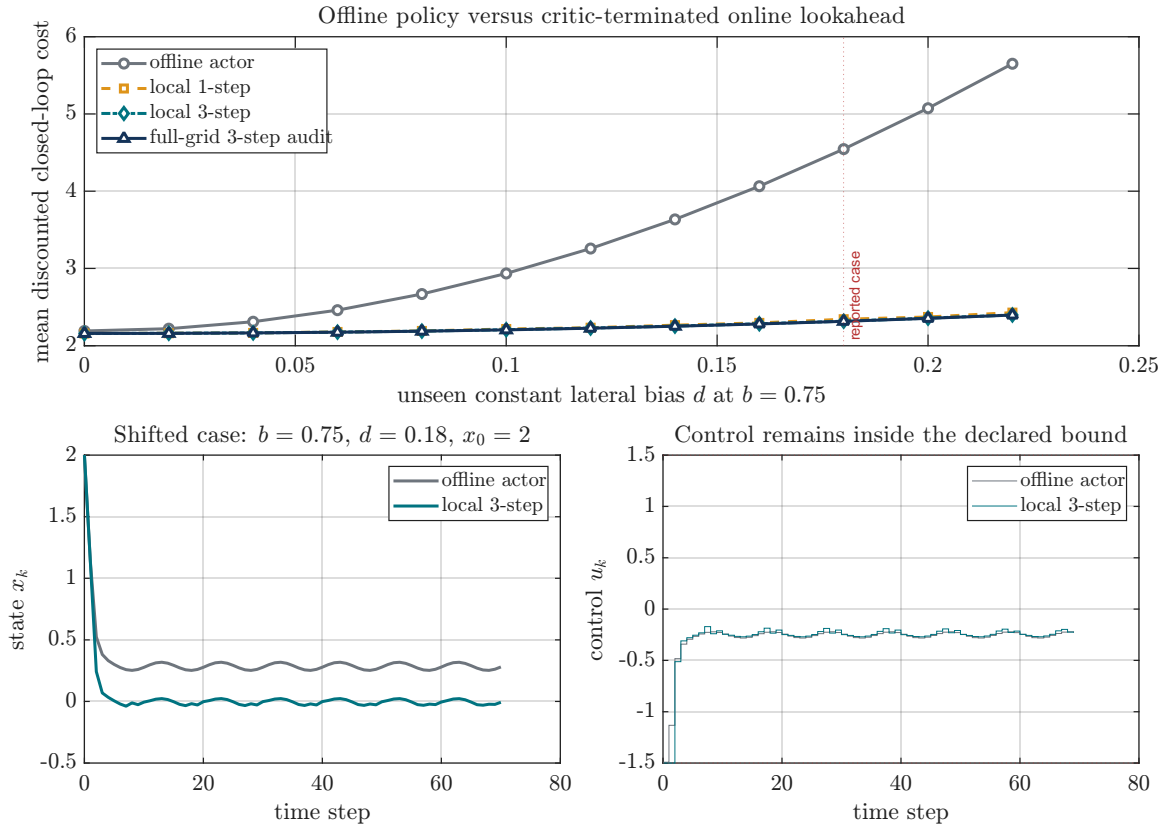


Figure 6.9: Original online-improvement audit. The offline actor is fixed while the lookahead layer uses the current effectiveness $b = 0.75$ and bias d. Local one- and three-step searches reduce audited cost throughout the tested interval. The full-grid curve is a candidate-set audit, not an exact infinite-horizon optimum. The lower panels show one shifted trajectory and bounded controls.

Table 6.6: Model and sample roles in the Chapter 6 experiment and representative policy algorithms.

Component	What the update consumes	Model/sample interpretation
Current CEM actor	complete nominal-simulator costs	simulation based and derivative free; no transition derivatives or critic, but the samples come from a model
PPO alternative	recent trajectories and returns/advantages	normally called model free at the update; on-policy samples may still be generated by a simulator
TD3/SAC alternative	replay transitions and bootstrapped Q targets	off-policy and sample based at the update; a simulator or physical system may supply replay
Fitted critic here	Monte Carlo actor returns	sample-based policy evaluation on nominal-model rollouts
Online lookahead	explicit successor calculation with current (b, d)	model based at execution
Held-out audit	fixed-seed shifted-model trajectories	simulation-based validation, not a training update

The classification belongs to a *component and use*, not merely to an algorithm name. CEM can also be run on physical experiments and then be model-free with respect to the search update; PPO can be embedded in a model-based data-generation pipeline; and a learned dynamics model can supply replay for TD3 or SAC. The online layer in this example is unambiguously model based because it queries a declared transition equation before choosing the root action.

Implementation note

The base-MATLAB script `generate_figures.m` in the Chapter 6 code directory records seeds, policy-search settings, candidate grids, and all model parameters. It writes both figures, CSV metrics, and a numerical summary. Students can replace the two-parameter actor with a neural policy without changing the online-lookahead interface.

6.12 Design and Deployment Checklist

Before choosing or deploying a policy-space method, document the following.

1. **Control object.** Is execution deterministic, stochastic, or a filtered statistic of a stochastic actor?
2. **Action structure.** Which actuator constraints are enforced by the architecture, and which require an external optimizer or filter?

3. **Evaluation target.** Does the critic evaluate the current actor, a delayed target actor, a soft policy, or Bellman optimality?
4. **Data relation.** Are samples on-policy, replayed off-policy, or produced by a declared model? Where is coverage measured?
5. **Improvement restriction.** Is the update clipped, delayed, entropy regularized, constrained to a parameter class, or selected by elite search?
6. **Online layer.** Is the actor applied directly, used as a base policy, or used to initialize constrained lookahead?
7. **Fallback.** What happens when the optimizer misses its deadline, the observation is out of distribution, or the critic is unreliable?
8. **Audit.** Report closed-loop cost, constraints, worst cases, computation, and sensitivity to seeds/model shifts. Training reward alone is insufficient.

6.13 Chapter Summary

Policy-space approximation stores a control law directly. It is especially useful for continuous actions and low-latency execution, but it does not remove policy evaluation. Policy-gradient methods use Q-factors or advantages to differentiate expected cost; actor–critic methods implement interleaved approximate policy evaluation and restricted improvement.

PPO is on-policy stochastic API with a clipped ratio surrogate. TD3 is off-policy deterministic API with twin critics, delayed actor updates, and target smoothing. SAC performs entropy-regularized soft policy iteration. Derivative-free methods such as CEM remain valuable for small structured controllers. These methods differ in data use and policy representation, not in the need for controller-level validation.

The chapter's main control connection is that a learned actor and critic are inputs to an execution architecture. The actor can be a base policy, warm start, local search center, and fallback; the critic can terminate online lookahead. The mobility experiment shows how this combination adapts a frozen offline policy to changed control effectiveness while keeping model use, sampling, and audit roles explicit.

6.14 Exercises

Exercise 6.1 (Deterministic action bounds)**.** Derive the Jacobian of Equation (6.4) with respect to the unsquashed output. Explain why gradients can become small near an actuator limit and why this is different from satisfying a state constraint.

Exercise 6.2 (Baseline identity)**.** Prove Equation (6.15) for a finite action set. State the regularity condition needed for a continuous action density.

Exercise 6.3 (Cost and reward signs). Starting from the reward-form PPO clipped objective, substitute $A^r = -A^c$ and derive Equation (6.34). Identify two places where a silent sign error could enter an implementation.

Exercise 6.4 (GAE recursion). Show that the truncated GAE sequence satisfies $\widehat{A}_t = \delta_t + \alpha\lambda\widehat{A}_{t+1}$. Compare the limits $\lambda = 0$ and $\lambda = 1$ and relate them to Chapter 4.

Exercise 6.5 (Twin critics under a cost convention). Explain why TD3's smaller-reward target becomes a larger-cost target. Construct a numerical example with two biased critics and compare the selected target.

Exercise 6.6 (SAC temperature). For two discrete actions with soft costs $Q_1 < Q_2$, derive the Gibbs policy that minimizes expected Q plus $\tau \log \pi$. Plot its probabilities as τ decreases and explain why temperature is not a robustness certificate.

Exercise 6.7 (CEM reproduction). Run the Chapter 6 script and reproduce the parameter trajectory. Change the elite fraction and variance floor. Report both training and held-out policy cost rather than only the best sampled candidate.

Exercise 6.8 (Actor architecture). Replace the cubic actor in Equation (6.61) by a linear saturated actor. Compare action RMSE, nominal policy cost, shifted cost, and the benefit of three-step lookahead.

Exercise 6.9 (Critic error versus online control). Fit quadratic, quartic, and sixth-order even critics to the same Monte Carlo targets. Compare critic RMSE with the root actions and closed-loop costs produced by one- and three-step lookahead.

Exercise 6.10 (Local candidate coverage). Vary the offset range and number of candidates in Equation (6.64). Plot computation against cost and identify a case in which actor-centered search excludes a useful action.

Exercise 6.11 (Rollout improvement). For a finite MDP and an exactly evaluated base policy, prove that one-step rollout with a candidate set containing the base action is no worse than the base policy. List which steps of the proof fail with sampled rollout and an approximate terminal critic.

Exercise 6.12 (Algorithm selection for a slalom project)**.** Design two pipelines for the course slalom problem: one using PPO and one using TD3 or SAC. For each, specify observation, action, critic target, exploration, constraints, online lookahead, fallback, and the model/data used in the final audit. Justify the comparison metrics before running either algorithm.

7 Learning Dynamics and Uncertainty

Planned role

This chapter addresses the second central difficulty of the book: the dynamics or disturbance model is uncertain, changing, or learned from data.

Planned contents

- Offline and online system identification
- Full-model, residual, black-box, and gray-box learning
- Deterministic and stochastic predictive models
- Recursive estimation and neural dynamics
- Aleatoric and epistemic uncertainty
- Excitation, closed-loop bias, and distribution shift
- Control-relevant model validation

8 Control with Learned Models

Planned role

This chapter closes the loop between model learning and decision making. It places modern model-based RL and learning-based MPC inside the book's DP framework.

Planned contents

- Explicit learned models for planning and policy training
- Dyna-style updates, synthetic rollouts, and model bias
- Short learned-model rollouts as in model-based policy optimization
- Residual-model and uncertainty-aware MPC
- PPO, TD3, and SAC actors as warm starts or base policies
- Learned critics as terminal values for online lookahead
- When online planning improves—or degrades—a frozen DRL policy
- Slalom and surrounding-driver response examples

9 Stability, Safety, and Online Adaptation

Planned role

This chapter distinguishes empirical safety from guarantees and studies what must be preserved when values, policies, or models are updated online.

Planned contents

- Lyapunov stability and performance certificates
- Recursive feasibility, invariant sets, and backup policies
- Robust and stochastic MPC
- Safety filters, shields, and control barrier functions
- Safety layers and backup controllers around PPO, TD3, and SAC policies
- Safe exploration and online model updates
- Adaptive dynamic programming and neuro-adaptive control

10 Multiagent Decision and Coordination

Planned role

This chapter extends online improvement to structured joint decisions and uses connected-vehicle coordination as the principal mobility setting.

Planned contents

- Joint-action growth and structured decision order
- Standard rollout and one-agent-at-a-time rollout
- Ordering, reshuffling, and parallel evaluation
- Centralized planning and decentralized information
- Centralized training with decentralized execution, MAPPO, and multi-agent SAC
- Merge, intersection, and roundabout coordination
- Learning surrounding-driver response models

11 Integrated Mobility Case Studies and Outlook

Planned role

This chapter assembles the book's methods into complete experimental pipelines and separates durable control principles from rapidly changing research topics.

Planned contents

- Staged slalom benchmark: tracking, minimum time, learned residuals, and safety
- CAV C0–C2: two-vehicle control, multiagent rollout, and mixed autonomy
- HEV energy management and eco-driving bridge examples
- Experimental design, ablation, and reproducibility
- World models, foundation models, VLMs, and LLMs as an outlook

A Mathematical Background

Planned role

This appendix will provide concise reminders rather than replace prerequisite courses.

Planned contents

- Matrix definiteness and quadratic forms
- Conditional expectation and Markov models
- Convex optimization and KKT conditions
- Fixed points, contractions, and Newton's method

B Additional Finite-Horizon Approximate DP Methods

Planned role

This appendix will collect finite-horizon approximation variants that would interrupt the main infinite-horizon control narrative.

Planned contents

- Stage-dependent approximation architectures
- Finite-horizon rollout and truncated rollout
- Scenario and sampling approximations
- Additional error bounds

C Modern RL Algorithm Bridge and Extended Models

This appendix supplies a compact entry point for readers who have not taken a separate deep-reinforcement-learning course. It is not a replacement for the derivations in Chapters 4 and 6. Its purpose is to define the algorithm names used there and place each one in the book's policy-evaluation–policy- improvement framework.

Chapter 1 will ultimately provide a one-page conceptual orientation: why learning is introduced and where value, policy, and model approximation enter a controller. Detailed algorithm cards belong here because placing them in Chapter 1 would interrupt that motivation before the control problem has been defined.

C.1 A Common Actor–Critic Template

An *actor* is a parameterized policy $\pi_\theta(u \mid x)$ or a deterministic map $u = \mu_\theta(x)$. A *critic* approximates J_π, Q_π, or a related advantage. A modern actor–critic iteration usually contains four logically separate operations:

1. a behavior policy collects transitions;
2. a critic target approximately evaluates a policy;
3. an actor objective approximately improves the policy within its parameter class; and
4. delayed targets, replay, clipping, or trust-region devices regulate the numerical update.

Safety filters, MPC, and online lookahead are additional control layers. They are not implied by the word actor–critic.

Most AI references maximize reward with symbols (s, a, r, π, γ). This book minimizes cost with (x, u, g, μ, α). The conversion is $r = -g$, $\gamma = \alpha$, $V^\pi = -J_\mu$, and $Q^\pi_{\text{reward}} = -Q_{\mu,\text{cost}}$. Signs in a software library must therefore be checked before comparing its "return" with the physical cost reported in this book.

C.2 DQN: A Critic That Also Defines a Discrete Actor

DQN stores a neural Q-factor and executes

$$\mu_\theta(x) \in \arg\min_{u \in \mathscr{U}} \widetilde{Q}(x, u; \theta). \tag{C.1}$$

Table C.1: A first algorithm map for readers entering from control.

Method	Actor/action	Critic target	Data relation	DP interpretation
DQN	implicit finite-action $\arg\min Q$	sampled Q-optimality TD target	off-policy replay	approximate Q-value iteration
PPO	stochastic actor	usually J_π plus an advantage estimate	on-policy batches	restricted approximate policy improvement
TD3	deterministic continuous actor	two Q critics with delayed targets	off-policy replay	deterministic approximate PI with bias controls
SAC	stochastic continuous actor	entropy-regularized Q critics	off-policy replay	soft policy iteration

Its replay target is a sampled optimality TD target evaluated with delayed parameters. Therefore the critic also defines an implicit actor when the action set is finite. Replay makes the update off-policy relative to the mixture of behavior policies stored in the buffer. Target networks, Double DQN, and dueling outputs address moving labels, selection bias, and representation sharing; they do not change the underlying finite-action Bellman objective. Chapter 4 provides the equations and limitations (Mnih et al. 2015; Hasselt et al. 2016; Wang et al. 2016).

DQN is a natural reference for discrete maneuver selection. It does not emit an arbitrary continuous steering or torque command unless that command has been discretized or represented by a finite primitive library.

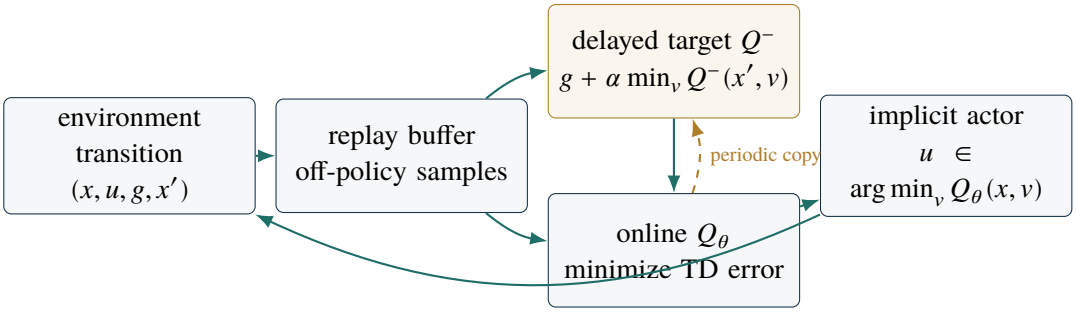


Figure C.1: Original concept diagram for DQN. The Q critic is trained by an off-policy optimality TD target and also supplies the finite-action actor.

C.3 PPO: On-Policy Improvement with a Restricted Ratio Step

PPO uses a stochastic actor and fresh trajectories from a recent policy π_{old}. Define the likelihood ratio

$$r_t(\theta) = \frac{\pi_\theta(u_t \mid x_t)}{\pi_{\text{old}}(u_t \mid x_t)}. \tag{C.2}$$

In the conventional reward-maximizing notation, the clipped surrogate is

$$L^{\text{clip}}(\theta) = \widehat{\mathbb{E}}_t\big[\min\big(r_t(\theta)\widehat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\widehat{A}_t\big)\big]. \tag{C.3}$$

The actor maximizes this expression. The advantage estimate $\widehat{A}_t$ is usually constructed from returns and a value critic. The ratio asks how much the new actor changes the probability of an action that was sampled by the old actor; clipping limits the incentive for a very large single update. It is a numerical policy-update device, not a hard bound on closed-loop state deviation or constraint violation (Schulman, Wolski, et al. 2017).

From the book's viewpoint, PPO is approximate PI with an on-policy critic and a restricted stochastic policy-space improvement. The critic answers the first organizing question in Section 4.10: it is intended to evaluate the recent behavior/target policy used to construct the batch, not J^* directly. For a cost-minimizing implementation one may negate costs to use the standard reward objective, provided that the conversion is documented.

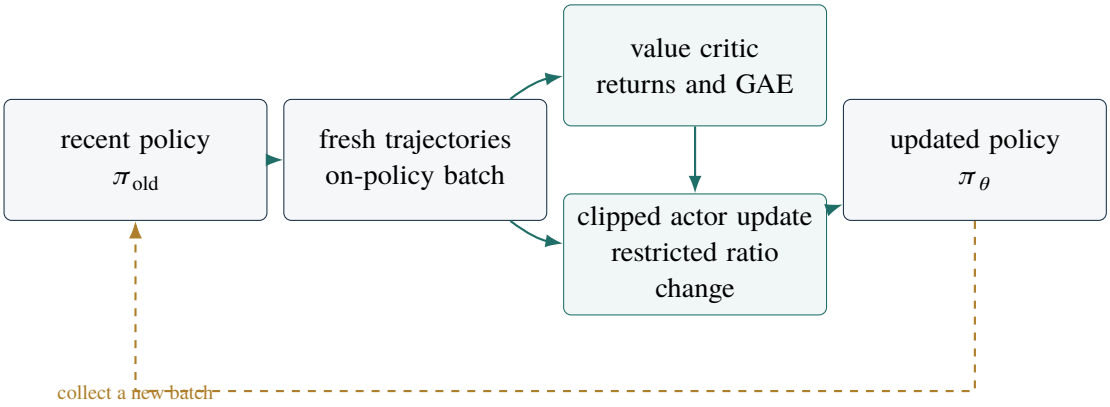


Figure C.2: Original concept diagram for PPO. Evaluation and restricted policy improvement reuse a recent on-policy batch; the next iteration recollects data.

C.4 TD3: Deterministic Continuous-Action API

TD3 uses a deterministic actor $u = \mu_\theta(x)$ and two Q critics. In the usual reward convention, its target uses the smaller of two delayed target critics, adds small clipped noise to the target action, and updates the actor less often than the critics. Under a literal cost conversion, the conservative

twin-critic selection reverses from a minimum reward to a maximum cost. Schematically,

$$u' = \mathrm{clip}(\mu_{\theta^-}(x') + \xi, \mathcal{U}), \tag{C.4}$$

$$y_{\mathrm{cost}}^{\mathrm{TD3}} = g + \alpha \max_{j\in\{1,2\}} \widetilde{Q}_j(x', u'; \varphi_j^-), \tag{C.5}$$

$$\theta \leftarrow \theta - \eta_\mu \nabla_\theta \widehat{\mathbb{E}}_x\big[\widetilde{Q}_1(x, \mu_\theta(x); \varphi_1)\big]. \tag{C.6}$$

The target-action noise smooths sharp critic errors over nearby controls; the delayed actor update allows additional critic fitting between improvements. These devices reduce specific approximation pathologies but do not prove that the learned deterministic actor is safe or globally optimal (Fujimoto et al. 2018).

In DP language, TD3 is off-policy approximate policy iteration in both value and policy space. The Q critics support continuous-action comparison, while the actor amortizes the inner minimization into one forward pass.

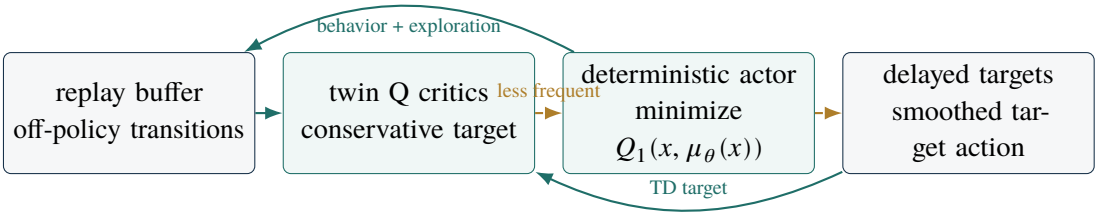


Figure C.3: Original concept diagram for TD3. Twin critics are fitted more often than the deterministic actor; delayed targets and target-action smoothing regulate the evaluation–improvement loop.

C.5 SAC: Entropy-Regularized Soft Policy Iteration

SAC learns a stochastic continuous actor and Q critics while adding an entropy preference. A cost-form soft objective is

$$J_{\mathrm{soft}}(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \alpha^t (g(x_t, u_t) - \tau \mathcal{H}(\pi(\cdot \mid x_t)))\right], \tag{C.7}$$

where $\tau \geq 0$ is the temperature. Entropy rewards a broad action distribution during learning, while the cost term preserves the physical objective. SAC alternates a soft Q-evaluation target and a policy update that makes the actor favor low soft-Q actions without collapsing its entropy too quickly (Haarnoja et al. 2018).

The temperature is not process-noise variance and does not by itself quantify robustness. Likewise, stochastic exploration is not a safety mechanism. For a mobility controller, execution must specify whether the mean, mode, or a sample of the policy is applied and how actuator and state constraints are enforced.

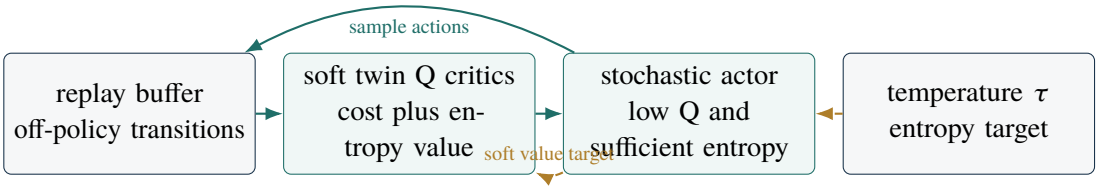


Figure C.4: Original concept diagram for SAC. Soft policy evaluation and stochastic policy improvement share replay data; temperature controls the entropy term, not physical robustness or safety.

C.6 Reading the Algorithms Through Three Questions

For any named RL algorithm, ask:

1. **Which policy does the critic evaluate?** Fixed-policy MC and TD answer this in Section 4.3; API and the cards above identify how that policy changes. Q-learning instead targets optimality directly.
2. **How is improvement represented?** DQN takes a finite arg min; PPO, TD3, and SAC fit explicit actors. Chapter 6 develops policy-space approximation.
3. **Is another improvement performed online?** A frozen actor can be a base policy or warm start, and a frozen critic can be a terminal value. Chapters 4, 5, and 8 develop rollout, MPC, and learned-model implementations.

These questions are more durable than a catalogue of implementation names. They also expose where model information, samples, constraints, and real-time computation enter the controller.

C.7 Other Extended DP Models

Future revisions of this appendix will add self-contained entries for aggregation and discretization, belief-state control under partial observability, minimax and robust reinforcement learning, and selected technical proofs. Until those entries are complete, the corresponding chapter claims should remain scoped to the fully observed discounted models stated in the main text.

Bibliography

Anderson, B. D. O. and J. B. Moore (2007). *Optimal Control: Linear Quadratic Methods*. Mineola, NY: Dover Publications.

Bertsekas, D. P. (2019). *Reinforcement Learning and Optimal Control*. Belmont, MA: Athena Scientific. ISBN: 9781886529397.

– (2022). *Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control*. Nashua, NH: Athena Scientific. ISBN: 9781886529175.

– (2025). *A Course in Reinforcement Learning*. 2nd ed. Nashua, NH: Athena Scientific. ISBN: 9781886529298.

Boer, P.-T. de, D. P. Kroese, S. Mannor, and R. Y. Rubinstein (2005). "A Tutorial on the Cross-Entropy Method". In: *Annals of Operations Research* 134.1, pp. 19–67. DOI: 10.1007/s10479-005-5724-z.

Fazel, M., R. Ge, S. M. Kakade, and M. Mesbahi (2018). "Global Convergence of Policy Gradient Methods for the Linear Quadratic Regulator". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research, pp. 1467–1476. URL: https://proceedings.mlr.press/v80/fazel18a.html.

Fujimoto, S., H. van Hoof, and D. Meger (2018). "Addressing Function Approximation Error in Actor-Critic Methods". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research, pp. 1587–1596. URL: https://proceedings.mlr.press/v80/fujimoto18a.html.

Gu, S., T. Lillicrap, I. Sutskever, and S. Levine (2016). "Continuous Deep Q-Learning with Model-Based Acceleration". In: *Proceedings of the 33rd International Conference on Machine Learning*. Vol. 48. Proceedings of Machine Learning Research, pp. 2829–2838. URL: https://proceedings.mlr.press/v48/gu16.html.

Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine (2018). "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research, pp. 1861–1870. URL: https://proceedings.mlr.press/v80/haarnoja18b.html.

Hasselt, H. van, A. Guez, and D. Silver (2016). "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 2094–2100. DOI: 10.1609/aaai.v30i1.10295. URL: https://ojs.aaai.org/index.php/AAAI/article/view/10295.

Kingma, D. P. and J. Ba (2015). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*. arXiv: 1412.6980. URL: https://arxiv.org/abs/1412.6980.

Lagoudakis, M. G. and R. Parr (2003). "Least-Squares Policy Iteration". In: *Journal of Machine Learning Research* 4, pp. 1107–1149. URL: https://www.jmlr.org/papers/v4/lagoudakis03a.html.

Lattimore, T. and C. Szepesvári (2020). *Bandit Algorithms*. Cambridge: Cambridge University Press. DOI: 10.1017/9781108571401.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2016). "Continuous Control with Deep Reinforcement Learning". In: *International Conference on Learning Representations*. arXiv: 1509.02971. URL: https://arxiv.org/abs/1509.02971.

Loshchilov, I. and F. Hutter (2019). "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

Mayne, D. Q., J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert (2000). "Constrained Model Predictive Control: Stability and Optimality". In: *Automatica* 36.6, pp. 789–814. DOI: 10.1016/S0005-1098(99)00214-9.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015). "Human-Level Control Through Deep Reinforcement Learning". In: *Nature* 518.7540, pp. 529–533. DOI: 10.1038/nature14236. URL: https://doi.org/10.1038/nature14236.

Onori, S., L. Serrao, and G. Rizzoni (2016). *Hybrid Electric Vehicles: Energy Management Strategies*. SpringerBriefs in Electrical and Computer Engineering. London: Springer. DOI: 10.1007/978-1-4471-6781-5.

Rajamani, R. (2012). *Vehicle Dynamics and Control*. 2nd ed. New York, NY: Springer. DOI: 10.1007/978-1-4614-1433-9.

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

Schulman, J., P. Moritz, S. Levine, M. Jordan, and P. Abbeel (2016). "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *International Conference on Learning Representations*. arXiv: 1506.02438. URL: https://arxiv.org/abs/1506.02438.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). *Proximal Policy Optimization Algorithms*. arXiv: 1707.06347. URL: https://arxiv.org/abs/1707.06347.

Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller (2014). "Deterministic Policy Gradient Algorithms". In: *Proceedings of the 31st International Conference on Machine Learning*. Vol. 32. Proceedings of Machine Learning Research 1, pp. 387–395. URL: https://proceedings.mlr.press/v32/silver14.html.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning*

Research 15.56, pp. 1929–1958. URL: https://www.jmlr.org/papers/v15/srivastava14a.html.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge, MA: MIT Press. URL: http://incompleteideas.net/book/the-book-2nd.html.

Sutton, R. S., D. McAllester, S. Singh, and Y. Mansour (2000). "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems 12*. MIT Press, pp. 1057–1063.

Wabersich, K. P. and M. N. Zeilinger (2021). "A Predictive Safety Filter for Learning-Based Control of Constrained Nonlinear Dynamical Systems". In: *Automatica* 129, p. 109597. DOI: 10.1016/j.automatica.2021.109597.

Wang, Z., T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas (2016). "Dueling Network Architectures for Deep Reinforcement Learning". In: *Proceedings of the 33rd International Conference on Machine Learning*. Vol. 48. Proceedings of Machine Learning Research, pp. 1995–2003. URL: https://proceedings.mlr.press/v48/wangf16.html.

Watkins, C. J. C. H. and P. Dayan (1992). "Q-Learning". In: *Machine Learning* 8.3–4, pp. 279–292. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.