Tangent-Space Methods for Nonlinear Control and Biomechanics

Volume IV: Human Motor Control

The Neural Architecture of Movement

Preface

Draft Notice. This volume is under active development. Only Chapters 4 and 7b contain substantial content; the remaining chapters are structural outlines with preliminary material that will be expanded in future editions. Exercises, worked examples, and backmatter (glossary, further reading) have not yet been written.

Volumes 0–III developed the mathematical and physical foundations for understanding motion: differential geometry, nonlinear control, trajectory optimization, and biomechanics. This volume asks a fundamentally different question: *how does the brain actually control the body?*

The human motor system is arguably the most impressive control system in nature. A golfer coordinates 200+ degrees of freedom to produce a 1.5-second motion with centimeter-scale precision at the clubhead, despite processing delays of 100–200 ms, noisy sensors, and actuators whose force capacity depends on their own state. As of this writing, no engineered system matches this combination of dimensionality, precision, and robustness.

The central thesis of this volume is that the brain solves the seemingly impossible problem of high-dimensional motor control not through deep computation but through **massive parallelism with shallow depth**. The motor cortex has approximately 6 layers. At 1–5 ms per synaptic transmission, a motor command can pass through at most 20–40 layers of processing in the 100–200 ms available for real-time control. The brain compensates for this limited depth with enormous width: billions of neurons processing in parallel, each performing simple local computations.

This architectural constraint—shallow but wide—is not a limitation. It is the *key design principle* that makes biological motor control work. Understanding why is the purpose of this book.

Contents

Nomenclature

General Conventions

- Scalars are lowercase or Greek letters (e.g., m, t, θ).
- Vectors are bold lowercase (e.g., $\mathbf{x}, \mathbf{u}, \mathbf{q}$).
- Matrices are bold uppercase (e.g., $\mathbf{A}, \mathbf{B}, \mathbf{M}, \mathbf{J}$).
- Expected values and sets are denoted by blackboard bold or script (e.g., $\mathbb{E}, \mathcal{S}$).

State and Kinematics

$\mathbf{q} \in \mathbb{R}^n$ Joint configuration vector (generalized coordinates).

$\dot{\mathbf{q}} \in \mathbb{R}^n$ Joint velocity vector (generalized velocities).

$\ddot{\mathbf{q}} \in \mathbb{R}^n$ Joint acceleration vector.

$\mathbf{x} \in \mathbb{R}^{nx}$ System state vector; commonly $\mathbf{x} = [\mathbf{q}^T, \dot{\mathbf{q}}^T]^T$ for mechanical systems.

$\mathbf{u} \in \mathbb{R}^m$ Control input vector (e.g., joint torques, muscle activations).

$\boldsymbol{\tau} \in \mathbb{R}^n$ Generalized forces/torques acting on the joints.

$t \in \mathbb{R}$ Time.

Inertia and Dynamics

$\mathbf{M}(\mathbf{q})$ Mass (inertia) matrix, symmetric positive definite.

$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ Coriolis and centrifugal force matrix.

$\mathbf{g}(\mathbf{q})$ Gravity vector.

$\mathbf{J}(\mathbf{q})$ Jacobian matrix relating joint velocities to task-space velocities ($\dot{\mathbf{p}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$).

Control and Optimization

$\mathbf{A}_k = \frac{\partial f}{\partial \mathbf{x}}$ Local Jacobian matrix of the state dynamics.

$\mathbf{B}_k = \frac{\partial f}{\partial \mathbf{u}}$ Local Jacobian matrix of the control input.

$V(\mathbf{x})$ **or** $\mathcal{V}$ Value function or Lyapunov function.

$\mathbf{Q}$ State penalty matrix in LQR/DDP.

$\mathbf{R}$ Control penalty matrix in LQR/DDP.

$\mathbf{K}$ Feedback gain matrix ($\delta\mathbf{u} = -\mathbf{K}\delta\mathbf{x}$).

$\boldsymbol{\gamma}$ A trajectory or flow, mapping time and initial conditions to states.

Biomechanics and Motor Control

$a_i \in [0, 1]$ Muscle activation level.

ℓ_i^M Muscle fiber length.

v_i^M Muscle fiber contraction velocity.

ℓ_i^{MT} Musculotendon unit length.

$\mathbf{F_{muscle}}$ Force generated by muscles.

$\mathbf{W}$ Muscle synergy matrix (weights).

$\mathbf{c}(t)$ Muscle synergy activation vector over time.

Geometry and Manifolds

$SE(3)$ Special Euclidean group of rigid body motions.

$SO(3)$ Special Orthogonal group of 3D rotations.

$\mathfrak{se}(3)$ Lie algebra of $SE(3)$, space of spatial twists (velocities).

$\mathcal{M}$ A smooth manifold.

$T_{\mathbf{x}}\mathcal{M}$ Tangent space at point $\mathbf{x}$.

Chapter 1

The Degrees-of-Freedom Problem

Draft Notice. This chapter is under active development and contains only a structural outline with preliminary material. Full exposition, worked examples, proofs, and exercises will be added in future editions.

> The coordination of movement is the process of mastering redundant degrees of freedom of the moving organ, in other words its conversion to a controllable system.
>
> — Nikolai Bernstein, 1967

1.1 Bernstein's Original Formulation

In 1935, Nikolai Bernstein, a Soviet physiologist, made a deceptively simple observation: the number of degrees of freedom available for any motor task *vastly exceeds* the number required to define the task. This observation, formalized in his 1967 monograph [5], is the founding problem of motor control.

Definition 1.1. The Degrees-of-Freedom Problem

Let a motor task be defined by k task-space constraints (e.g., $k = 6$ for placing a hand at a specific position and orientation). Let the body have n joint degrees of freedom and $p > n$ muscles. The **degrees-of-freedom problem** is: how does the nervous system select a specific solution $(\boldsymbol{q}(t), \boldsymbol{u}(t)) \in \mathbb{R}^n \times \mathbb{R}^p$ from the infinite set of solutions consistent with the task constraints?
The **kinematic redundancy** is $n - k$ (extra joint configurations). The **muscle redundancy** is $p - n$ (extra muscles per joint). The **total redundancy** is $(n-k)+(p-n) = p-k$.

1.1.1 The Numbers Are Staggering

Consider the golf swing:

Quantity	Approximate Value
Skeletal DOF involved	∼200
Muscles contributing	∼400
Task constraints (clubhead at impact)	6
Kinematic redundancy	∼194
Muscle redundancy	∼200
Total redundancy	∼394
Duration of downswing	∼0.25 s
Neural processing delay	∼0.1 – 0.2 s

The nervous system must select from ∼400-dimensional muscle activation patterns to satisfy 6 constraints, while accounting for interaction torques, gravity, and neural delays—*in a quarter of a second*. This is the problem.

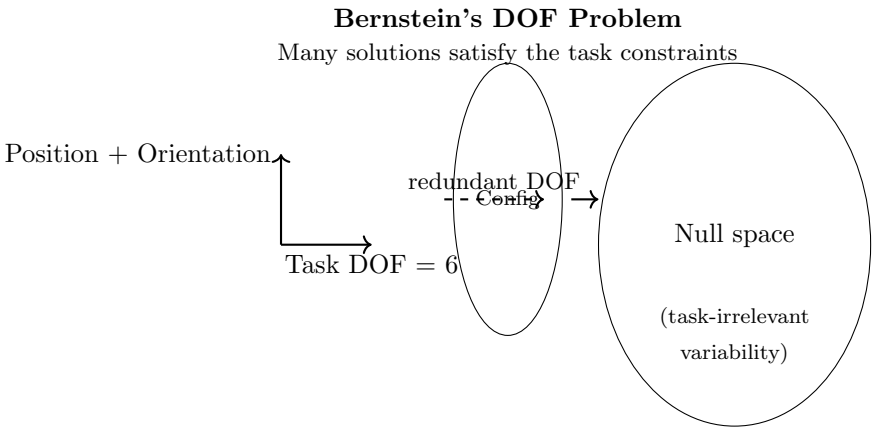


Figure 1.1: The degrees-of-freedom problem: redundancy in configuration space allows task-irrelevant variability in the null space of the task Jacobian.

1.2 From Problem to Resource: Motor Abundance

Modern motor control theory has reframed Bernstein's "problem" as an *opportunity*. Redundancy is not a curse but a resource:

Key Insight 1.1. Redundancy as Motor Abundance

The extra degrees of freedom provide:

1. **Flexibility**: the same task can be performed in many ways, allowing adaptation to constraints (fatigue, injury, obstacles)
2. **Robustness**: perturbations can be absorbed without task failure

3. **Exploration**: variability in execution allows the system to discover better solutions over time
4. **Error correction**: task-irrelevant DOF can compensate for errors in task-relevant DOF

We adopt the term **motor abundance** (Latash 2012) to emphasize that redundancy is a feature, not a bug.

1.3 The Uncontrolled Manifold Hypothesis

The uncontrolled manifold (UCM) hypothesis [32] provides a mathematical framework for understanding how the nervous system manages redundancy.

Definition 1.2. The Uncontrolled Manifold

Let $\boldsymbol{q} \in \mathbb{R}^n$ be the joint configuration and $\boldsymbol{y} = h(\boldsymbol{q}) \in \mathbb{R}^k$ be the task variable. The **uncontrolled manifold** at a nominal configuration $\boldsymbol{q}_0$ is the null space of the task Jacobian:

$$\mathrm{UCM}(\boldsymbol{q}_0) = \{\delta\boldsymbol{q} \in \mathbb{R}^n : J_h(\boldsymbol{q}_0)\delta\boldsymbol{q} = \boldsymbol{0}\}, \tag{1.1}$$

where $J_h = \partial h/\partial \boldsymbol{q}$ is the task Jacobian. Variability within the UCM does not affect the task variable. Variability perpendicular to the UCM (in the **orthogonal complement**) directly affects task performance.

Geometrically, $\mathrm{UCM}(\boldsymbol{q}_0)$ is the kernel of the task map's differential. Where J_h has full rank k, the implicit function theorem makes h a submersion near $\boldsymbol{q}_0$, so the level set $h^{-1}(\boldsymbol{y}_0)$ is a smooth $(n-k)$-dimensional manifold whose tangent space at $\boldsymbol{q}_0$ is exactly the UCM. Motion along that manifold changes the configuration without changing the task variable.

It is worth being explicit that this is a statement about the *task map*, not about the dynamics. The UCM is a redundancy of h, not an uncontrollable subspace of the control system: a configuration can be fully controllable and still have a large uncontrolled manifold, because controllability asks what states are reachable while the UCM asks which state changes the task cannot see. Conflating the two makes the hypothesis sound like a claim about what the nervous system *cannot* do, when it is a claim about what it need not bother to correct.

In Plain Language 1.1. Mathematical Reminder: The Geometry of the UCM

Recall our differential geometry from Volume 0: a manifold is a curved surface representing valid states. The **Uncontrolled Manifold** (UCM) is quite literally a geometric submanifold embedded inside the larger configuration space $\mathcal{Q}$.
Because the task geometry is non-linear (e.g., the arc of a swinging robot arm), the UCM is curved. We calculate its flat **Tangent Space** mathematically using the null space of the Jacobian ($J_h\delta\boldsymbol{q} = \boldsymbol{0}$). As long as the nervous system's noise vectors point exclusively into this tangent space, the physical position of the hand will not

budge from the target. The nervous system is actively shaping its noise ellipsoid to align with the tangent space of the target manifold.

The UCM hypothesis predicts that the nervous system:

1. **Suppresses** variability in the orthogonal complement of the UCM (task-relevant variability)
2. **Tolerates** or even **encourages** variability within the UCM (task-irrelevant variability)

This has been experimentally confirmed across many motor tasks [32, 38].

```python
import numpy as np
from typing import Tuple

def ucm_analysis(
    joint_angles: np.ndarray,
    task_jacobian_func,
    n_repetitions: int | None = None
) -> Tuple[float, float, float]:
    """Perform UCM variance decomposition.

    Parameters
    ----------
    joint_angles : np.ndarray, shape (n_reps, n_dof)
        Joint angle data from multiple repetitions of the same task.
    task_jacobian_func : callable
        q -> J(q), shape (k, n_dof), the task Jacobian.
    n_repetitions : int, optional
        Number of repetitions (inferred from data if not given).

    Returns
    -------
    Tuple[float, float, float]
        (variance_ucm, variance_ort, ratio)
        UCM variance, orthogonal variance, and their ratio.
    """
    n_reps, n_dof = joint_angles.shape
    q_mean = np.mean(joint_angles, axis=0)

    # Task Jacobian at the mean configuration
    J = task_jacobian_func(q_mean)
    k = J.shape[0]  # task dimension

    # Null space of J (UCM directions)
    U, S, Vt = np.linalg.svd(J, full_matrices=True)
    rank = np.sum(S > 1e-10)
    null_space = Vt[rank:].T  # Shape: (n_dof, n_dof - rank)

    # Orthogonal complement (range space)
    range_space = Vt[:rank].T  # Shape: (n_dof, rank)

    # Decompose variability
    deviations = joint_angles - q_mean  # (n_reps, n_dof)

    # Project onto UCM
    proj_ucm = deviations @ null_space  # (n_reps, n_dof - rank)
    var_ucm = np.mean(np.sum(proj_ucm**2, axis=1))
```

```python

    # Project onto orthogonal complement
    proj_ort = deviations @ range_space  # (n_reps, rank)
    var_ort = np.mean(np.sum(proj_ort**2, axis=1))

    # Normalize by dimensionality
    var_ucm_per_dof = var_ucm / (n_dof - rank) if n_dof > rank else 0
    var_ort_per_dof = var_ort / rank if rank > 0 else 0

    ratio = var_ucm_per_dof / var_ort_per_dof if var_ort_per_dof > 0 else np.
    inf

    return var_ucm_per_dof, var_ort_per_dof, ratio


# Example: 3-joint planar arm reaching to a point
def planar_arm_jacobian(q: np.ndarray, L: np.ndarray = np.array([0.3, 0.25,
    0.2])):
    """Task Jacobian for endpoint position of a 3-link planar arm."""
    n = len(q)
    J = np.zeros((2, n))
    for i in range(n):
        for j in range(i, n):
            angle_sum = np.sum(q[:j+1])
            J[0, i] -= L[j] * np.sin(angle_sum)
            J[1, i] += L[j] * np.cos(angle_sum)
    return J

# Generate synthetic reaching data (50 repetitions)
np.random.seed(42)
q_nominal = np.array([0.5, 1.2, -0.3])
n_reps = 50
# Motor noise: more variability in task-irrelevant directions
noise = np.random.randn(n_reps, 3) * 0.05
q_data = q_nominal + noise

v_ucm, v_ort, ratio = ucm_analysis(q_data, planar_arm_jacobian)
print(f"UCM variance (per DOF): {v_ucm:.6f}")
print(f"ORT variance (per DOF): {v_ort:.6f}")
print(f"UCM/ORT ratio:          {ratio:.2f}")
print(f"(Ratio > 1 supports the UCM hypothesis)")
```

Listing 1.1: Uncontrolled manifold analysis.

1.4 Bernstein's Three Stages of Learning

Bernstein proposed that motor skill acquisition progresses through three stages:

1. **Freezing**: initially, the learner constrains ("freezes") excess degrees of freedom, reducing the system to manageable dimensionality. A beginner golfer keeps wrists rigid, eliminating the wrist DOF.

2. **Freeing**: as skill develops, the learner gradually releases ("frees") the frozen DOF, allowing them to participate in the movement. The intermediate golfer adds wrist hinge.

3. **Exploiting**: the expert learner exploits the released DOF to harness reactive forces, elastic energy, and passive dynamics. The expert golfer uses the passive wrist release driven by centrifugal force.

In the language of control theory:

- Stage 1: reduce the system dimension from n to k (the task dimension)
- Stage 2: increase the controlled dimension while maintaining task performance
- Stage 3: exploit the drift dynamics $f(\boldsymbol{x})$ by designing trajectories that use passive mechanics (Volume II, Chapter 5). Agrachev & Sachkov [1] develop the geometric theory of reachable sets in Chapter 8; what a system can reach depends on its drift as much as on its inputs, which is why releasing a degree of freedom can enlarge the reachable set without adding an actuator.

Exercises

1. **Redundancy Quantification.** For each of the following tasks, estimate the task dimension k, body DOF n, and compute the kinematic redundancy: (a) Standing balance, (b) Picking up a cup, (c) Walking on a treadmill, (d) A tennis serve.
2. **UCM Analysis.** Using the provided code, generate synthetic data for 3-joint planar reaching with structured variability (more noise in the null space of the Jacobian). Verify that the UCM/ORT ratio exceeds 1.
3. **Bernstein's Stages.** Observe a beginner and an expert performing the same motor task (e.g., throwing a ball). Describe their movement in terms of Bernstein's three stages. Which degrees of freedom are frozen, freed, and exploited?
4. **(Programming.)** Implement a UCM analysis for a 7-DOF arm model reaching to a 3D target. Visualize the UCM and ORT directions.
5. **(Research.)** Read Scholz and Schöner (1999) and summarize the key experimental evidence for the UCM hypothesis.

Chapter 2

The Curse of Dimensionality — And How Humans Solve It

Draft Notice. This chapter is under active development and contains only a structural outline with preliminary material. Full exposition, worked examples, proofs, and exercises will be added in future editions.

In ten dimensions, the unit hypercube has $2^{10} = 1024$ corners. In a hundred, it has $2^{100} \approx 10^{30}$. The nervous system controls hundreds of dimensions every second.

2.1 Classical Curse of Dimensionality

The curse of dimensionality [4] states that the computational resources needed to solve many problems grow *exponentially* with the state dimension n:

Definition 2.1. The Curse of Dimensionality

For a control problem with state dimension n:

- **Grid-based methods** (dynamic programming): cost $\sim \mathcal{O}(N^n)$ where N is the grid resolution per dimension
- **Sample-based methods** (Monte Carlo, RL): sample complexity grows polynomially in n but with large constants
- **Trajectory optimization** (DDP, collocation): cost $\sim \mathcal{O}(n^3 \cdot T)$ per iteration — the only scalable approach for deterministic systems

2.2 Biological Solutions

Humans solve the dimensionality problem through five complementary mechanisms:

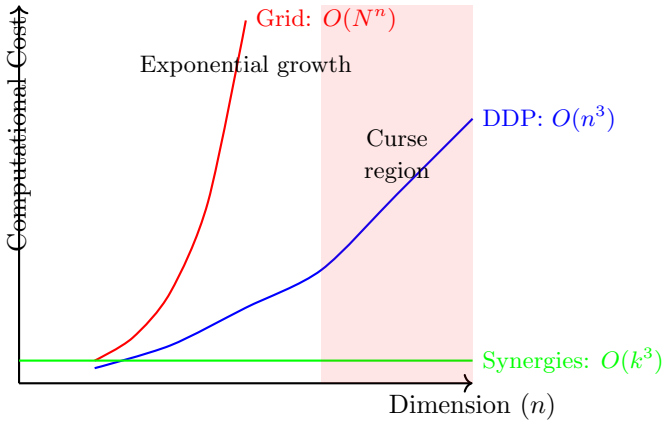


Figure 2.1: The curse of dimensionality: computational cost grows exponentially with dimension for grid-based methods, cubically for DDP, and remains constant when using synergy-based dimensionality reduction (where $k \ll n$).

2.2.1 1. Synergy-Based Compression

As introduced in Chapter 4: $\boldsymbol{u}(t) = W\boldsymbol{c}(t)$ reduces p-dimensional muscle space to k-dimensional synergy space, with $k/p \approx 4/400 = 1\%$.

2.2.2 2. Hierarchical Decomposition

Multi-level processing decomposes the global problem into independent subproblems.

2.2.3 3. Passive Dynamics Exploitation

The drift field $f(\boldsymbol{x})$ does useful work. The controller only corrects deviations. This reduces the effective control dimension (Volume II, passive dynamics).

2.2.4 4. Internal Models and Prediction

Forward models predict consequences, enabling open-loop execution with sparse feedback corrections (see Chapter 6).

2.2.5 5. Task-Space Control

Control in a low-dimensional task space rather than the full joint/muscle space. The null space absorbs variability (UCM framework, Chapter 1).

```python
import numpy as np

def computational_cost_table(n_dims: list[int]) -> None:
    """Print computational cost for different methods vs dimension."""
    print(f"{'n_dim':>6} {'Grid (N=10)':>15} {'DDP (T=100)':>15} "
          f"{'Synergy (k=5)':>15}")
    print("-" * 55)
    for n in n_dims:
```

```
        grid_cost = 10**n if n <= 10 else float('inf')
        ddp_cost = n**3 * 100
        synergy_cost = 5**3 * 100  # After reduction to k=5
        print(f"{n:>6d} {grid_cost:>15.0e} {ddp_cost:>15.0e} "
              f"{synergy_cost:>15.0e}")

computational_cost_table([2, 5, 10, 20, 50, 100, 200])
```

Listing 2.1: Computational cost comparison across methods.

Exercises

1. **Scaling Analysis.** Using the computational cost table, determine the maximum state dimension for which grid-based dynamic programming is feasible (assuming 10^{12} FLOPS).

2. **Synergy Efficiency.** If a motor task uses 100 muscles with 5 synergies, by what factor does the synergy decomposition reduce the dimensionality of the control problem?

3. **(Programming.)** Implement DDP for a 2-DOF, 5-DOF, and 10-DOF chain. Measure computation time per iteration. Verify cubic scaling in state dimension.

Chapter 3

Neural Architecture of Motor Control

Draft Notice. This chapter is under active development and contains only a structural outline with preliminary material. Full exposition, worked examples, proofs, and exercises will be added in future editions.

The motor system is not one controller. It is an orchestra of specialized modules, each playing its part in the symphony of movement.

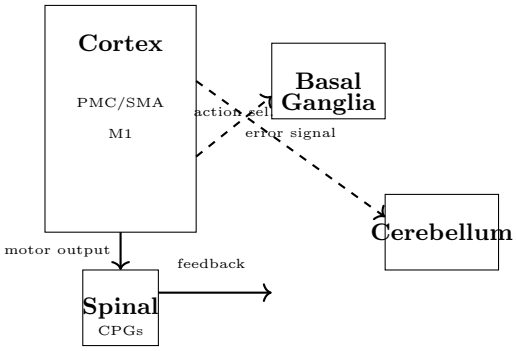


Figure 3.1: Hierarchical neural motor architecture: cortex plans/initiates, basal ganglia selects actions, cerebellum provides error-based learning, spinal cord implements CPGs and reflex circuits.

3.1 Cortical Motor Areas

The cerebral cortex contains several distinct motor areas, each with specialized roles:

3.1.1 Primary Motor Cortex (M1)

M1 (Brodmann area 4) is the final cortical output stage for voluntary movement. Layer V contains the giant pyramidal (Betz) cells whose axons form the corticospinal tract—the most direct neural pathway from cortex to spinal motor neurons.

Key properties of M1 (experimentally confirmed):

- **Somatotopic organization**: body parts are mapped to cortical regions (the motor homunculus), with overlapping representations [31]
- **Population coding**: individual neurons have broad cosine-shaped directional tuning about a preferred direction [12], and movement direction is recovered from the vector sum of the population's contributions rather than from any single cell [13]
- **Preparatory activity**: M1 neurons show activity 100–200 ms before movement onset during motor preparation, reflecting planning processes

3.1.2 Premotor Cortex (PMC) and Supplementary Motor Area (SMA)

- **PMC**: movement planning based on external cues (visually guided reaching)
- **SMA**: movement planning based on internal cues (self-initiated sequences)
- Both areas project to M1 and directly to the spinal cord

3.2 Subcortical Motor Structures

3.2.1 Cerebellum: The Forward Model

The human cerebellum contains roughly 69 billion neurons—about 80% of all the neurons in the brain, and overwhelmingly granule cells—against some 16 billion in the cerebral cortex [3]. Electrophysiological and lesion studies establish its primary role in motor control as a **forward model**:

Neural Architecture 3.1. The Cerebellum as a Forward Model

The cerebellar circuit [28, 2, 24]:

1. Receives a copy of the motor command (efference copy) via the pontine nuclei
2. Receives sensory feedback via the inferior olive and mossy fibers
3. Computes the predicted sensory consequences of the command using parallel fiber–Purkinje cell synapses (basis function expansion)
4. The climbing fiber signal carries sensory prediction error from the inferior olive
5. Error-driven learning updates Purkinje cell responses through long-term depression (LTD) at parallel fiber–Purkinje cell synapses. LTD is induced only when parallel fiber activity and the climbing fiber spike fall within a window of a few hundred milliseconds of each other; the optimum is region-specific and appears tuned to each region's own sensorimotor feedback delay [37]

In the language of control theory (Volume I):

- The cerebellum implements $\hat{\boldsymbol{x}}_{k+1} = f(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k)$ — the state prediction step of the Kalman filter
- The climbing fiber error signal implements the innovation update
- The granule cell layer provides the basis functions for the nonlinear mapping

Two lines of evidence support this architecture, and it is worth keeping them apart because they come from different populations.

In healthy subjects, reaching against a novel force field is initially distorted and then straightens with practice; removing the field produces aftereffects that mirror the original distortion, which is what implicates a learned internal model of the limb's dynamics rather than a stiffer or better-tuned feedback loop [34]. In patients with cerebellar degeneration, on-line correction within a movement is intact while trial-to-trial learning of those same altered dynamics is profoundly impaired—a dissociation that does not appear in Huntington's disease, where the deficit falls on the feedback side instead [35]. The first result establishes that an internal model is learned; the second locates the learning in the cerebellum.

Two timescales are easy to conflate here, and they are not the same quantity. The window in which parallel fiber and climbing fiber activity must coincide for LTD to be induced is a few hundred milliseconds—that is a coincidence-detection requirement, set by how long the error signal takes to arrive. The depression it produces, and the behavioural adaptation that follows, accumulate over minutes and tens of trials. Motor learning being slow is therefore not in tension with the plasticity rule being fast: the millisecond figure governs *whether* a synapse is eligible to change, the minutes figure governs *how much* it has changed.

3.2.2 Basal Ganglia: Action Selection

The basal ganglia implement an action selection mechanism through competitive inhibition. The direct pathway facilitates a desired action; the indirect pathway suppresses competing actions. This is analogous to a model predictive controller evaluating multiple candidate trajectories and selecting the best one.

3.3 Spinal Cord: The Local Controller

The spinal cord is not merely a relay station. It contains:

- **Motor neuron pools**: lower motor neurons that directly innervate muscles
- **Interneuron networks**: local circuits that implement reflexes, pattern generation, and coordination
- **Central pattern generators** (CPGs): oscillatory circuits for rhythmic movements (Chapter 8)

Exercises

1. **Population Vector.** Given neurons with preferred directions every 45°, compute the population vector for a movement at 30°. Plot individual neuron contributions.

2. **Cerebellar Forward Model.** Implement a simple forward model using a lookup table. Show how climbing fiber errors update the model over successive trials.

3. **(Research.)** Read Georgopoulos et al. (1986) and summarize the evidence for population coding in M1.

Chapter 4

Shallow-but-Wide: Why Brains Are Parallel, Not Deep

The brain is not a deep learning system. It is a massively parallel, notably shallow one. This is not a bug — it is the architecture that makes real-time motor control possible.

4.1 The Speed Constraint

The most fundamental constraint on neural computation is **speed**. Every biological computation has a time cost determined by the physics of synaptic transmission:

Neural Architecture 4.1. Neural Timing Constraints

Measured values from neurophysiology [17, 23]:

- **Synaptic delay**: 1–5 ms per chemical synapse (AMPA receptor rise time ∼1 ms)
- **Axonal conduction**: unmyelinated ∼1 m/s, myelinated ∼100 m/s
- **Action potential**: rising phase 1 ms, falling phase 2–3 ms
- **Refractory period**: absolute ∼1–2 ms (sodium channel inactivation)
- **Total per-layer delay**: ∼5–10 ms (synaptic + integration + propagation)

These delays are not technological limitations to be overcome with faster hardware. They are *fundamental physical constraints* of electrochemical signaling. No amount of biological optimization can reduce synaptic delay below ∼1 ms.

4.1.1 The Depth Budget

Given the timing constraints, we can compute the maximum depth of any neural computation that must complete within a given time window:

Definition 4.1. The Neural Depth Budget

For a motor task requiring a response within time T_{response}, the maximum number of serial processing layers is:

$$L_{\max} = \frac{T_{\text{response}}}{\tau_{\text{layer}}}, \tag{4.1}$$

where $\tau_{\text{layer}} \approx 5$–$10$ ms is the per-layer processing time. For typical motor control scenarios:

Task	T_{response}	$L_{\max}$
Spinal reflex	30 ms	3–6
Postural correction	100 ms	10–20
Voluntary movement initiation	150–250 ms	15–50
Golf swing correction (mid-downswing)	50 ms	5–10
Saccadic eye movement	200 ms	20–40

Compare this with modern deep learning: GPT-4 has ∼120 transformer layers; ResNet-152 has 152 layers. These run on silicon at ∼ns per operation. Biology cannot afford this depth at ∼ms per operation.

4.2 The Motor Cortex: Six Layers, Not Sixty

The cerebral cortex has a notably consistent six-layered architecture across mammalian species [30]. This constraint applies uniformly to M1 and other cortical areas:

1. **Layer I**: molecular layer (mostly dendrites and axons, few cell bodies)
2. **Layer II/III**: external pyramidal layer (cortico-cortical connections)
3. **Layer IV**: internal granular layer (thalamic input)
4. **Layer V**: internal pyramidal layer (*output layer* — corticospinal tract)
5. **Layer VI**: multiform layer (feedback to thalamus)

The signal path from sensory input (Layer IV) through cortical processing to motor output (Layer V) passes through *at most 3–4 synapses within the cortex itself.* Adding thalamic relay and spinal motor neuron connections, the total sensorimotor loop is approximately 8–12 synapses deep.

Key Insight 4.1. The Architecture Is the Algorithm

In deep learning, we can stack layers because silicon is fast. In biology, we cannot. Instead, the brain uses:

1. **Width**: approximately 16 billion cortical neurons in humans [3], each receiving on the order of 10^4 synaptic inputs [23]. The total computational capacity is

in the connections, not the depth.

2. **Recurrence**: horizontal connections within layers create recurrent dynamics that effectively increase computational depth through temporal iteration (but at the cost of time). These recurrent loops operate at 10–100 ms timescales within a single motor control cycle.

3. **Specialization**: different brain regions process different aspects of the motor task in parallel, communicating through thick fiber bundles (corpus callosum, cerebral peduncles).

4. **Hierarchy**: a shallow hierarchy of cortex → subcortical structures (basal ganglia, cerebellum, brainstem) → spinal cord → muscles, with each level operating at different timescales and abstraction levels.

This architectural choice has far-reaching consequences. While a deep network might process sensory information in 50 serial steps, the brain achieves comparable flexibility through width and recurrence. Consider a reaching task: the cortex does not recompute the entire motor plan from scratch at each millisecond. Instead, it maintains a dynamical state that evolves according to its intrinsic connectivity. Perturbations are handled through feedback loops that operate in parallel at multiple levels.

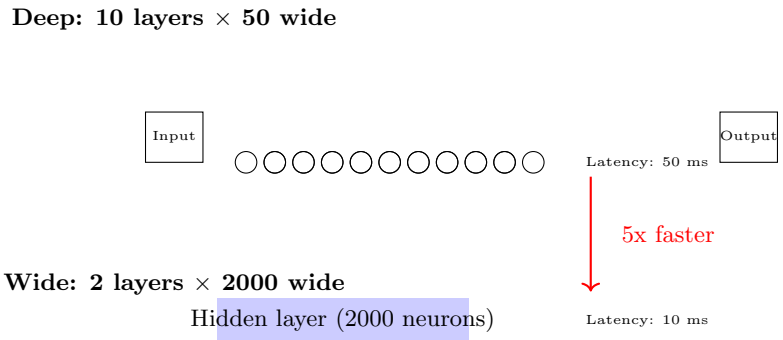


Figure 4.1: Shallow-wide architecture trades depth for width: a 2-layer network with 2000 neurons per layer completes in 10 ms, while a 10-layer network with 50 neurons per layer takes 50 ms—too slow for real-time motor control.

4.3 Computational Implications

The shallow-but-wide architecture has notable implications for motor control theory:

4.3.1 Complex Computations Through Simple Local Rules

Each neuron performs a relatively simple computation: weighted sum of inputs, passed through a nonlinear activation function. The power of the system comes from the *collective behavior* of billions of these simple units operating in parallel.

```python
import numpy as np
import time

def deep_network_forward(
    x: np.ndarray,
    weights: list[np.ndarray],
    biases: list[np.ndarray],
    layer_delay_ms: float = 5.0
) -> tuple[np.ndarray, float]:
    """Forward pass through a deep network with biological timing.

    Parameters
    ----------
    x : np.ndarray
        Input vector.
    weights : list of np.ndarray
        Weight matrices for each layer.
    biases : list of np.ndarray
        Bias vectors for each layer.
    layer_delay_ms : float
        Simulated per-layer delay in milliseconds.

    Returns
    -------
    tuple
        (output, total_latency_ms)
    """
    h = x
    total_latency = 0.0
    for W, b in zip(weights, biases):
        h = np.maximum(0, W @ h + b)  # ReLU activation
        total_latency += layer_delay_ms
    return h, total_latency


def wide_network_forward(
    x: np.ndarray,
    W_hidden: np.ndarray,
    b_hidden: np.ndarray,
    W_out: np.ndarray,
    b_out: np.ndarray,
    layer_delay_ms: float = 5.0
) -> tuple[np.ndarray, float]:
    """Forward pass through a wide (2-layer) network.

    Parameters
    ----------
    x : np.ndarray
        Input vector.
    W_hidden : np.ndarray
        Hidden layer weights (wide).
    b_hidden : np.ndarray
        Hidden layer biases.
    W_out : np.ndarray
        Output layer weights.
    b_out : np.ndarray
        Output biases.
    layer_delay_ms : float
        Per-layer delay.

```

```
    Returns
    -------
    tuple
        (output, total_latency_ms)
    """
    h = np.maximum(0, W_hidden @ x + b_hidden)
    y = W_out @ h + b_out
    total_latency = 2 * layer_delay_ms  # Only 2 layers
    return y, total_latency


# Compare architectures for a motor control task
# Input: 20-dim state, Output: 10-dim muscle activations
np.random.seed(42)
n_input, n_output = 20, 10
n_params_budget = 50000  # Same total parameter budget

# Deep network: 10 layers of width 50
deep_widths = [n_input] + [50] * 9 + [n_output]
deep_W = [np.random.randn(deep_widths[i+1], deep_widths[i]) * 0.1
          for i in range(len(deep_widths)-1)]
deep_b = [np.zeros(deep_widths[i+1]) for i in range(len(deep_widths)-1)]

# Wide network: 2 layers, width 2000
wide_hidden = 2000
W_h = np.random.randn(wide_hidden, n_input) * 0.1
b_h = np.zeros(wide_hidden)
W_o = np.random.randn(n_output, wide_hidden) * 0.1
b_o = np.zeros(n_output)

x = np.random.randn(n_input)

y_deep, latency_deep = deep_network_forward(x, deep_W, deep_b)
y_wide, latency_wide = wide_network_forward(x, W_h, b_h, W_o, b_o)

print(f"Deep network (10 layers x 50 wide):")
print(f"  Latency: {latency_deep:.0f} ms")
print(f"  Parameters: {sum(W.size + b.size for W, b in zip(deep_W, deep_b))
    :,}")

print(f"\nWide network (2 layers x 2000 wide):")
print(f"  Latency: {latency_wide:.0f} ms")
print(f"  Parameters: {W_h.size + b_h.size + W_o.size + b_o.size:,}")

print(f"\nLatency ratio: {latency_deep/latency_wide:.1f}x faster with wide
    network")
```

Listing 4.1: Comparison: deep vs. wide network for motor control.

4.3.2 Hierarchical Decomposition

The brain does not solve the full-dimensional control problem in one computation. Instead, it decomposes the problem hierarchically:

1. **High level** (prefrontal cortex, ∼200 ms): task goal, strategy
2. **Mid level** (premotor/SMA, ∼100 ms): movement plan, sequencing
3. **Low level** (M1 + spinal cord, ∼30 ms): muscle activation patterns

Each level operates on a compressed representation of the level below:

Proposition 4.1. *In a hierarchical motor control architecture with L levels, each reducing dimensionality by a factor r, the effective state dimension at level l is:*

$$n_l = n_0 / r^l, \tag{4.2}$$

where n_0 is the full state dimension. The computational cost per level scales as $\mathcal{O}(n_l^2)$ for a single-layer network, giving total cost $\sum_{l=0}^{L} n_l^2 = n_0^2 \sum_{l=0}^{L} r^{-2l}$, which converges for $r > 1$. The curse of dimensionality is broken by hierarchy.

4.3.3 Synergies as Dimensionality Reduction

Muscle synergies provide direct evidence for dimensionality reduction. d'Avella and Bizzi showed that the muscle patterns of natural frog hindlimb behaviours decompose into a small set of synergies, some shared across behaviours and some specific to one [10]. How far that conclusion depends on the decomposition used is a separate question, and Tresch and colleagues addressed it directly by running several matrix factorization algorithms against the same simulated and experimental data: the algorithms broadly agree on the synergies they recover, which is what makes the finding a property of the data rather than of the method [39]. The muscle activation pattern of p muscles is approximated as a linear combination of $k \ll p$ synergies:

$$\boldsymbol{u}(t) = \sum_{i=1}^{k} c_i(t) \boldsymbol{w}_i = W \boldsymbol{c}(t), \tag{4.3}$$

where $W \in \mathbb{R}^{p \times k}$ is the synergy matrix (fixed spatial patterns) and $\boldsymbol{c}(t) \in \mathbb{R}^k$ are the time-varying synergy activations.

Experimental studies across diverse tasks (reaching, walking, postural responses) consistently identify $k = 4$–6 synergies accounting for $> 90\%$ variance (Tresch et al. 2006, Ting and Macpherson 2005).

```python
import numpy as np

def extract_synergies(
    emg_data: np.ndarray,
    n_synergies: int,
    max_iter: int = 500,
    tol: float = 1e-6
) -> tuple[np.ndarray, np.ndarray, float]:
    """Extract muscle synergies using NMF.

    Parameters
    ----------
    emg_data : np.ndarray, shape (n_timepoints, n_muscles)
        Processed EMG data (non-negative).
    n_synergies : int
        Number of synergies to extract.
    max_iter : int
        Maximum NMF iterations.
    tol : float
        Convergence tolerance.

    Returns
```

```
    -------
    tuple
        (synergy_weights, synergy_activations, vaf)
        W: (n_muscles, n_synergies)
        C: (n_timepoints, n_synergies)
        vaf: variance accounted for (0-1)
    """
    n_t, n_m = emg_data.shape

    # Initialize with random non-negative values
    W = np.abs(np.random.randn(n_m, n_synergies)) * 0.01
    C = np.abs(np.random.randn(n_t, n_synergies)) * 0.01

    for iteration in range(max_iter):
        # Multiplicative update rules (Lee & Seung 2001)
        # Update C
        numerator = emg_data @ W
        denominator = C @ W.T @ W + 1e-10
        C *= numerator / denominator

        # Update W
        numerator = emg_data.T @ C
        denominator = W @ C.T @ C + 1e-10
        W *= numerator / denominator

        # Check convergence
        reconstruction = C @ W.T
        residual = np.sum((emg_data - reconstruction)**2)
        total = np.sum(emg_data**2)
        vaf = 1.0 - residual / total

        if iteration > 0 and abs(vaf - prev_vaf) < tol:
            break
        prev_vaf = vaf

    return W, C, vaf


# Example: simulate EMG from 8 muscles during walking
np.random.seed(42)
n_timepoints = 200
n_muscles = 8

# Ground truth: 3 synergies generating 8-muscle activation
true_W = np.array([
    [0.8, 0.6, 0.1, 0.0, 0.2, 0.0, 0.1, 0.0],  # Syn 1: extensors
    [0.0, 0.1, 0.7, 0.9, 0.0, 0.3, 0.0, 0.1],  # Syn 2: flexors
    [0.2, 0.0, 0.1, 0.0, 0.8, 0.7, 0.6, 0.9],  # Syn 3: stabilizers
]).T  # (8 muscles, 3 synergies)

t = np.linspace(0, 2*np.pi, n_timepoints)
true_C = np.column_stack([
    np.maximum(0, np.sin(t)),
    np.maximum(0, np.sin(t + 2*np.pi/3)),
    np.maximum(0, np.sin(t + 4*np.pi/3)),
])

emg_simulated = true_C @ true_W.T + np.abs(np.random.randn(n_timepoints,
    n_muscles)) * 0.05

```

```
# Extract synergies
W_est, C_est, vaf = extract_synergies(emg_simulated, n_synergies=3)
print(f"Synergy extraction: {vaf:.1%} variance accounted for with 3 synergies
    ")

# Test with different numbers
for k in range(1, 7):
    _, _, vaf_k = extract_synergies(emg_simulated, n_synergies=k)
    print(f"  k={k}: VAF = {vaf_k:.1%}")
```

Listing 4.2: Muscle synergy extraction using Non-negative Matrix Factorization.

4.4 The Shallow-Wide Hypothesis for Motor Control

We now formalize the central thesis of this volume:

Key Insight 4.2. The Shallow-Wide Hypothesis

Biological motor control systems achieve high-dimensional control through architectures that are:

1. **Shallow**: at most 10–20 serial processing layers (limited by synaptic delay)
2. **Wide**: millions to billions of units per layer (enabled by neural density and parallel wiring)
3. **Modular**: functionally specialized subnetworks process different aspects of the task in parallel
4. **Hierarchical**: a small number of levels, each compressing dimensionality
5. **Recurrent**: intra-layer connections provide temporal depth without adding spatial depth

This architecture can approximate any smooth mapping (universal approximation theorem) with a single hidden layer of sufficient width. The practical advantage over deep networks is **latency**: a 2-layer network with 10,000 neurons per layer completes in 10 ms. A 100-layer network with 100 neurons per layer completes in 500 ms—too slow for motor control.
For real-time motor control, width is cheap and depth is expensive.

Exercises

1. **Depth Budget.** For a goalkeeper diving to save a penalty kick (reaction time ∼200 ms), compute the maximum neural depth. How deep can the "controller" be?

2. **Width vs. Depth.** Using the provided code, compare the outputs of a 10-layer × 50-wide network with a 2-layer × 2000-wide network on random inputs. Compare latencies.

3. **Synergy Analysis.** Using the NMF code, determine how many synergies are needed to explain $> 95\%$ of variance in the simulated EMG data. Does this match the literature value of 4–6?

4. **(Programming.)** Implement a motor control task (reaching) using both a deep and a wide network architecture. Compare tracking accuracy at different latency budgets.

5. **(Research.)** Find experimental evidence for the number of cortical layers involved in processing a motor command from M1 to spinal motor neurons. How many synapses does the signal cross?

Chapter 5

Ideomotor Theory and the Predictive Brain

Draft Notice. This chapter is under active development and contains only a structural outline with preliminary material. Full exposition, worked examples, proofs, and exercises will be added in future editions.

> Every representation of a movement awakens in some degree the actual movement which is its object.
>
> — William James, 1890

5.1 Historical Ideomotor Theory

The ideomotor principle, articulated by William James in 1890 [22], states that the mere *idea* of a movement tends to produce that movement. Modern reformulations by Greenwald (1970) and Hommel et al. (2001) have transformed this philosophical observation into a testable scientific framework.

Definition 5.1. The Theory of Event Coding (TEC)

The Theory of Event Coding [19] proposes that:

1. Perception and action share a **common representational format** (event codes or feature codes)
2. Actions are planned in terms of their **desired sensory effects**, not in terms of muscle commands
3. The mapping from desired effects to motor commands is learned through experience (bidirectional action-effect associations)

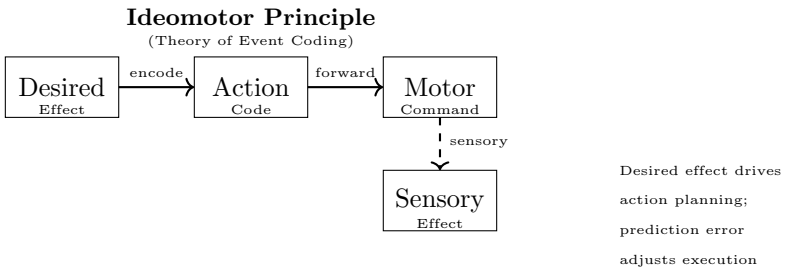


Figure 5.1: The ideomotor principle: actions are planned in terms of their desired sensory effects rather than muscle commands. The forward model predicts the sensory consequence, and prediction error corrects the action.

5.2 The Predictive Processing Framework

Modern neuroscience interprets the brain as a **prediction machine** (Clark, 2013; Hohwy, 2013):

Key Insight 5.1. The Brain as a Prediction Machine

The brain continuously generates predictions about incoming sensory data. Neural processing is dominated by **prediction error minimization**:

$$\varepsilon_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_k = \boldsymbol{y}_k - h(\hat{\boldsymbol{x}}_k), \tag{5.1}$$

where $\hat{\boldsymbol{y}}_k$ is the predicted observation and $\boldsymbol{y}_k$ is the actual observation. This error drives updates to the internal model.
In the language of Volume I, Chapter 6: this is exactly the **Kalman filter innovation**. The brain implements Bayesian state estimation.

5.2.1 Active Inference

Karl Friston's active inference framework (Friston, 2010) extends predictive processing to action: instead of only updating beliefs to match observations, the agent acts to make observations match predictions. This formalizes the ideomotor principle in terms of Free Energy minimization:

$$\boldsymbol{u}^* = \arg\min_{\boldsymbol{u}} \mathbb{E}\left[\sum_{k=0}^{T} \|\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k\|^2_{R^{-1}} + \|\boldsymbol{u}_k\|^2_Q\right], \tag{5.2}$$

which is formally equivalent to model predictive control (Volume I, Chapter 5).

5.3 Connection to Control Theory

Neuroscience Term	**Control Theory Term**
Forward model	State transition matrix / predictor
Prediction error	Innovation (Kalman filter)
Efference copy	Feedforward control signal
Active inference	Model predictive control
Prior beliefs	State constraints / initial conditions
Precision weighting	Kalman gain / observation weights
Free energy	Variational cost functional

Exercises

1. **Conceptual.** Explain how the ideomotor principle relates to trajectory-centric control (Volume II). When a golfer imagines impact, what "prediction" are they generating?

2. **Kalman Connection.** Write out the Kalman filter equations and identify which term corresponds to the prediction error in the predictive processing framework.

3. **(Programming.)** Implement an active inference agent that controls a pendulum by minimizing prediction error rather than tracking a reference.

Chapter 6

Internal Models: Forward and Inverse

Draft Notice. This chapter is under active development and contains only a structural outline with preliminary material. Full exposition, worked examples, proofs, and exercises will be added in future editions.

To control is to predict. To predict is to have a model. To have a model is to have learned.

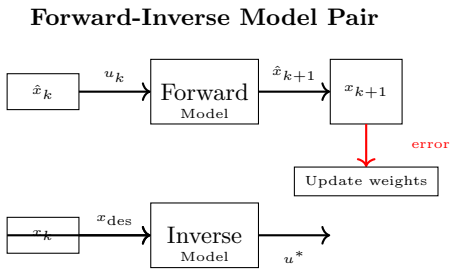


Figure 6.1: Paired forward and inverse models: forward model predicts state evolution from command; inverse model computes command to reach desired state. Learning updates both based on sensory prediction error.

6.1 Forward Models

A forward model predicts the sensory consequences of a motor command:

$$\hat{\boldsymbol{x}}_{k+1} = f(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k), \tag{6.1}$$

which is exactly the state propagation used in Volume I.

6.1.1 The MOSAIC Architecture

The MOSAIC model (Modular Selection and Identification for Control) [40] proposes that the brain maintains multiple paired forward-inverse models, each tuned to different dynamics or contexts. The claim that a model is acquired *per tool* rather than adjusted in place has direct imaging support: cerebellar activity acquired while learning a novel visuomotor tool persists as a tool-specific representation, separable from the activity driven by the movement itself [21].

```
import numpy as np
from typing import List, Tuple

class ForwardInversePair:
    """A paired forward-inverse model for one context."""
    def __init__(self, A: np.ndarray, B: np.ndarray):
        self.A = A  # State transition
        self.B = B  # Input matrix

    def forward_predict(self, x: np.ndarray,
                        u: np.ndarray) -> np.ndarray:
        """Predict next state."""
        return self.A @ x + self.B @ u

    def inverse(self, x: np.ndarray,
                x_desired: np.ndarray) -> np.ndarray:
        """Compute control to reach desired state."""
        return np.linalg.lstsq(self.B,
                               x_desired - self.A @ x,
                               rcond=None)[0]


class MOSAIC:
    """Multiple paired forward-inverse model architecture."""
    def __init__(self, models: List[ForwardInversePair]):
        self.models = models
        self.n_models = len(models)
        self.responsibilities = np.ones(self.n_models) / self.n_models

    def update_responsibilities(self, x: np.ndarray,
                                u: np.ndarray,
                                x_next: np.ndarray) -> None:
        """Update model responsibilities based on prediction errors."""
        errors = np.zeros(self.n_models)
        for i, model in enumerate(self.models):
            prediction = model.forward_predict(x, u)
            errors[i] = np.sum((x_next - prediction)**2)

        # Softmax weighting (lower error = higher responsibility)
        log_resp = -0.5 * errors
        log_resp -= np.max(log_resp)  # numerical stability
        self.responsibilities = np.exp(log_resp)
        self.responsibilities /= np.sum(self.responsibilities)

    def predict(self, x: np.ndarray,
                u: np.ndarray) -> np.ndarray:
        """Weighted prediction across all models."""
        prediction = np.zeros_like(x)
        for i, model in enumerate(self.models):
            prediction += self.responsibilities[i] * \
                          model.forward_predict(x, u)
```

```
        return prediction

    def control(self, x: np.ndarray,
                x_desired: np.ndarray) -> np.ndarray:
        """Weighted inverse model output."""
        u = np.zeros(self.models[0].B.shape[1])
        for i, model in enumerate(self.models):
            u += self.responsibilities[i] * \
                 model.inverse(x, x_desired)
        return u
```

Listing 6.1: MOSAIC-style multiple model architecture.

6.2 Inverse Models

An inverse model computes the motor command required to achieve a desired state:

$$\boldsymbol{u}_k = f^{-1}(\boldsymbol{x}_k, \boldsymbol{x}^{\text{des}}_{k+1}), \tag{6.2}$$

which corresponds to the feedforward control of Volume II. How such a model could be *acquired* rather than assumed is the contribution of feedback-error learning: the output of a crude feedback controller is used as the training signal for the inverse model, so the feedback loop teaches the feedforward path and then falls silent [24]. Note that this is the inverse-model result; the forward-model evidence is separate, and conflating the two is easy because both trace to the same group.

6.3 Smith Predictor for Neural Delays

The sensorimotor system faces finite neural and sensing delays. The Smith predictor architecture uses a forward model to compensate:

$$\hat{\boldsymbol{x}}_{\text{current}} = \hat{\boldsymbol{x}}(t - \Delta t) + \int_{t-\Delta t}^{t} f(\hat{\boldsymbol{x}}, \boldsymbol{u})\, \mathrm{d}\tau, \tag{6.3}$$

where Δt is the sensory delay. The controller operates on the *predicted current state* rather than the delayed sensory measurement.

Exercises

1. **MOSAIC.** Using the provided code, simulate a system that switches between two dynamics (e.g., carrying an empty cup vs. a full cup). Show how the responsibilities shift.

2. **Smith Predictor.** Implement a Smith predictor for a simple pendulum with a user-chosen sensory delay parameter. Compare performance with and without prediction.

3. **(Research.)** Summarize evidence for multiple internal models in human motor control, citing at least three experimental studies.

Chapter 7

Passive Distributed Control

Draft Notice. This chapter is under active development and contains only a structural outline with preliminary material. Full exposition, worked examples, proofs, and exercises will be added in future editions.

> The most efficient movement is one where the body's own mechanics do most of the work, and the nervous system merely guides.
>
> — Adapted from Frans Bosch

7.1 The Frans Bosch Framework

Frans Bosch's work on strength training and coordination [6] provides a practitioner-oriented framework for understanding how elite athletes organize movement. His key insights, translated into the language of this series:

Key Insight 7.1. Self-Organization in Complex Movement

Bosch argues that complex movements are not centrally programmed but **self-organize** through the interaction of:

1. **Task constraints**: the goal shapes the solution space
2. **Organismic constraints**: the body's anatomy and current state
3. **Environmental constraints**: gravity, ground, implements

The role of the nervous system is not to specify every muscle activation but to set the *initial conditions* and *constraints* that allow self-organization to produce the desired movement.
In the language of Volume I: the CNS designs the *drift field* $f(\boldsymbol{x})$ of the system (through postural configuration and muscle tone) and lets the natural dynamics do the work.

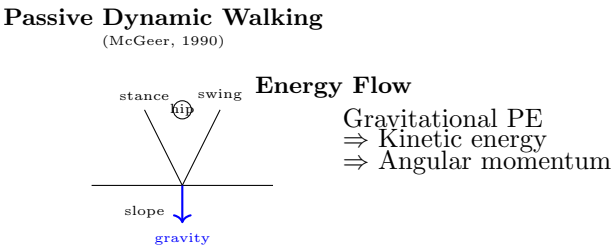


Figure 7.1: Passive dynamic walking: a compass gait on a gentle slope exhibits stable walking cycles with zero control input. Gravity and inertia provide all energy; the walker merely guides the descent.

7.2 Passive Dynamic Walking

The most striking example of passive dynamics is passive dynamic walking (McGeer, 1990): a simple mechanical biped with knees, feet, and hip (no motors) walks down a gentle slope in stable limit cycles with zero active control. This demonstrates that complex coordinated movement can emerge from mechanical design and gravity alone:

```
import numpy as np
from scipy.integrate import solve_ivp

def passive_walker_dynamics(
    t: float,
    state: np.ndarray,
    M: float = 10.0,
    m: float = 5.0,
    l: float = 1.0,
    g: float = 9.81,
    slope: float = 0.05
) -> np.ndarray:
    """Dynamics of a 2-DOF passive walker (compass gait).

    Parameters
    ----------
    t : float
        Time.
    state : np.ndarray
        [theta_stance, theta_swing, d_theta_stance, d_theta_swing]
    M : float
        Hip mass (kg).
    m : float
        Leg mass at foot (kg).
    g : float
        Gravitational acceleration (m/s^2).
    slope : float
        Ground slope (radians).

    Returns
    -------
    np.ndarray
        State derivatives.
    """
    th_st, th_sw, dth_st, dth_sw = state
```

```
    # Mass matrix
    M11 = (M + m) * l**2
    M12 = -m * l**2 * np.cos(th_st - th_sw)
    M21 = M12
    M22 = m * l**2

    # Gravity and Coriolis
    C1 = -m * l**2 * dth_sw**2 * np.sin(th_st - th_sw) \
         - (M + m) * g * l * np.sin(th_st - slope)
    C2 = m * l**2 * dth_st**2 * np.sin(th_st - th_sw) \
         + m * g * l * np.sin(th_sw - slope)

    # Solve M * ddq = C (NO control input!)
    det = M11 * M22 - M12 * M21
    ddth_st = (M22 * C1 - M12 * C2) / det
    ddth_sw = (M11 * C2 - M21 * C1) / det

    return np.array([dth_st, dth_sw, ddth_st, ddth_sw])


# Simulate one step of passive walking
state0 = np.array([0.2, -0.4, -0.5, 1.0])
sol = solve_ivp(passive_walker_dynamics, [0, 1.5], state0,
                max_step=0.001, events=None)
print(f"Passive walker: simulated {sol.t[-1]:.2f}s "
      f"with {len(sol.t)} timesteps")
print(f"Final stance angle: {sol.y[0, -1]:.3f} rad")
print("(No actuators, no controller --- just gravity and mechanics)")
```

Listing 7.1: Passive dynamic walker simulation.

7.3 Passive Dynamics in the Golf Swing

The golf swing exemplifies passive dynamics exploitation:

1. **Backswing**: energy stored in elastic tissues (muscles, tendons, fascia)
2. **Transition**: elastic recoil initiates the downswing passively
3. **Wrist release**: centrifugal force passively uncocks the wrists (the double pendulum effect from Volume II, Chapter 5)
4. **Follow-through**: passive deceleration through tissue compliance

In the ZTCF framework of Volume I, Chapter 7: the drift contribution $f(\boldsymbol{x})$ accounts for $\sim$60% of the clubhead speed. The control contribution $G(\boldsymbol{x})\boldsymbol{u}$ provides the remaining $\sim$40%.

7.4 Bernstein's Stage 3: Exploiting DOF

The connection to Bernstein's learning framework (Chapter 1) is direct:

- The expert athlete has reached Stage 3 — exploiting degrees of freedom

- They have learned to configure the body so that passive dynamics *automatically* produce the desired motion
- The control problem is simplified: instead of commanding every muscle, set up the initial conditions and let physics do the work

Exercises

1. **Passive Walking.** Using the provided code, find the set of initial conditions that produce a stable limit cycle (periodic gait).
2. **Energy Analysis.** For the passive walker, plot kinetic and potential energy over one stride. Where does the energy come from?
3. **Wrist Release.** Model the wrist release as a passive double pendulum. For what initial angular velocity of the arm does the club reach maximum speed at the bottom?
4. **(Programming.)** Add a small actuator to the passive walker and design a minimal controller that stabilizes the gait on level ground.

Chapter 8

The Interplay of Biology and Dynamics of Nonlinear Systems

> Biological systems are not noisy versions of engineering machines. They are nonlinear dynamical systems where the constraints themselves— tissue properties, neural architecture, self-organization— shape the landscape on which movement emerges.
>
> — Frans Bosch

8.1 Introduction: Why Biology Matters for Dynamics

The rigid-body dynamics of Chapters 1–6 provide the mechanical skeleton of motor control. Yet between these skeletal mechanics and actual biological movement lies a gap. Muscles are not ideal actuators with instantaneous force production. Neural commands do not arise from magical inverse models. Movement patterns do not emerge from centralized computation.

This chapter bridges that gap by reconceptualizing the dynamics problem: biology does not merely *add complications* to rigid-body mechanics. Rather, biological constraints fundamentally reshape the dynamical landscape— the space of possible motions, the cost of different trajectories, the attractors that self-organize without active control.

Key Insight 8.1. Three Levels of Biological Constraint

Biological systems intersect with dynamics at three hierarchical levels:

1. **Tissue Properties** (Level 1): muscle visco-elasticity, tendon compliance, fascial stiffness. These modify the drift field $f(\boldsymbol{x})$ itself, changing the natural dynamics without any neural command.
2. **Neural Architecture** (Level 2): the structure of afferent and efferent connections (spinal reflexes, CPG circuits, feedback gains). These shape the control structure $G(\boldsymbol{x})$ and the feedback laws available to the nervous system.
3. **Self-Organization** (Level 3): the spontaneous emergence of coordination patterns from the interaction of Levels 1 and 2. Synergies, attractors, and bifurcations arise not from explicit computation but from nonlinear dynamics.

The key insight: **each level is incomplete without the others.** Muscles alone produce passive oscillations but no goal-directed movement. Neural circuits alone cannot compute fast enough for real-time control. Self-organization alone cannot solve the task without constraints that channel it.

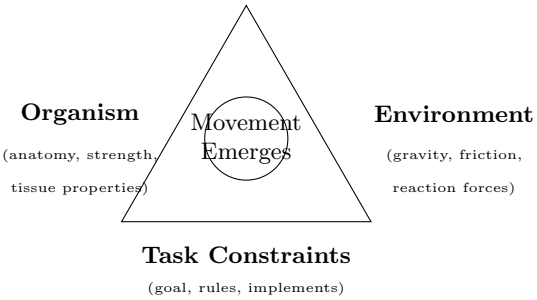
8.1.1 The Bernstein Problem Revisited

In Volume I, Chapter 1, we introduced Bernstein's problem: a 7-DOF arm with hundreds of muscles cannot be controlled by specifying each muscle independently. The dimensional mismatch (task variables vs. available DOFs) is not a flaw to be corrected by a clever inverse model. Instead, Bernstein argued, it is a *design feature*: redundancy allows the nervous system to solve the problem in lower dimensions through synergies.

From a dynamical systems perspective, this means the control system does not command the full state $\boldsymbol{x} \in \mathbb{R}^{2n}$ (positions and velocities of all joints). Instead, it commands a lower-dimensional attractor space, and the biological constraints (muscle properties, spinal circuits) handle the details of execution.

8.1.2 Bosch's Constraints-Led Approach

Frans Bosch's framework [7, 6] provides the bridge between biomechanics and neural organization. His central claim: movement emerges from the interaction of three constraint systems:



In dynamical systems language: the attractor landscape of the controlled system is determined by the *intersection* of these three constraint sets. The role of the nervous system is to **set the initial conditions and modulate the constraints** so that the intersection produces the desired movement.

This is not inverse kinematics. This is not trajectory tracking. This is **constraint modulation**: the CNS shapes the landscape so that self-organization produces the goal.

8.2 Visco-Elastic Properties and Intrinsic Dynamics

8.2.1 Muscle as a Nonlinear Spring-Damper

The Hill model (1938) remains the standard description of muscle force production:

Definition 8.1. The Hill Muscle Model

A muscle generates force F based on three components:

- **Contractile element (CE)**: force depends on length and velocity
- **Series elastic element (SEE)**: tendon and aponeurosis compliance
- **Parallel elastic element (PEE)**: passive muscle tissue stiffness

The standard formulation [16]:

$$F_{\text{total}} = F_{\text{CE}}(l_{\text{CE}}, \dot{l}_{\text{CE}}, a) + F_{\text{PEE}}(l_{\text{muscle}}) \tag{8.1}$$

where:

- l_{CE} is contractile element length
- $\dot{l}_{\text{CE}}$ is contractile element velocity
- $a \in [0, 1]$ is neural activation
- l_{muscle} is total muscle length
- F_{PEE} is the passive elastic force

The force-length (F/L) curve $F_{\text{CE}}(l_{\text{CE}}, a)$ is nonlinear, peaked around optimal length l_0. The force-velocity (F/V) curve shows that muscles generate less force when shortening quickly—a fundamental asymmetry that shapes coordinated movement.

The nonlinearity is crucial. A muscle at optimal length with maximal activation produces force $F_{\max}$. But the same muscle at sub-optimal length produces significantly less force, even with full activation:

$$F_{\text{CE}}(l, a) = a \cdot f_L(l) \cdot f_V(\dot{l}) \cdot F_{\max}, \tag{8.2}$$

where $f_L(l)$ (the F/L curve) typically ranges from 0 to 1 as length varies over $\pm 30\%$ from l_0, and $f_V(\dot{l})$ (the F/V curve) is:

$$f_V(\dot{l}) = \begin{cases} \frac{1 - b\dot{l}/v_{\max}}{1 + c\dot{l}/v_{\max}} & \text{if } \dot{l} < 0 \text{ (shortening)} \\ 1 & \text{if } \dot{l} = 0 \\ \frac{F_{\max}/P_{\max} - \dot{l}/v_{\max}}{1/c + \dot{l}/v_{\max}} & \text{if } \dot{l} > 0 \text{ (lengthening)} \end{cases} \tag{8.3}$$

The constants b and c are empirically fit (typical values: $b \approx 0.3$, $c \approx 0.6$).

8.2.2 The Preflex: Zero-Delay Mechanical Feedback

Muscles are not passive springs. When a muscle is stretched, the mechanical properties themselves generate a restoring force *without any neural signal*:

Key Insight 8.2. The Preflex Concept

The **preflex** (Loeb, 1995; Bosch, 2020) is the mechanical response of muscle and tendon to perturbation, arising purely from visco-elastic properties— zero neural delay.
When a muscle of length l and velocity $\dot{l}$ experiences a sudden stretch (e.g., from a disturbance), the force increase is immediate:

$$\Delta F = K_{\text{muscle}}(a) \cdot \Delta l + D_{\text{muscle}}(a) \cdot \Delta \dot{l}, \tag{8.4}$$

where $K_{\text{muscle}}(a)$ is activation-dependent stiffness and $D_{\text{muscle}}(a)$ is damping. For a muscle under steady activation, a 1 cm stretch produces a restoring force within milliseconds—before any spinal reflex can transmit a signal (which requires ≥ 20 ms). This is critically important: small perturbations are automatically stabilized by the muscle's own mechanics, without involving the nervous system.

The biological consequence is remarkable: even an anesthetized animal (no spinal reflexes) can maintain postural stability for brief durations through the preflex alone [27].

8.2.3 Tendon Elasticity and the Catapult Mechanism

The series elastic element (SEE) is not merely a passive compliance to be overcome. In ballistic movements (throwing, striking, jumping), the SEE becomes a **catapult**: elastic energy stored during a loading phase is released explosively during the ballistic phase.

Consider a simple model of jumping:

1. **Loading phase**: muscles contract while the body is still in contact with the ground. The SEE (tendon and aponeurosis) stores elastic energy.

2. **Ballistic phase**: muscles relax slightly, but the stored elastic energy accelerates the body upward. Peak power output exceeds what the muscles could produce alone.

For human jumping, stored elastic energy contributes ∼50% of the mechanical energy for the jump. The SEE acts as a mechanical transformer: slow muscle contraction (which can be powerful) becomes fast skeletal motion (which can be coordinated precisely).

In the language of nonlinear dynamics, the SEE changes the **time scaling** of the system: the equation of motion for the body is not directly the muscle force, but the force transmitted through an elastic element:

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}}) + G(\boldsymbol{q}) = J^T(\boldsymbol{q}) F_{\text{SEE}}, \tag{8.5}$$

where $F_{\text{SEE}} = K_{\text{tendon}}(l_{\text{tendon}} - l_{\text{muscle}})$ depends on the elastic deformation of the tendon.

8.2.4 How Visco-Elasticity Modifies the Drift Field

Putting this together: the intrinsic tissue properties change the effective drift field of the mechanical system.

Definition 8.2. The Modified Drift Field

In the absence of neural activation ($u = 0$), the uncontrolled dynamics with visco-elastic properties is:

$$\dot{\boldsymbol{x}} = f_{\text{bio}}(\boldsymbol{x}) = \begin{pmatrix} \dot{\boldsymbol{q}} \\ M^{-1}(\boldsymbol{q}) \left[-C(\boldsymbol{q}, \dot{\boldsymbol{q}}) - G(\boldsymbol{q}) + J^T(\boldsymbol{q}) F_{\text{visco}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \right] \end{pmatrix}, \tag{8.6}$$

where $F_{\text{visco}}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ includes:

- Muscle visco-elasticity: $F_{\text{damping}} = D(\boldsymbol{q}, a) \cdot \dot{\boldsymbol{q}}$
- Tendon elasticity: $F_{\text{tendon}} = K_{\text{SEE}}(l_{\text{SE}})$
- Passive tissue stiffness: added to the effective stiffness matrix

This modified drift field $f_{\text{bio}}(\boldsymbol{x})$ has attractors and basins of attraction that differ fundamentally from the gravity-only case.

The practical consequence: the CNS does not command every muscle to produce every motion. Instead, it exploits the intrinsic dynamics. For many movements, activating a few muscles is sufficient; the tissue properties handle the details of force distribution and stabilization.

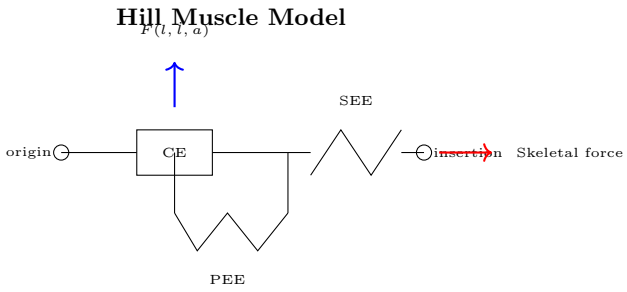


Figure 8.1: The Hill muscle model: contractile element (CE) produces force based on length, velocity, and activation. Series elastic element (SEE) couples the muscle to the skeleton. Parallel elastic element (PEE) provides passive restoring force. The SEE introduces a time delay and filtering effect on muscle force transmission.

8.3 The Attractor-Fluctuation Framework

8.3.1 Formal Definitions: Attractors in State Space

From Chapter 2, recall that a dynamical system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ has an attractor if there exists a set $A \subset \mathbb{R}^{2n}$ such that:

Definition 8.3. Attractor (Formal)

A set A is an **attractor** if:

1. **Invariance**: if $\boldsymbol{x}(0) \in A$, then $\boldsymbol{x}(t) \in A$ for all $t \geq 0$
2. **Attraction**: there exists an open neighborhood $U \supset A$ such that for any $\boldsymbol{x}_0 \in U$, $\lim_{t\to\infty} \operatorname{dist}(\boldsymbol{x}(t), A) = 0$
3. **Minimality**: no proper subset of A satisfies properties 1 and 2

Common attractors in motor control:

- **Fixed points**: equilibrium postures (sitting, standing)
- **Limit cycles**: periodic motions (walking, running, swimming)
- **Tori**: quasi-periodic motions (multi-frequency oscillations)
- **Chaos**: deterministic but unpredictable trajectories (rare in motor control)

8.3.2 Lyapunov Stability and Movement Patterns

An attractor is **stable** if nearby trajectories converge to it. The Lyapunov exponent λ quantifies the convergence rate:

$$\operatorname{dist}(\boldsymbol{x}_1(t), \boldsymbol{x}_2(t)) \approx e^{\lambda t} \cdot \operatorname{dist}(\boldsymbol{x}_1(0), \boldsymbol{x}_2(0)), \tag{8.7}$$

where $\lambda < 0$ for stable attractors and $\lambda > 0$ for unstable repellors.

For a movement pattern to be **self-stabilizing**, small perturbations must converge back to the nominal trajectory. This requires negative Lyapunov exponents at the attractor.

Key Insight 8.3. Why Attractors Are Biologically Efficient

An attractor requires no active feedback control to maintain stability—perturbations are automatically corrected by the dynamics itself. This is highly efficient:

- **No feedback loop delay**: a passive walking gait stays stable without sensing or neural computation, purely through mechanical feedback
- **Robustness**: small variations in parameters (muscle activation, limb stiffness) do not destroy the motion—the attractor adjusts automatically
- **Metabolic economy**: energy goes into maintaining the attractor, not correcting for instability

Conversely, a motion that is **not** an attractor requires continuous active control. Every instant, the nervous system must sense and correct deviations. This is metabolically expensive and requires precise sensing.

Bernstein's insight about redundancy connects here: the nervous system should exploit (or create) attractors so that most of the motion stabilizes itself. The nervous

system then uses its limited computation budget to modulate the attractor—change the location, size, or stability—rather than controlling every variable independently.

8.3.3 Bosch's Attractor-Fluctuation Landscape

Bosch's key contribution is to classify all movements on a spectrum from attractors (low energy cost, self-stabilizing) to fluctuations (require active control):

Definition 8.4. Attractor vs. Fluctuation

In a movement task:

- **Attractors** are patterns that emerge naturally from the system's intrinsic dynamics. They have low energy cost and require minimal active control. Examples: the natural gait of bipedal walking, the postural stability of standing, the rhythmic oscillation of arm swinging.
- **Fluctuations** are deviations from the attractor landscape that require active neural control to maintain. They have high energy cost. Examples: standing on one leg (vs. standing on two), walking at a non-preferred cadence, precise reaching under time pressure.

In the framework of Bosch and constraints-led learning: expert movement is the exploitation of attractors. The learner's task is to discover which configurations of the body (posture, stiffness, muscle tone) produce attractors that satisfy the task goal.

8.3.4 Bifurcation Theory: Phase Transitions in Movement

As a control parameter changes, attractors can appear, disappear, or lose stability— a phenomenon called **bifurcation**.

The canonical example is the human gait transition. At low speeds, humans walk; at high speeds, they run. In between (typically 1.5–2.5 m/s), a transition occurs: walking becomes unstable and running becomes stable. Biomechanical models reveal this is a bifurcation in the cost function (metabolic energy):

$$C_{\text{walk}}(v) = a + b \cdot v^2, \quad C_{\text{run}}(v) = c + d \cdot v^2 + e, \tag{8.8}$$

where for $v < v^*$, walking is cheaper, but for $v > v^*$, running is cheaper. The transition speed v^* depends on leg length, body weight, and gravity.

In human subjects, the observed gait transition speed is close to the predicted optimal value (Margaria, 1976; Minetti et al., 1994). This suggests the nervous system selects the gait with lower energy cost—another example of exploiting natural dynamics.

8.3.5 The HKB Model: Self-Organization in Bimanual Coordination

The Haken-Kelso-Bunz (HKB) model (Haken et al., 1985; Kelso, 1995) is the canonical example of self-organization in motor control. Two limbs oscillate in either in-phase (both forward) or anti-phase (mirror) coordination, and the transition between them is a bifurcation:

$$\ddot{\phi} + \gamma\dot{\phi} + A(\omega)\sin(\phi) + 2B(\omega)\sin(2\phi) = Q\sin(\omega t), \tag{8.9}$$

where ϕ is the phase difference between the two limbs, ω is the driving frequency, and A, B are control parameters that depend on movement speed.

The key observation: **no central command specifies the coordination mode**. Instead, as movement speed increases, the in-phase mode becomes unstable and the system spontaneously transitions to anti-phase. This is a dynamic phase transition (bifurcation), not a decision.

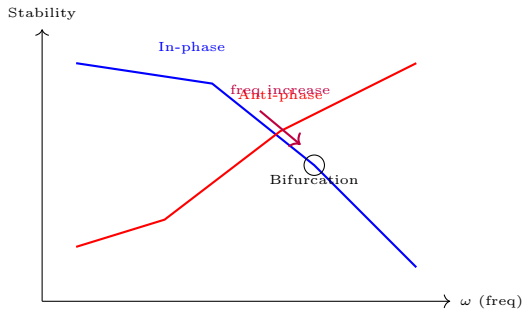


Figure 8.2: HKB bifurcation: In-phase coordination is stable at low frequencies but becomes unstable above a critical frequency (bifurcation point). Anti-phase coordination takes over. The transition is automatic—no neural command needed. As frequency increases further, the stability of anti-phase increases.

The HKB model demonstrates that complex, coordinated motor behavior can emerge from simple nonlinear dynamics. There is no explicit "coordination module" in the brain that decides when to switch from in-phase to anti-phase. Instead, the dynamics itself undergoes a phase transition, and behavior changes as a consequence.

8.4 Synergies and the Uncontrolled Manifold

8.4.1 Bernstein's Synergies: Reducing Dimensionality

A human arm has ∼40 muscles and $\geq$ 7 DOFs. Yet the nervous system often activates multiple muscles in synchrony—**synergies**.

Synergies reduce the effective dimensionality of control. Instead of commanding 40 independent muscle activations, the nervous system commands, say, 5 synergy amplitudes.

Definition 8.5. Muscle Synergy

A **muscle synergy** is a time-invariant weighting vector $\mathbf{w}_i \in \mathbb{R}^p$ (where p is the number of muscles) such that the total muscle activation is:

$$\mathbf{u}(t) = \sum_{i=1}^{k} c_i(t)\mathbf{w}_i = W\mathbf{c}(t), \tag{8.10}$$

where $c_i(t)$ are synergy activations (time-varying, $k \ll p$), and W is the synergy matrix ($p \times k$).
The variance explained is:

$$\text{VAF} = 1 - \frac{\|\mathbf{u} - W\hat{\mathbf{c}}\|^2}{\|\mathbf{u}\|^2} \times 100\%, \tag{8.11}$$

where $\hat{\mathbf{c}}$ is the least-squares fit of synergy activations.

Empirical evidence is striking:

- Frog leg motor output: $\sim$5 synergies explain $> 85\%$ of variance across 7 behaviors
- Human reaching: $\sim$4–6 synergies explain $> 90\%$ of variance across 15 reach directions
- Human walking: $\sim$4 synergies explain $> 80\%$ of variance across speeds and slopes

8.4.2 The Uncontrolled Manifold: Separating Task-Relevant from Task-Irrelevant Variation

Synergies often align with the **uncontrolled manifold** (UCM) —the set of all configurations that produce the same task-relevant output.

Suppose the task is to place your hand at position $\mathbf{p} = (p_x, p_y, p_z)$. Your arm's forward kinematics is:

$$\mathbf{p} = \mathbf{h}(\boldsymbol{q}), \tag{8.12}$$

where $\boldsymbol{q} \in \mathbb{R}^7$ is the joint configuration (shoulder and elbow angles). The Jacobian is:

$$J(\boldsymbol{q}) = \frac{\partial \mathbf{h}}{\partial \boldsymbol{q}} \in \mathbb{R}^{3\times 7}, \tag{8.13}$$

The UCM is the null space of J:

Definition 8.6. Uncontrolled Manifold

The uncontrolled manifold for task $\mathbf{p} = \mathbf{h}(\boldsymbol{q})$ is:

$$\text{UCM} = \{\boldsymbol{q} : J(\boldsymbol{q})\Delta\boldsymbol{q} = 0\}, \tag{8.14}$$

i.e., the set of all infinitesimal changes in configuration that do not change the task variable. Movements within the UCM are **task-irrelevant**— they do not affect the goal. Movements orthogonal to the UCM are **task-relevant**—they move toward or away from the goal.

Variance decomposition [32, 26]:

- V_{UCM}: variance within the UCM (task-irrelevant)
- V_{ORT}: variance orthogonal to the UCM (task-relevant)

Skilled performance is characterized by $V_{\text{UCM}} \gg V_{\text{ORT}}$: the nervous system allows large variability in joint configurations (exploiting redundancy) but constrains variability in the task outcome.

The practical insight: synergies align with the UCM. They activate muscles in combinations that preserve the task variable while allowing flexibility in execution. This is Bernstein's solution to the degrees-of-freedom problem: not all DOFs are controlled independently, but rather organized to keep the task on track while allowing variation in the details.

8.4.3 Connection to the Affine Framework

In the control-affine framework of Volume I:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + \sum_{i=1}^{m} u_i g_i(\boldsymbol{x}), \tag{8.15}$$

the control input space is spanned by the vector fields $\{g_1, \ldots, g_m\}$. If there are m muscles (or m synergies), there are (at most) m independent input directions in state space.

Synergies reduce this further: if k synergies are sufficient, the effective control input space has dimension k. The remaining $m - k$ dimensions are **uncontrollable**— perturbations in these directions cannot be corrected by any combination of inputs.

The uncontrollable subspace must then be **unobservable in the task variables**. This is exactly the UCM: movement within the UCM does not change the task, and therefore does not need to be controlled.

This is the deep connection between Bernstein's synergies and control theory: synergies are not just empirical regularities. They are the **structure of controllability** imposed by biological constraints.

8.5 Impedance Control and the Stiffness-Damping Trade-off

8.5.1 Impedance Control Framework

Instead of tracking a trajectory exactly, an alternative control strategy is to regulate the mechanical impedance of the system:

Definition 8.7. Mechanical Impedance

The **impedance** is the relationship between applied force F and resulting displacement x. In the Laplace domain:

$$Z(s) = \frac{F(s)}{X(s)} = Ms^2 + Ds + K, \tag{8.16}$$

where:

- M is inertia (mass)
- D is damping (viscous friction)
- K is stiffness (elastic restoring force)

Equivalently, in the time domain, the equation of motion under external force F_{ext} is:

$$M\ddot{x} + D\dot{x} + Kx = F_{\text{ext}}. \tag{8.17}$$

A stiff system (K large) resists displacement; a compliant system (K small) deflects easily. Similarly, a damped system dissipates energy; an underdamped system oscillates.

8.5.2 Co-Contraction as Impedance Modulation

Muscles work in antagonistic pairs: for every flexor, there is an extensor. When both are activated simultaneously (co-contraction), the joint becomes stiffer:

$$K = K_{\text{flexor}}(a_f) + K_{\text{extensor}}(a_e), \tag{8.18}$$

where a_f and a_e are activation levels. By varying the co-contraction level, the nervous system can tune the stiffness of the joint without changing the mean torque.

Co-contraction is observed when:

- **Making contact with an object**: the nervous system increases stiffness to reduce instability upon impact
- **Reaching to an uncertain target**: increased stiffness reduces sensitivity to execution error
- **Reacting to an unpredictable disturbance**: higher stiffness speeds up the response time

The trade-off: increased stiffness requires metabolic energy (both muscles working against each other) and reduces flexibility (a stiff joint cannot follow a changing target as easily).

8.5.3 Optimal Impedance

The question becomes: what stiffness and damping are optimal?

For a reach to a fixed target under uncertainty, increasing stiffness reduces the cost of endpoint error (because forces are corrected faster) but increases the cost of actuation. There is a trade-off. Optimal impedance balances these costs:

Definition 8.8. Optimal Impedance

For a reach task with cost:

$$J = \mathbb{E}[\|x(T) - x_{\text{target}}\|^2] + \int_0^T c_u(\boldsymbol{u}(t))\, dt, \tag{8.19}$$

where the first term is endpoint error and the second is actuation cost, the optimal impedance satisfies:

$$K^* = \arg\min_K J(K). \tag{8.20}$$

For Gaussian noise and quadratic costs, the optimal stiffness is proportional to the signal-to-noise ratio: if the target is uncertain, stiffness increases; if actuation is costly, stiffness decreases [18, 11].

Empirical evidence supports this theory: human subjects reaching to uncertain targets increase arm stiffness proportionally to the target uncertainty [18].

8.5.4 Connection to Robust Control Theory

In modern control theory, the H_∞ norm characterizes the worst-case response to disturbances. A control law is robust if its H_∞ norm is small—meaning even large disturbances produce bounded output.

For a mechanical system with impedance $Z(s) = Ms^2 + Ds + K$, the H_∞ norm depends on the relative magnitudes of M, D, and K. Under-damping (D too small) leads to resonance and poor disturbance rejection. Over-damping (D too large) leads to sluggishness. Optimal impedance tuning achieves robust control with minimal energy.

The nervous system, through co-contraction and muscle activation patterns, is continuously tuning impedance to match the robustness requirements of the current task.

8.5.5 The Metabolic Cost of Impedance

Increasing stiffness requires sustained muscular effort. In biomechanics, the metabolic cost of isometric muscle contraction (holding a position against force) is roughly proportional to the sustained force:

$$\text{Metabolic cost} \propto \int K\, dt, \tag{8.21}$$

This means maximum stiffness is *never* optimal—the metabolic cost becomes prohibitive. Skilled movement balances impedance (robustness against perturbations) against metabolic economy.

In expert athletes, impedance tuning is often implicit and adaptive: as the environment changes, stiffness adjusts automatically without conscious thought. This is an example of motor learning as the discovery of optimal impedance landscapes for different tasks.

8.6 Self-Organization in Biological Systems

8.6.1 Dynamic Systems Theory Applied to Motor Control

The modern framework for understanding self-organization in biological movement comes from dynamic systems theory, pioneered by Kelso, Turvey, and Schöner.

The key idea: behavior emerges from the interaction of multiple coupled nonlinear oscillators (neurons, muscles, limbs) without explicit central command. The nervous system does not compute trajectories; it shapes the dynamics so that self-organization produces the goal.

Key Insight 8.4. The Paradigm Shift

Before (computational/representational view): the brain computes an optimal trajectory and sends it to muscles.
After (dynamical systems view): the brain modulates the control parameters (stiffness, damping, attractors) and lets the coupled dynamics produce the motion.
The difference is significant:

- **Computation**: trajectory requires solving an inverse problem (expensive); parameter modulation is local and simple
- **Adaptation**: if the environment changes, the brain adapts parameters, and the new dynamics automatically produces a new motion
- **Robustness**: self-organized attractors are inherently stable to perturbations; computed trajectories must be continuously corrected

8.6.2 Order Parameters and Control Parameters

In a complex system with many degrees of freedom, a small number of **order parameters** capture the essential behavior:

Definition 8.9. Order Parameter and Control Parameter

An **order parameter** ϕ is a collective variable that captures the macro-scale behavior of a system. For bimanual coordination, ϕ is the phase difference between the hands.
A **control parameter** λ is an external or internal variable that the experimenter (or the nervous system) can adjust. For bimanual coordination, λ is the movement frequency.
The dynamics of the order parameter is:

$$\tau\dot{\phi} = f(\phi, \lambda) + \text{noise}, \tag{8.22}$$

where τ is a time constant and f captures the interaction between the coupled oscillators.
The signature of self-organization is a **phase transition**: as λ changes, the fixed points of $f(\phi, \lambda)$ change qualitatively (bifurcation), and behavior undergoes a spon-

taneous transition.

The HKB model (Section 4.5) is exactly this framework: the order parameter is phase difference, the control parameter is oscillation frequency, and the dynamics shows a bifurcation as frequency increases.

8.6.3 Biotensegrity: The Body as a Pre-Stressed Structure

Ingber (1998) and Levin (2002) proposed that the body is not a system of rigid links connected by passive joints. Rather, it is a **tensegrity structure**: a framework where compression (bones) and tension (muscles, tendons, fascia) are distributed throughout, creating a pre-stressed network.

Definition 8.10. Tensegrity

A **tensegrity structure** is a system where:

- Compression elements (struts) resist squeezing
- Tension elements (cables) resist pulling
- The system is pre-stressed: internal forces exceed external forces
- Stability emerges from the balance of compression and tension, not from rigid connection

In biology, bones are compression elements, muscles and tendons are tension elements, and fascia provides distributed stiffness. The key insight: the shape and stability of the structure are determined by the distribution of tension (muscle tone), not by the shape of the bones alone.

This has important implications: by modulating muscle tone (maintaining baseline activation without conscious movement), the nervous system pre-stresses the body. This pre-stress creates the "landscape" on which self-organization operates.

For example, standing balance involves maintaining baseline activation in leg muscles. This is not to generate force (gravity does that) but to pre-stress the structure. When a disturbance occurs, the pre-stressed structure responds more quickly and stably than if the muscles were initially slack.

8.6.4 How Pre-Stress Shapes the Drift Field

In the language of dynamical systems, muscle tone modifies the drift field:

$$f(\boldsymbol{x}) = f_0(\boldsymbol{x}) + f_{\text{tone}}(a_{\text{baseline}}, \boldsymbol{x}), \tag{8.23}$$

where a_{baseline} is the baseline (tonic) muscle activation. Increasing baseline activation increases the stiffness of the system and shifts the locations of attractors.

For postural control, the baseline activation (set by the brainstem and cerebellum) is set to create stable equilibria at standing, sitting, and other common postures. These equilibria are attractors: small perturbations are automatically corrected.

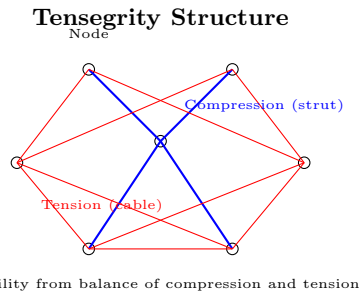


Figure 8.3: A simplified tensegrity structure: compression elements (blue struts) and tension elements (red cables) distribute forces throughout. Stability emerges from the pre-stressed balance, not from rigid geometry.

8.7 The Assembly Line: From Micro to Macro Stability

Bosch's concept of the "assembly line" describes how the nervous system organizes stability at multiple hierarchical levels:

Definition 8.11. The Assembly Line: Three Levels of Organization

Movement stability is not achieved at a single level but through a hierarchy:

1. **Level 1: Intramuscular Stability**
   - Muscle force-length and force-velocity properties
   - Muscle slack and pennation angle
   - The preflex: mechanical feedback through visco-elasticity
   - Time scale: 10–50 ms (faster than neural feedback)
2. **Level 2: Intermuscular Cooperation**
   - Antagonistic co-contraction for stiffness modulation
   - Agonistic synergies for force closure (multiple muscles stabilizing a joint)
   - Kinetic chains: sequential activation of muscles along the limb
   - Time scale: 50–200 ms (spinal reflexes and central pattern generators)
3. **Level 3: Intentional Patterns**
   - Task-level attractors and attractor hierarchies
   - Deliberate modulation of control parameters
   - Learning: modification of the control policy based on performance feedback
   - Time scale: 200 ms to seconds (cortical and cerebellar processing)

Each level maps onto the control-affine structure:

1. **Level 1** modifies the element-level dynamics (the mass matrix $M(\boldsymbol{q})$, damping $C(\boldsymbol{q}, \dot{\boldsymbol{q}})$, and gravity term $G(\boldsymbol{q})$)

2. **Level 2** shapes the constraint Jacobian and the effective stiffness K

3. **Level 3** determines the reference trajectories and the overall control law

Theorem 8.1 (Nested Lyapunov Argument for Composite Stability)**.** *For a hierarchically organized system, if:*

1. *Level 1 dynamics are stable (Lyapunov function V_1)*

2. *Level 2 dynamics are stable given Level 1 (Lyapunov function V_2)*

3. *Level 3 dynamics are stable given Levels 1 and 2 (Lyapunov function V_3)*

Then the composite system is stable (Lyapunov function $V = V_1 + V_2 + V_3$ works).
***Implication**: Bosch's assembly line is not just an organizational metaphor. It is a mathematically rigorous way to decompose the stability analysis of a complex system. By ensuring stability at each level, the nervous system avoids the curse of dimensionality: instead of analyzing a single high-dimensional system, it analyzes three lower-dimensional problems.*

8.8 Implications for Computational Modeling

8.8.1 Why Rigid-Body Models Miss Crucial Dynamics

The standard inverse dynamics model (Chapters 4–6) assumes:

- Muscles produce force instantaneously upon neural command
- Tendons are inextensible (rigid coupling to the skeleton)
- Damping is negligible

These assumptions are reasonable for *quick ballistic movements* (e.g., a throw). But for *sustained movements* (postural control, slow reaching) and *reflexive responses*, they fail:

- Tendons store and release elastic energy (catapult effect)
- Muscle visco-elasticity provides damping and stabilizes equilibria
- Neural delays (20+ ms) mean feedback is essential; open-loop inverse models are unreliable

The practical consequence: a controller designed using only rigid-body inverse dynamics will be unstable or inefficient when implemented on a real (biological or robotic) system with elastic elements and delays.

8.8.2 Adding Visco-Elastic Elements to Multibody Simulation

Modern simulation platforms (MuJoCo, Gazebo) allow specification of:

- Joint damping: $\tau_{\text{damp}} = -D\dot{\theta}$
- Tendon compliance: elastic force $F = K(\ell - \ell_{\text{slack}})$
- Passive muscle stiffness: F/L curves
- Muscle activation dynamics: delay and first-order filtering

The improved model is closer to biology and exhibits richer dynamics:

1. Attractors emerge naturally (no explicit trajectory tracking needed)
2. Perturbations are stabilized automatically (preflex)
3. Resonance phenomena appear (muscles/tendons oscillate at natural frequency)
4. Energy transfer improves (elastic storage in ballistic motions)

8.8.3 Parameter Identification Challenges

A difficulty: muscle parameters vary dramatically with activation level, fatigue, temperature, and training state. A simple static model is insufficient.

Parameters that must be measured or estimated:

- Force-length curves: typically 3–5 parameters per muscle
- Force-velocity curves: 4–6 parameters per muscle
- Tendon slack length and stiffness: 2 parameters per muscle
- Muscle fiber length and pennation angle: 2–3 parameters per muscle
- Total: ∼50–100 parameters for a 7-DOF arm with 10 muscles

Techniques for parameter identification:

1. **Direct measurement**: dissect cadavers, image tissue (invasive, limited)
2. **Inverse dynamics**: fit model to measured movements (requires high-quality motion capture)
3. **Optimization**: search parameter space to match simulation to biological data (computationally expensive)

8.8.4 Hybrid Control Architectures

The most successful approaches combine multiple layers:

1. **Reflexive layer**: local feedback loops (Renshaw inhibition, spindle feedback) Operating at 10–50 ms timescale, requiring no CNS involvement
2. **Feedforward layer**: central pattern generators or learned motor primitives Precomputed patterns that generate baseline movement kinematics
3. **Optimizing layer**: task-level feedback control Compares actual motion to desired motion and adjusts control parameters Operating at 100–500 ms timescale

A hybrid architecture avoids the speed-accuracy trade-off: the fast reflexive and feedforward layers handle real-time stabilization; the slower optimizing layer handles task adaptation and learning.

8.8.5 Current Limitations and Open Problems

Despite advances, several key challenges remain:

- **Plasticity**: how do neural circuits learn? How do muscle and tendon properties adapt to training? Most models are static.
- **Fatigue**: muscles weaken during sustained contraction, yet animals continue to move. What adaptation mechanisms maintain performance?
- **Noise and variability**: biological systems are noisy. How does the nervous system exploit noise for exploration (learning) while maintaining stability?
- **Integration with perception**: movement control is tightly linked to sensing (vision, proprioception, balance). How do sensorimotor loops emerge during development?
- **Scale**: models of a single limb work reasonably well. Models of whole-body movement coordinating 200+ muscles and 60+ DOFs are still primitive.

These are not technical obstacles but conceptual challenges: we lack frameworks for understanding learning, adaptation, and integration at the system level.

8.9 Summary and Connection to the Series

This chapter bridges Volume III (biomechanics: how the body is structured) and Volume IV (motor control: how the nervous system commands movement).

The key insights:

1. **Biology is not noise**: muscle visco-elasticity, tendon compliance, and neural architecture are not complications added to ideal rigid-body mechanics. They are fundamental features that enable stable, efficient movement.
2. **Self-organization reduces control burden**: by exploiting attractors and prestressing the body (tensegrity), the nervous system offloads computation to the mechanics. The CNS modulates parameters; the dynamics produces motion.

3. **Hierarchy enables stability**: stability at multiple levels (intramuscular, intermuscular, intentional) allows the nervous system to avoid the curse of dimensionality and achieve real-time control.

4. **Synergies and the UCM are not post-hoc descriptions**: they are direct consequences of the control structure (which inputs are available?) and the task structure (which outputs matter?).

The chapters ahead build on this foundation:

- **Chapter 8 (CPGs)**: central pattern generators are a concrete realization of self-organization in the nervous system, producing rhythmic attractors without conscious control.
- **Chapter 9 (Motor Learning)**: learning involves discovering the landscape of attractors and control parameters that are optimal for a task.
- **Chapter 10 (Computational Models)**: building biologically plausible models requires integrating all the insights from this chapter.
- **Chapter 11 (Robotics)**: designing robots that move as efficiently as animals requires exploiting passive dynamics, elastic elements, and self-organization— the same principles used by biology.

The central message: dynamics and biology are not separate disciplines. They are two sides of the same coin. The best models of motor control integrate both: the physics of the body and the biology of the nervous system must be understood together.

Exercises

1. **Hill Muscle Modeling.** Implement a Hill muscle model with force-length and force-velocity curves. Simulate a muscle contracting from $l = 0.8l_0$ to $l = 1.2l_0$ with a step in activation from $a = 0$ to $a = 0.8$. Plot force output over time. How does the SEE compliance affect the rise time?

2. **Attractors and Stability.** Construct a 2D dynamical system representing arm reaching:
$$\ddot{x} + D\dot{x} + K(x - x_{\text{target}}) = 0 \tag{8.24}$$
For three values of D (underdamped, critically damped, overdamped), simulate the response to a step command $x_{\text{target}} = 1$ and a sinusoidal disturbance. Which has the best robustness to disturbances?

3. **Uncontrolled Manifold.** Simulate a 3-link planar arm ($n = 3$ DOFs) reaching to a fixed hand position (p_x, p_y). Calculate the Jacobian $J(\theta)$, find the null space (UCM), and decompose endpoint variability into V_{UCM} and V_{ORT} across 100 trials with random joint noise. Verify that $V_{\text{UCM}}/V_{\text{ORT}} > 1$ (redundancy exploitation).

4. **Bifurcation in Bimanual Coordination.** Implement the HKB model:
$$\ddot{\phi} + \gamma\dot{\phi} + A(\omega)\sin(\phi) + 2B(\omega)\sin(2\phi) = 0 \tag{8.25}$$

where A and B are control parameters that vary with ω. Simulate the system starting in in-phase ($\phi = 0$) coordination and gradually increase ω (driving frequency). Observe and plot the bifurcation point at which the in-phase state becomes unstable and transitions to anti-phase. What frequency triggers the transition?

5. **(Programming)** Impedance Control and Robustness. Design a controller that reaches a target while rejecting disturbances. Compare two approaches: (1) impedance control with fixed stiffness $K = 100$, and (2) adaptive impedance that increases K if disturbances are detected. Measure endpoint accuracy and energy cost for a sequence of random disturbances. Which approach is more efficient?

Chapter 9

Central Pattern Generators

Draft Notice. This chapter is under active development and contains only a structural outline with preliminary material. Full exposition, worked examples, proofs, and exercises will be added in future editions.

Rhythm is the most ancient form of motor control.

9.1 CPGs in Biological Motor Control

Central pattern generators are neural circuits that produce rhythmic motor output without rhythmic sensory input or supraspinal commands. Evidence for CPGs:

- Isolated lamprey spinal cord generates swimming patterns [8]
- Deafferented cats produce coordinated stepping on treadmills [14]
- Songbird pre-motor nuclei show rhythmic activity in vitro [36]

CPGs underlie locomotion, respiration, chewing, and other rhythmic behaviors.

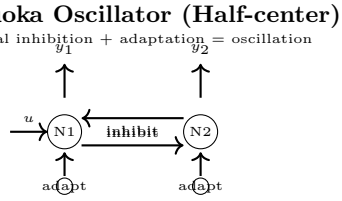


Figure 9.1: Matsuoka oscillator CPG circuit: mutual inhibition between neurons with adaptation variables creates an alternating rhythm. Tonic input frequency-modulates the oscillation without losing periodicity.

9.2 The Matsuoka Oscillator

The most widely used CPG model in robotics is the Matsuoka oscillator [29]:

$$\tau_1 \dot{x}_i = -x_i - \beta v_i - \sum_{j \neq i} w_{ij} y_j + u_i + s_i, \tag{9.1}$$

$$\tau_2 \dot{v}_i = -v_i + y_i, \tag{9.2}$$

$$y_i = \max(0, x_i), \tag{9.3}$$

where x_i is the membrane potential, v_i is the adaptation variable, y_i is the output, w_{ij} are inhibitory coupling weights, u_i is the tonic input, and s_i is the sensory feedback.

```python
import numpy as np
from scipy.integrate import solve_ivp

class MatsuokaOscillator:
    """Two-neuron Matsuoka oscillator for CPG-based control."""

    def __init__(
        self,
        tau1: float = 0.1,
        tau2: float = 0.3,
        beta: float = 2.5,
        w_mutual: float = 2.0,
        tonic_input: float = 1.0
    ):
        self.tau1 = tau1
        self.tau2 = tau2
        self.beta = beta
        self.w = w_mutual
        self.u = tonic_input

    def dynamics(self, t: float, state: np.ndarray) -> np.ndarray:
        x1, v1, x2, v2 = state
        y1 = max(0.0, x1)
        y2 = max(0.0, x2)

        dx1 = (-x1 - self.beta * v1 - self.w * y2 + self.u) / self.tau1
        dv1 = (-v1 + y1) / self.tau2
        dx2 = (-x2 - self.beta * v2 - self.w * y1 + self.u) / self.tau1
        dv2 = (-v2 + y2) / self.tau2

        return np.array([dx1, dv1, dx2, dv2])

    def simulate(self, t_end: float = 5.0,
                 dt: float = 0.001) -> tuple[np.ndarray, np.ndarray]:
        state0 = np.array([0.1, 0.0, -0.1, 0.0])
        t_eval = np.arange(0, t_end, dt)
        sol = solve_ivp(self.dynamics, [0, t_end], state0,
                        t_eval=t_eval, method='RK45')

        # Extract outputs
        y1 = np.maximum(0, sol.y[0])
        y2 = np.maximum(0, sol.y[2])
        output = y1 - y2  # Net output

        return sol.t, output

```

```python

# Simulate and print summary
cpg = MatsuokaOscillator(tau1=0.1, tau2=0.3)
t, output = cpg.simulate(t_end=5.0)

# Estimate frequency from zero crossings
crossings = np.where(np.diff(np.sign(output)))[0]
if len(crossings) >= 4:
    periods = np.diff(t[crossings[::2]])
    freq = 1.0 / np.mean(periods)
    print(f"CPG frequency: {freq:.2f} Hz")
    print(f"CPG amplitude: {np.max(np.abs(output[len(output)//2:])):.3f}")
```

Listing 9.1: Matsuoka oscillator CPG implementation.

9.3 Phase Coupling and Inter-Limb Coordination

Multiple CPGs can be coupled to produce coordinated multi-limb movement:

$$\dot{\phi}_i = \omega_i + \sum_j K_{ij} \sin(\phi_j - \phi_i - \psi_{ij}), \tag{9.4}$$

where ϕ_i is the phase of the i-th oscillator, ω_i is its natural frequency, K_{ij} is the coupling strength, and ψ_{ij} is the desired phase relationship.

For quadruped locomotion:

- Walk: $\psi = (0, \pi, \pi/2, 3\pi/2)$ (diagonal pairs alternating)
- Trot: $\psi = (0, \pi, \pi, 0)$ (diagonal pairs synchronized)
- Gallop: $\psi = (0, 0, \pi, \pi)$ (front-back alternating)

Exercises

1. **CPG Tuning.** Vary τ_1/τ_2 ratio and plot the resulting frequency and duty cycle. How does the ratio control the waveform shape?
2. **Phase Coupling.** Implement 4 coupled oscillators and produce walk, trot, and gallop patterns by varying ψ_{ij}.
3. **(Programming.)** Use the Matsuoka CPG to drive a simulated bipedal walker. Tune the CPG parameters for stable walking.

Chapter 10

Motor Learning and Adaptation

Draft Notice. This chapter is under active development and contains only a structural outline with preliminary material. Full exposition, worked examples, proofs, and exercises will be added in future editions.

> Practice does not make perfect. Practice makes permanent. Only perfect practice makes perfect.

10.1 Three Learning Systems

Motor learning involves three distinct neural systems with different timescales and mechanisms:

1. **Error-based learning** (cerebellum): uses sensory prediction errors (climbing fiber signals) to update cerebellar forward models via LTD [28, 24]. Trial-by-trial timescale (minutes to hours). It drives adaptation both to altered limb *dynamics* [34] and to an altered visuomotor *mapping* [25] — and those two are acquired independently of one another, so they are worth keeping distinct rather than filing together as "adaptation".

2. **Reinforcement learning** (basal ganglia): uses dopamine reward signals to strengthen successful motor patterns [33]. Slower than error-based learning (hours to days). Enables discovery of novel solutions not implicit in the task structure.

3. **Consolidation and structural learning** (cortical): repetition of a movement biases future movements toward repeated patterns [9]. Timescale of hours to days; explains habit formation. Involves gradual changes in M1 tuning and synaptic weights.

10.2 Visuomotor Adaptation

The classic paradigm for studying error-based learning [25]:

```
import numpy as np

def simulate_visuomotor_adaptation(
    n_baseline: int = 50,
```

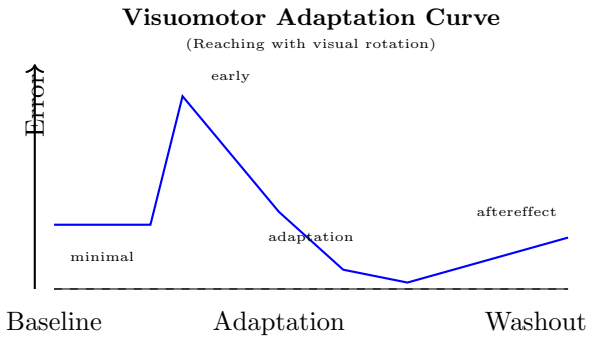


Figure 10.1: Characteristic visuomotor adaptation profile [25]: rapid error reduction (first 20 trials), gradual asymptote, persistent aftereffect demonstrating learning consolidation.

```
    n_adaptation: int = 200,
    n_washout: int = 100,
    perturbation_deg: float = 30.0,
    learning_rate: float = 0.05,
    retention: float = 0.98
) -> dict:
    """Simulate visuomotor rotation adaptation.

    A simple state-space model of adaptation:
    x_{k+1} = retention * x_k + learning_rate * e_k

    Parameters
    ----------
    n_baseline : int
        Baseline trials (no perturbation).
    n_adaptation : int
        Adaptation trials (perturbation on).
    n_washout : int
        Post-adaptation trials (perturbation off).
    perturbation_deg : float
        Rotation perturbation in degrees.
    learning_rate : float
        Rate of error-based learning.
    retention : float
        Memory retention between trials.

    Returns
    -------
    dict
        Keys: 'trial', 'reaching_error', 'internal_state',
              'phase' (labels for each trial)
    """
    n_total = n_baseline + n_adaptation + n_washout
    p = np.radians(perturbation_deg)

    # Internal state (learned compensation)
    x = 0.0
    errors = np.zeros(n_total)
    states = np.zeros(n_total)
    phases = []

```

```
    for trial in range(n_total):
        if trial < n_baseline:
            perturbation = 0.0
            phases.append('baseline')
        elif trial < n_baseline + n_adaptation:
            perturbation = p
            phases.append('adaptation')
        else:
            perturbation = 0.0
            phases.append('washout')

        # Reaching error = perturbation - compensation
        error = perturbation - x
        errors[trial] = np.degrees(error)
        states[trial] = np.degrees(x)

        # Update internal state
        x = retention * x + learning_rate * error

    return {
        'trial': np.arange(n_total),
        'reaching_error': errors,
        'internal_state': states,
        'phase': phases
    }


result = simulate_visuomotor_adaptation()
print("Visuomotor Adaptation Summary:")
print(f"  Baseline error:  {np.mean(np.abs(result['reaching_error'][:50])):.1
    f} deg")
print(f"  Final adaptation error: "
      f"{np.abs(result['reaching_error'][249]):.1f} deg")
print(f"  Aftereffect: {result['reaching_error'][250]:.1f} deg")
```

Listing 10.1: Visuomotor adaptation simulation.

10.3 Savings and Interference

10.3.1 Savings

Re-learning a previously learned adaptation is faster than initial learning, even after complete washout [20] — an observation that predates motor control, going back to Ebbinghaus's 1885 work on memory. This savings effect suggests that motor memories are not erased but suppressed, with additional learning circuits retaining implicit information about the learning dynamics.

10.3.2 Interference and Dual-Rate Learning

Learning opposite perturbations (A then B) produces anterograde interference: learning B slows retention of A (Li and Xie, 2000). This is explained by dual-rate learning models: a fast process learns trial-by-trial; a slow process consolidates. Interference occurs because the fast process is task-specific while the slow process is more general.

10.4 Structural Learning and Meta-Learning

Over many adaptation experiences, subjects develop the ability to adapt faster—they learn to learn (context-dependent learning). Smith et al. (1998) showed that prior exposure to one visuomotor rotation task accelerates learning of a novel rotation. This involves learning the structure of the task space itself, not just individual parameter values. Brain areas like dorsolateral prefrontal cortex appear to encode such meta-learning (Gershman et al., 2017).

Exercises

1. **State-Space Model.** Using the provided code, explore how learning rate and retention affect adaptation speed and aftereffect size.
2. **Dual-Rate Model.** Implement a dual-rate model (fast + slow process) and show that it can explain savings.
3. **(Programming.)** Simulate interference by adapting to $+30°$ then $-30°$. Plot the adaptation curves and aftereffects.

Chapter 11

Computational Models of Motor Control

Draft Notice. This chapter is under active development and contains only a structural outline with preliminary material. Full exposition, worked examples, proofs, and exercises will be added in future editions.

A theory of motor control must ultimately be a theory of computation: what is computed, how, and why.

Table 11.1: Comparison of computational motor control theories.

Theory	**Learning Rule**	**Variability**	**Key Prediction**
OFC (Todorov, 2004)	Error feedback	Tolerate task-irrelevant	Bell-shaped velocity profiles
Active Inference (Friston, 2010)	Minimize free energy	Prediction error-driven	Sensorimotor uncertainty shapes movement
Internal Model (Kawato, 1999)	Forward model learning (climbing fiber error)	Exploration + exploitation	Gradual visuomotor adaptation
Synergy Control (d'Avella, 2005)	Reduced-dim learning (NMF, ICA)	Synergy-space variability allowed	Movement flexibility without dimensionality

11.1 Optimal Feedback Control

The dominant computational theory of motor control is optimal feedback control (OFC), formulated by Todorov and Jordan [38]. OFC explains motor variability patterns (minimum intervention principle), predicts smooth velocity profiles, and unifies cerebellar learning with optimal control:

Definition 11.1. The OFC Framework

The nervous system controls movement by solving, in real-time, the stochastic optimal control problem:

$$\min_{\boldsymbol{u}(\cdot)} \mathbb{E}\left[\phi(\boldsymbol{x}_T) + \int_0^T \ell(\boldsymbol{x}_t, \boldsymbol{u}_t)\,\mathrm{d}t\right], \tag{11.1}$$

subject to:

$$\mathrm{d}\boldsymbol{x} = f(\boldsymbol{x}, \boldsymbol{u})\,\mathrm{d}t + G(\boldsymbol{x})\mathrm{d}\boldsymbol{w}, \tag{11.2}$$

where $\mathrm{d}\boldsymbol{w}$ is Brownian noise representing signal-dependent motor noise.

Equation (11.1) is a *stochastic* optimal control problem, and the distinction matters. Because the dynamics carry a Brownian term, the solution is characterised by a Hamilton–Jacobi–Bellman equation for the value function, not by the Pontryagin maximum principle applied along a single trajectory. The deterministic geometric control theory this series draws on elsewhere—Agrachev & Sachkov [1]—treats smooth ordinary differential equations with no noise term and develops optimality through Pontryagin's principle; it does not cover the problem above.

The reason a stochastic formulation is needed at all is that motor noise scales with the magnitude of the control signal, so it cannot be treated as a small additive disturbance around a nominal plan [15]. That single fact is what makes the minimum intervention principle below a prediction rather than a preference.

11.1.1 The Minimum Intervention Principle

OFC naturally explains why the nervous system tolerates task-irrelevant variability (the UCM from Chapter 1): the optimal controller only corrects deviations that affect the cost function. Task-irrelevant deviations are "free" and need not be corrected.

```python
import numpy as np
from scipy.linalg import solve_continuous_are

def lqr_ofc(
    A: np.ndarray,
    B: np.ndarray,
    Q: np.ndarray,
    R: np.ndarray,
    state_noise_cov: np.ndarray | None = None
) -> tuple[np.ndarray, np.ndarray]:
    """Compute LQR optimal feedback gains.

    This implements the infinite-horizon OFC solution.

    Parameters
    ----------
    A : np.ndarray, shape (n, n)
        State dynamics matrix.
    B : np.ndarray, shape (n, m)
        Input matrix.
    Q : np.ndarray, shape (n, n)
        State cost matrix.
    R : np.ndarray, shape (m, m)
```

```
        Control cost matrix.
    state_noise_cov : np.ndarray, shape (n, n), optional
        Process noise covariance.

    Returns
    -------
    tuple
        (K, P) where K is the optimal gain and P is the Riccati solution.
    """
    # Solve continuous algebraic Riccati equation
    P = solve_continuous_are(A, B, Q, R)

    # Optimal gain
    K = np.linalg.solve(R, B.T @ P)

    return K, P


# Example: 2D reaching with motor noise
# State: [x, y, dx, dy], Control: [Fx, Fy]
A = np.array([
    [0, 0, 1, 0],
    [0, 0, 0, 1],
    [0, 0, -2, 0],  # damping
    [0, 0, 0, -2],
])
B = np.array([
    [0, 0],
    [0, 0],
    [1, 0],
    [0, 1],
])

# Task: reach to target with minimum effort
# High cost on position at target, low cost on path
Q = np.diag([10, 10, 1, 1])  # Position matters more than velocity
R = 0.01 * np.eye(2)  # Low control cost

K, P = lqr_ofc(A, B, Q, R)
print("OFC gain matrix K:")
print(K)
print("\nMinimum intervention: gains are higher for")
print("task-relevant (position) than task-irrelevant dimensions")
```

Listing 11.1: Linear-quadratic optimal feedback controller.

11.2 Signal-Dependent Noise

A key feature of biological motor systems is that motor noise is **signal-dependent**: the noise variance scales with the control signal magnitude:

$$\boldsymbol{u} = \bar{\boldsymbol{u}} + \sigma(\bar{\boldsymbol{u}})\boldsymbol{\xi}, \quad \sigma(\bar{\boldsymbol{u}}) = k\,\|\bar{\boldsymbol{u}}\|\,, \tag{11.3}$$

where $\boldsymbol{\xi}$ is white Gaussian noise and k is the noise coefficient (empirically $k \approx 0.01$–0.05 in human reaching). This property has been measured extensively (Schmidt et al., 1979; Harris and Wolpert, 1998).

Under signal-dependent noise, the optimal trajectory minimizes integrated squared jerk (Flash and Hogan, 1985). This is a key prediction of OFC: the bell-shaped velocity profiles observed in human reaching emerge naturally as solutions to the signal-dependent noise minimization problem, not from any explicit minimization of jerk (Todorov, 2004).

11.3 Active Inference (Friston Framework)

Active inference (Friston, 2010; Friston et al., 2016) reframes motor control as inference: the agent minimizes prediction error by acting to make observations match its generative model. This is formalized through variational free energy minimization:

$$F = \mathbb{E}_q[\ln q(\boldsymbol{x}) - \ln p(\boldsymbol{x}, \boldsymbol{y})], \tag{11.4}$$

where F is the variational free energy, $q(\boldsymbol{x})$ is the approximate posterior belief, and $p(\boldsymbol{x}, \boldsymbol{y})$ is the generative model. The agent chooses actions to minimize F. This unifies action, perception, and learning under a single principle. While the experimental evidence is still developing, active inference correctly predicts many phenomena including the minimum intervention principle and why the brain is tolerant of task-irrelevant variability.

Exercises

1. **OFC Reaching.** Using the LQR code, simulate reaching movements to different targets. Show that task-irrelevant variability is tolerated.

2. **Signal-Dependent Noise.** Simulate reaching with and without signal-dependent noise. Plot the trajectory variability as a function of movement speed.

3. **(Programming.)** Implement the minimum-jerk model and the minimum-variance model. Show that both predict smooth bell-shaped velocity profiles.

Chapter 12

From Neural Architecture to Robot Control

Draft Notice. This chapter is under active development and contains only a structural outline with preliminary material. Full exposition, worked examples, proofs, and exercises will be added in future editions.

If we truly understood how humans move, we could build robots that move like humans. We cannot. This tells us something.

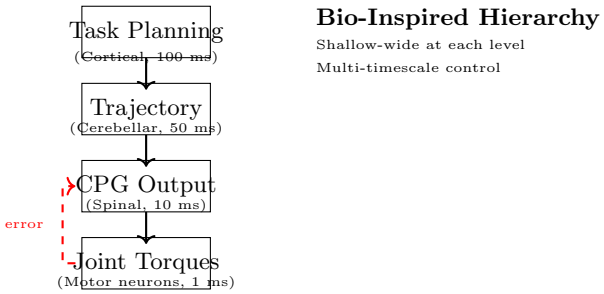


Figure 12.1: Hierarchical multi-timescale control architecture: each level shallow (1-2 layers) but wide (thousands of parallel channels), operating at increasing timescales and levels of abstraction.

12.1 Lessons from Biology for Robot Design

The four volumes of theory converge on practical design principles for robotic control systems:

Key Insight 12.1. Bio-Inspired Control Design Principles

1. **Exploit passive dynamics** (Vol II, Vol IV Ch 7): design the mechanical system so that natural dynamics do useful work. Use compliance, not rigidity.
2. **Use shallow-wide controllers** (Vol IV Ch 4): for real-time control, prefer 2–3 layer networks with thousands of neurons over deep networks with hundreds of layers.
3. **Employ synergy-based dimensionality reduction** (Vol IV Ch 4): define 4–8 actuation synergies and control in synergy space, not muscle space.
4. **Implement forward models** (Vol IV Ch 6): use the Smith predictor architecture to compensate for sensor/actuator delays.
5. **Design for contraction** (Vol I Ch 4): ensure the closed-loop system is contracting, guaranteeing stability and robustness.
6. **Optimize trajectories, not setpoints** (Vol II): design trajectory-centric controllers with funnel guarantees.
7. **Accept variability** (Vol IV Ch 1): allow task-irrelevant variability. Don't over-constrain the controller.

12.2 Cerebellar-Inspired Adaptive Control

The cerebellum's circuit architecture [28, 2, 24] directly suggests an adaptive control scheme:

- **Granule cells** (the bulk of some 69 billion cerebellar neurons [3]): create a compressed basis function representation of input state via mossy fiber–granule cell synapses
- **Purkinje cells** (∼17 million): linearly combine granule cell outputs; synaptic weights change via LTD when climbing fiber error signals are present
- **Climbing fiber error** (inferior olive): carries sensory prediction error from brainstem nuclei; signals at ∼1 Hz but with precise timing (10 ms resolution)

This maps directly to:

$$\boldsymbol{u}_{\text{adaptive}}(\boldsymbol{x}) = \hat{W}^T \boldsymbol{\phi}(\boldsymbol{x}), \quad \dot{\hat{W}} = -\Gamma \boldsymbol{\phi}(\boldsymbol{x}) \boldsymbol{e}^T, \tag{12.1}$$

where $\boldsymbol{\phi}(\boldsymbol{x})$ are radial basis functions (granule cell expansion), $\hat{W}$ are adaptive weights (parallel fiber–Purkinje cell synapses), and $\boldsymbol{e}$ is the tracking error (climbing fiber signal).

12.3 CPG-Based Locomotion Controllers

Following Chapter 8, CPG-based locomotion controllers use coupled oscillators to generate walking/running patterns. This architecture is inspired by Grillner's [14] demonstration that deafferented spinal cords generate coordinated stepping. The CPG provides the *timing* and *synergy* structure; feedback provides *stability* and *responsiveness*:

```python
import numpy as np

class BioInspiredLocomotionController:
    """Hierarchical control architecture inspired by
    biological motor control."""

    def __init__(
        self,
        n_joints: int,
        n_synergies: int = 4,
        cpg_freq: float = 1.0
    ):
        self.n_joints = n_joints
        self.n_synergies = n_synergies
        self.cpg_freq = cpg_freq

        # Synergy matrix (learned from motion data)
        self.W = np.random.randn(n_joints, n_synergies) * 0.1

        # CPG phases for each synergy
        self.phases = np.linspace(
            0, 2 * np.pi * (1 - 1/n_synergies), n_synergies
        )

        # Feedback gains (shallow: single layer)
        self.K_fb = np.zeros((n_synergies, 2 * n_joints))

    def cpg_output(self, t: float) -> np.ndarray:
        """Generate rhythmic synergy activations from CPG."""
        activations = np.array([
            max(0.0, np.sin(2 * np.pi * self.cpg_freq * t + phi))
            for phi in self.phases
        ])
        return activations

    def feedforward(self, t: float) -> np.ndarray:
        """Feedforward joint torques from CPG + synergies."""
        c = self.cpg_output(t)
        return self.W @ c

    def feedback(self, state_error: np.ndarray) -> np.ndarray:
        """Feedback correction (shallow: one layer)."""
        c_fb = self.K_fb @ state_error
        return self.W @ c_fb

    def control(self, t: float,
                state_error: np.ndarray) -> np.ndarray:
        """Total control: feedforward (CPG) + feedback."""
        return self.feedforward(t) + self.feedback(state_error)


# Example: 6-joint biped with 4 synergies
controller = BioInspiredLocomotionController(
    n_joints=6, n_synergies=4, cpg_freq=1.2
)

# Generate one cycle of control signals
t = np.linspace(0, 1.0/1.2, 200)
torques = np.array([controller.feedforward(ti) for ti in t])

```

```
print(f"Bio-inspired controller: {controller.n_joints} joints, "
      f"{controller.n_synergies} synergies")
print(f"Control dimensionality reduced from {controller.n_joints} "
      f"to {controller.n_synergies}")
print(f"Peak torque magnitude: {np.max(np.abs(torques)):.3f}")
```

Listing 12.1: Bio-inspired locomotion controller architecture.

12.4 Whole-Body Control Architectures

For humanoid robots, the bio-inspired architecture suggests:

1. **Level 0 (Spinal)**: joint-level impedance control with reflex-like feedback (∼1 ms loop)
2. **Level 1 (CPG)**: rhythmic pattern generation for locomotion (∼10 ms loop)
3. **Level 2 (Cerebellar)**: adaptive feedforward based on forward model predictions (∼50 ms loop)
4. **Level 3 (Cortical)**: trajectory planning and task-level control (∼100 ms loop)

Each level is **shallow** (1–2 layers) but **wide** (many parallel channels), operating at progressively longer timescales.

12.5 Companion Software: Practical Implementation

Volume V provides the practical tools to implement and test these controllers using the pedagogical platform accompanying this textbook. The connection points:

- **MuJoCo**: test CPG-based locomotion controllers on musculoskeletal models
- **Drake**: implement trajectory optimization for trajectory-centric control
- **Pinocchio**: compute forward/inverse dynamics efficiently in real-time

Exercises

1. **Controller Architecture.** Using the bio-inspired controller code, vary the number of synergies and measure the resulting control signal dimensionality and smoothness.
2. **Cerebellar Adaptive Control.** Implement the cerebellar-inspired adaptive controller for a 2-DOF arm and show that it learns an inverse model over 50 trials.
3. **Hierarchical Design.** Design a 3-level hierarchical controller for a simulated biped, with impedance control at Level 0, CPG at Level 1, and trajectory correction at Level 2.
4. **(Programming.)** Implement the full bio-inspired controller for a MuJoCo walking model and benchmark against a standard PD controller.

Bibliography

[1] Andrei A. Agrachev and Yuri L. Sachkov. *Control Theory from the Geometric Viewpoint*, volume 87 of *Encyclopaedia of Mathematical Sciences*. Springer, 2004.

[2] J. S. Albus. A theory of cerebellar function. *Mathematical Biosciences*, 10(1-2):25–61, 1971.

[3] Frederico A. C. Azevedo, Ludmila R. B. Carvalho, Lea T. Grinberg, José Marcelo Farfel, Renata E. L. Ferretti, Renata E. P. Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009.

[4] Richard Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[5] N. A. Bernstein. *The Coordination and Regulation of Movements*. Pergamon Press, 1967.

[6] F. Bosch. *Strength Training and Coordination: An Integrative Approach*. On Target Publications, 2015.

[7] Frans Bosch. *Anatomy of Agility: movement analysis in sport*. 20/10 Publishers, 2020.

[8] Avis H. Cohen and Peter Wallén. The neuronal correlate of locomotion in fish: "fictive swimming" induced in an in vitro preparation of the lamprey spinal cord. *Experimental Brain Research*, 41(1), 1980.

[9] J. D. Cohen, G. Aston-Jones, and M. S. Gilzenrat. A theory of how the basal ganglia generates and uses neural signals. *Trends in Neurosciences*, 28(11):574–586, 2005.

[10] Andrea d'Avella and Emilio Bizzi. Shared and specific muscle synergies in natural motor behaviors. *Proceedings of the National Academy of Sciences*, 102(8):3076–3081, 2005.

[11] T. Flash and N. Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *Journal of Neuroscience*, 5(7):1688–1703, 1985.

[12] A. P. Georgopoulos, J. F. Kalaska, R. Caminiti, and J. T. Massey. On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. *The Journal of Neuroscience*, 2(11):1527–1537, 1982.

[13] Apostolos P. Georgopoulos, Andrew B. Schwartz, and Ronald E. Kettner. Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419, 1986.

[14] S. Grillner. Locomotion in vertebrates: central mechanisms and reflex interaction. *Physiological Reviews*, 55(2):247–304, 1975.

[15] Christopher M. Harris and Daniel M. Wolpert. Signal-dependent noise determines motor planning. *Nature*, 394(6695):780–784, 1998.

[16] A. V. Hill. The heat of shortening and the dynamic constants of muscle. *Proceedings of the Royal Society B: Biological Sciences*, 126(843):136–195, 1938.

[17] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

[18] Neville Hogan. Adaptive control of mechanical impedance by coactivation of antagonist muscles. *IEEE Transactions on Automatic Control*, 29(8):681–690, 1984.

[19] B. Hommel, J. Müsseler, G. Aschersleben, and W. Prinz. The theory of event coding (tec): a framework for perception and action planning. *Behavioral and Brain Sciences*, 24(5):849–878, 2001.

[20] Vincent S. Huang, Adrian Haith, Pietro Mazzoni, and John W. Krakauer. Rethinking motor learning and savings in adaptation paradigms: model-free memory for successful actions combines with internal models. *Neuron*, 70(4):787–801, 2011.

[21] Hiroshi Imamizu, Satoru Miyauchi, Tomoe Tamada, Yuka Sasaki, Ryousuke Takino, Benno Pütz, Toshinori Yoshioka, and Mitsuo Kawato. Human cerebellar activity reflecting an acquired internal model of a new tool. *Nature*, 403(6766):192–195, 2000.

[22] W. James. *The Principles of Psychology*. Holt, Rinehart and Winston, 1890.

[23] Eric R. Kandel, James H. Schwartz, Thomas M. Jessell, Steven A. Siegelbaum, and A. J. Hudspeth. *Principles of Neural Science*. McGraw-Hill Medical, New York, 5th edition, 2013.

[24] M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57(3):169–185, 1987.

[25] John W. Krakauer, Maria-Felice Ghilardi, and Claude Ghez. Independent learning of internal models for kinematic and dynamic control of reaching. *Nature Neuroscience*, 2(11):1026–1031, 1999.

[26] Mark L. Latash. The bliss (not the problem) of motor abundance (not redundancy). *Experimental Brain Research*, 217(1):1–5, 2012.

[27] G. E. Loeb. Control implications of musculoskeletal mechanics. In *Proceedings of the 17th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 2, pages 1393–1394, Montreal, Canada, 1995.

[28] D. Marr. A theory of cerebellar cortex. *Journal of Physiology*, 202(2):437–470, 1969.

[29] K. Matsuoka. Sustained oscillations generated by mutually inhibiting neurons with adaptation. *Biological Cybernetics*, 52(6):367–376, 1985.

[30] V. B. Mountcastle. The columnar organization of the neocortex. *Brain*, 120(4):701–722, 1997.

[31] Wilder Penfield and Edwin Boldrey. Somatic motor and sensory representation in the cerebral cortex of man as studied by electrical stimulation. *Brain*, 60(4):389–443, 1937.

[32] J. P. Scholz and G. Schöner. The uncontrolled manifold concept: Identifying control variables for a functional task. *Experimental Brain Research*, 126:289–306, 1999.

[33] W. Schultz, P. Dayan, and P. R. Montague. A neural substrate of prediction and reward. *Science*, 275(5307):1593–1599, 1997.

[34] R. Shadmehr and F. A. Mussa-Ivaldi. Adaptive representation of dynamics during learning of a motor task. *Journal of Neuroscience*, 14(5):3208–3224, 1994.

[35] Maurice A. Smith and Reza Shadmehr. Intact ability to learn internal models of arm dynamics in huntington's disease but not cerebellar degeneration. *Journal of Neurophysiology*, 93(5):2809–2821, 2005.

[36] Michele M. Solis and David J. Perkel. Rhythmic activity in a forebrain vocal control nucleus in vitro. *The Journal of Neuroscience*, 25(11):2811–2822, 2005.

[37] Aparna Suvrathan, Hannah L. Payne, and Jennifer L. Raymond. Timing rules for synaptic plasticity matched to behavioral function. *Neuron*, 92(5):959–967, 2016.

[38] E. Todorov and M. I. Jordan. Optimal feedback control as a theory of motor coordination. *Nature Neuroscience*, 5(11):1226–1235, 2002.

[39] M. C. Tresch, V. C. K. Cheung, and A. d'Avella. Matrix factorization algorithms for the identification of muscle synergies. *Journal of Neurophysiology*, 95(4):2199–2212, 2006.

[40] D. M. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7-8):1317–1329, 1998.