Embodied-AI:
Perception, Representation and Action

- **DATA 8010 & ELEC 8111**
- **Instructor: Prof. Yanchao Yang**

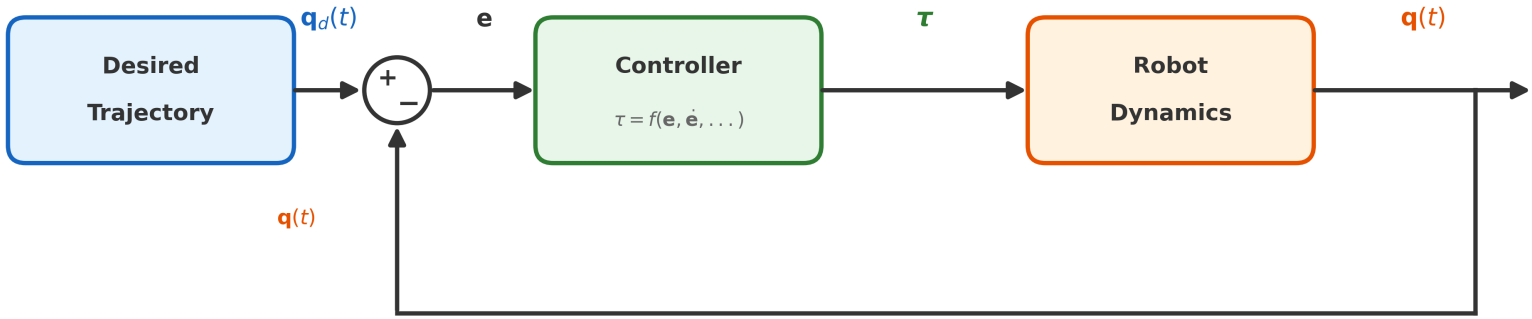
Control for Manipulation and Locomotion

Embodied AI: Perception, Representation and Action

The Control Problem

How do we make robots move the way we want them to?

Key Symbols: $\mathbf{q} \in \mathbb{R}^n$ = joint positions • $\mathbf{q}_d(t)$ = desired trajectory • $\boldsymbol{\tau} \in \mathbb{R}^n$ = joint torques

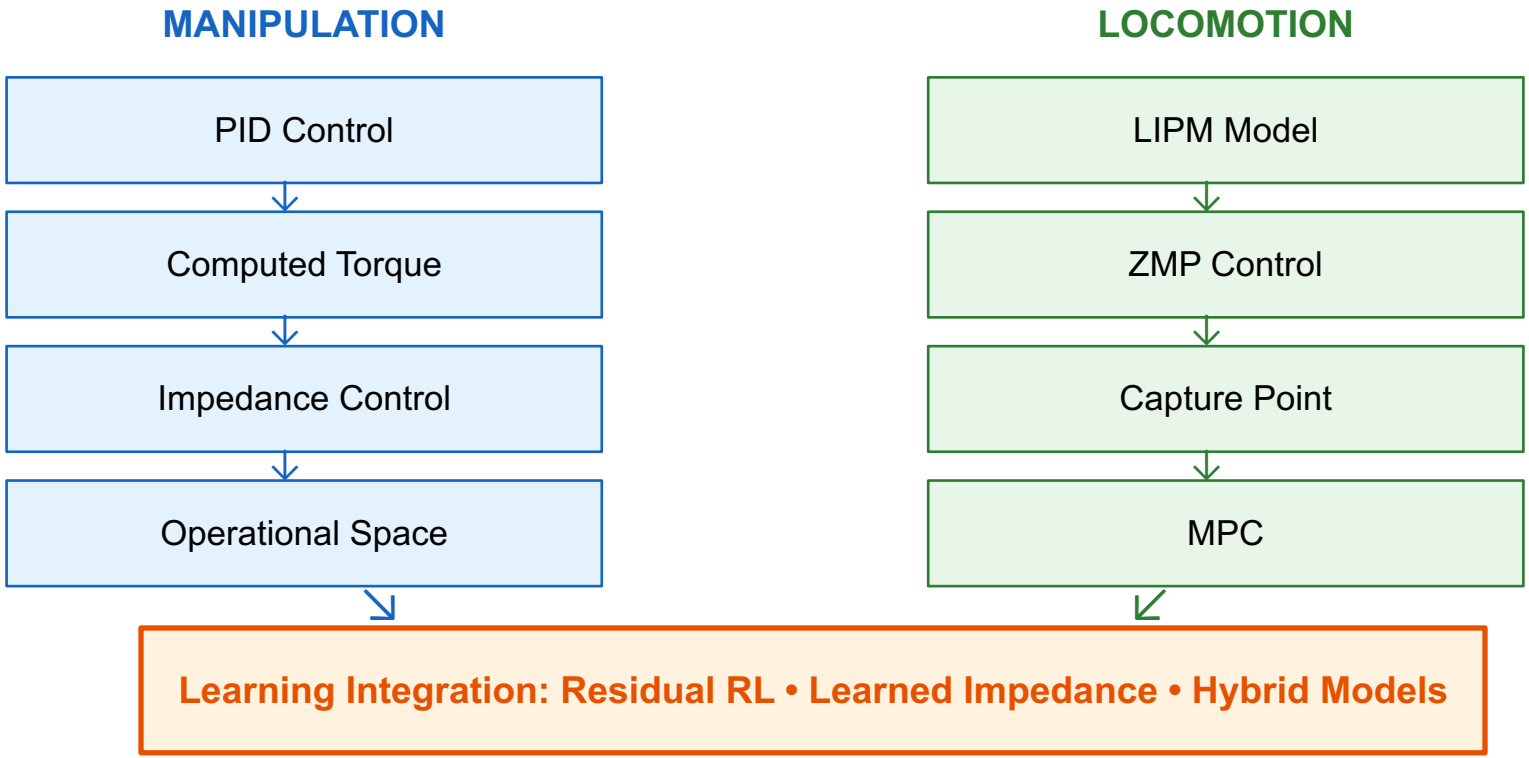


Feedback (sensors measure actual position)

The Challenge: Robot dynamics are nonlinear, coupled, and configuration-dependent. The controller must compute $\boldsymbol{\tau}$ from the error between desired $\mathbf{q}_d(t)$ and actual $\mathbf{q}(t)$. This gap between trajectory specification and motor commands is where control theory lives.

Today's Journey

From simple to sophisticated, each method addresses limitations of its predecessor



Learning Objectives

By the end of this lecture, you will be able to:

1. Explain computed torque control and feedback linearization for nonlinear robot dynamics
2. Design impedance controllers and distinguish them from explicit force control
3. Explain operational space control and its connection to VLAs and diffusion policies
4. Describe the Linear Inverted Pendulum Model (LIPM) and derive ZMP stability criterion
5. Explain capture point and its role in push recovery for bipedal robots
6. Describe MPC for locomotion and its role as the baseline that learning augments
7. Analyze how classical controllers integrate with learning (residual policies, learned impedance)

Control Modes: A Quick Review

Position Control	Velocity Control	Torque Control
Command: target angles $\mathbf{q}$	Command: target velocity $\dot{\mathbf{q}}$	Command: motor torques $\boldsymbol{\tau}$
Robot servos to $\mathbf{q}_d(t)$	Robot tracks $\dot{\mathbf{q}}_d(t)$	Direct actuation
✓ Stable, predictable	✓ Smooth motion	✓ Enables compliance
✗ Stiff at contact	✗ Can drift over time	✗ Needs dynamics model
Use: Pick-and-place	**Use:** Continuous tasks	**Use:** Contact, learning

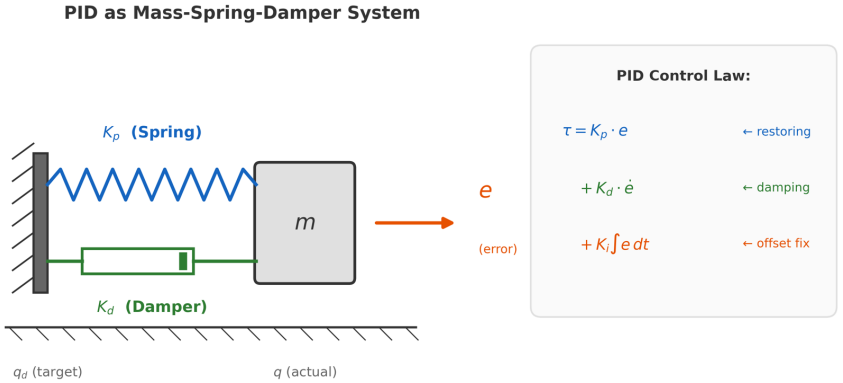
Learning Connection

In RL for robotics, action space choice is critical.
Isaac Lab defaults to position control for stability during early training.
Torque control enables compliance but requires 10-100× more samples.
Many successful systems use position control with learned residuals, combining stability with adaptability.

PID Control: The Starting Point

Variable Definitions

$\mathbf{q}, \mathbf{q}_d \in \mathbb{R}^n$ — Actual and desired joint positions
$\dot{\mathbf{q}}, \dot{\mathbf{q}}_d \in \mathbb{R}^n$ — Actual and desired velocities
$\mathbf{e} = \mathbf{q}_d - \mathbf{q}$ — Position error
$\dot{\mathbf{e}} = \dot{\mathbf{q}}_d - \dot{\mathbf{q}}$ — Velocity error
$\mathbf{K}_p, \mathbf{K}_i, \mathbf{K}_d$ — Gain matrices (diagonal)
$\boldsymbol{\tau} \in \mathbb{R}^n$ — Joint torque command



EQ1: PID Control Law

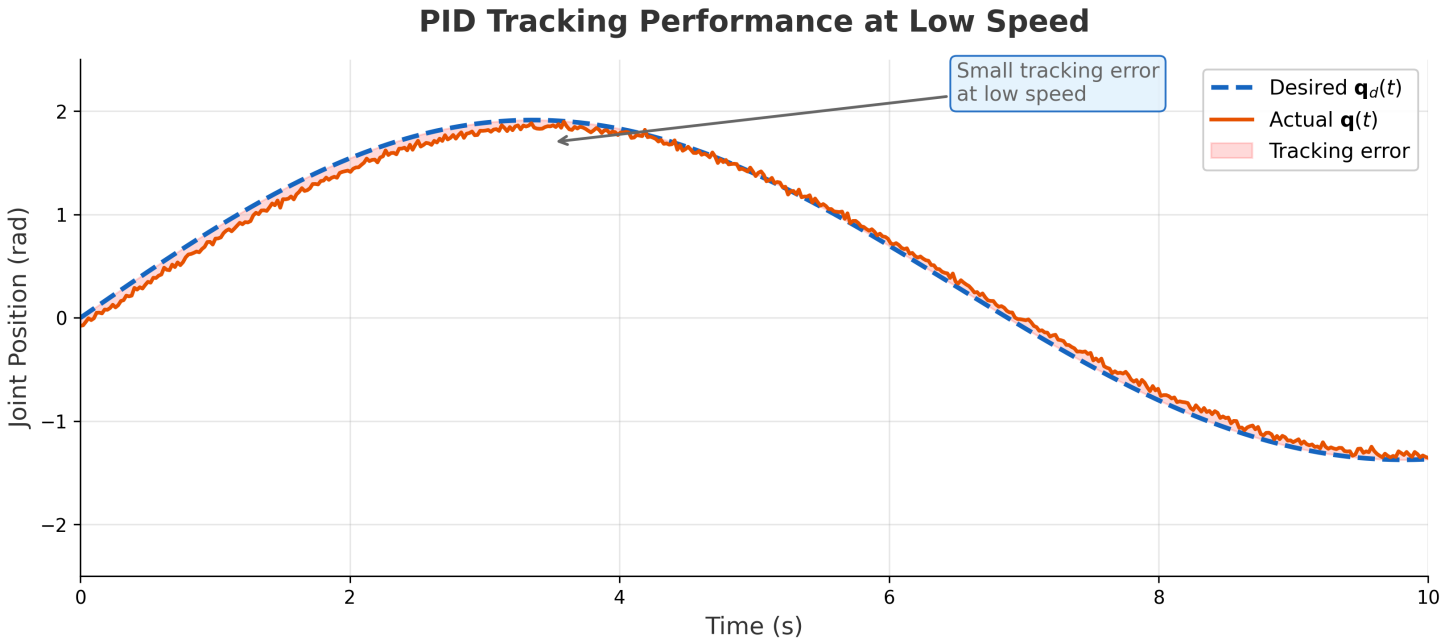
$$\boldsymbol{\tau} = \mathbf{K}_p \mathbf{e} + \mathbf{K}_i \int_0^t \mathbf{e}(\sigma) d\sigma + \mathbf{K}_d \dot{\mathbf{e}}$$

Intuition: Kp·e = spring (pulls toward target) • Kd·ė = damper (resists velocity) • Ki·∫e = accumulated push (fixes offset)

Key Point: With diagonal gain matrices, PID controls each joint independently—as if they were n separate single-joint systems. This is simple but ignores that real robot joints are physically coupled...

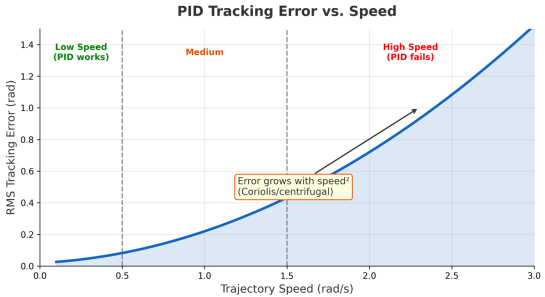
PID Performance: Slow Motion

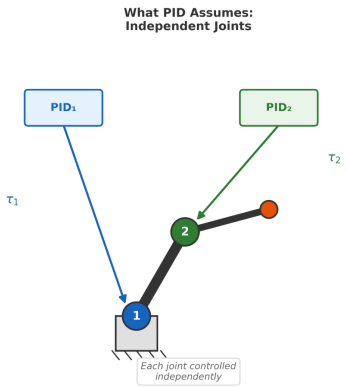
At low speeds, PID tracking is acceptable

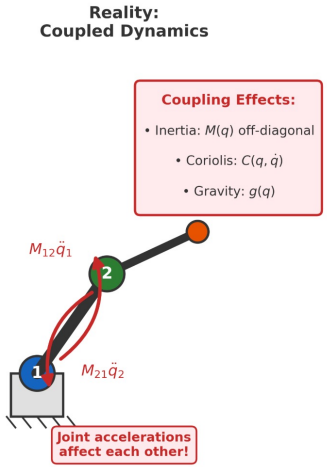


Key Observation: When dynamics are slow and loads are light, nonlinearities don't dominate. PID works.

When PID Fails: The Coupling Problem







Why does this happen?

1. **Inertia Coupling:** Joint 1 acceleration creates torques on Joint 2
2. **Coriolis/Centrifugal:** Velocity-dependent forces grow with speed
3. **Config. Dependence:** Same τ → different q̈ at different q
4. **Gravity Variation:** Gravity load changes with arm configuration

The Fundamental Problem

PID uses diagonal gain matrices, treating each joint independently.
But joints are physically coupled through dynamics: $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{g}(\mathbf{q})$ create cross-joint effects that independent controllers cannot compensate.

The Key Insight: Use the Model

Recall from Lecture 2: The Manipulator Dynamics Equation

Recall: $\mathbf{q}$ = positions, $\dot{\mathbf{q}}$ = velocities. New: $\ddot{\mathbf{q}}$ = accelerations, all $\in \mathbb{R}^n$

Manipulator Dynamics

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}$$

$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}$ = Inertial forces (n×n mass matrix × acceleration)

$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$ = Coriolis & centrifugal forces (velocity-dependent)

$\mathbf{g}(\mathbf{q})$ = Gravity forces (configuration-dependent)

Key Insight: Model-Based Cancellation

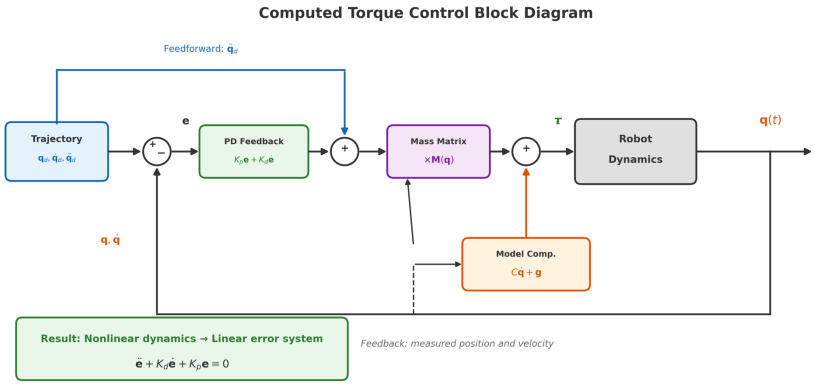
If we know $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{g}(\mathbf{q})$, we can choose $\boldsymbol{\tau}$ to **cancel these nonlinear terms!**
Instead of fighting the dynamics, we use them.
This is the core idea behind **computed torque control** (also called inverse dynamics control or feedback linearization).

Computed Torque Control

$\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$ = desired position, velocity, acceleration (from trajectory planner)

Computed Torque Control Law

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})[\ddot{\mathbf{q}}_d + \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}})] + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q})$$



Structure: $\boldsymbol{\tau}$ = M(q)·[feedforward + feedback] + model compensation

Resulting Error Dynamics (with substitution, define $\mathbf{e} = \mathbf{q}_d - \mathbf{q}$)

$$\ddot{\mathbf{e}} + \mathbf{K}_d\dot{\mathbf{e}} + \mathbf{K}_p\mathbf{e} = \mathbf{0} \quad \leftarrow \texttt{Linear 2nd-order!}$$

Why This Matters (Feedback Linearization)

The nonlinear robot dynamics become a simple linear error system.
Standard linear control theory now applies: choose $\mathbf{K}_p, \mathbf{K}_d$ for desired damping ratio ζ and natural frequency ω_n.
Error converges exponentially—works at any speed if model is accurate.

Computed Torque: Trade-offs

✓ Advantages

- Achieves linear error dynamics
- Works at high speeds (if model is accurate)
- Principled: uses physics directly
- Full matrix gains decouple joint control

✗ Limitations

- Requires accurate dynamics model
- Sensitive to model errors (robustness)
- Computationally expensive (real-time)
- No handling of unexpected contacts

Learning Connection

Model-based RL follows the same logic: use known physics to simplify learning. Methods like MBPO learn a dynamics model, then plan through it. Physics-informed neural networks embed the dynamics equation structure. The insight is universal: the better your model, the less your learning must compensate.

But what about contacts? *Computed torque assumes free-space motion. When the robot touches the environment, we need a different approach...*

Impedance Control

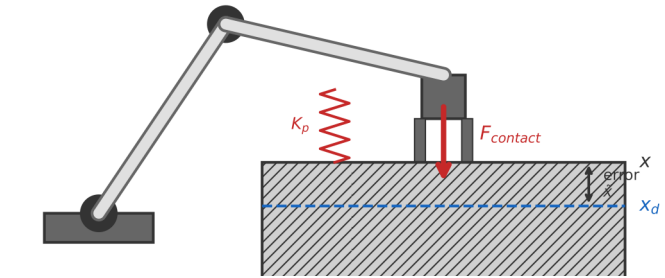
Controlling Interaction, Not Just Motion

Key Question: How should a robot behave when it contacts the environment?

The Problem with Position Control at Contact

Scenario: Robot arm approaching a surface to perform an assembly task

Controller tries to reach x_d, generating large contact force



In a **position controller** (like PID), the control law is:

$$\tau = K_p(x_d - x) + K_d(\dot{x}_d - \dot{x}) + K_i \int (x_d - x)dt$$

What happens?

1. Robot commanded to position xd

2. Surface is at x < xd

3. Position controller fights to reach xd

4. Result: F = Kp(xd - x) grows!

The Danger

Position control treats any deviation from the commanded position as an error to be corrected.
At contact, this means the robot pushes harder and harder against the environment.
This can damage the robot, the workpiece, or the environment—and is dangerous for human collaboration.

Two Approaches to Contact

Force Control

Goal: Regulate contact force F

Method: Measure F, adjust position to maintain Fd

Problem: Unstable in free space! When F = 0, controller has no feedback.

Use case: Polishing, grinding (always in contact)

Impedance Control

Goal: Regulate F-x relationship

Method: Behave like a virtual spring-damper system

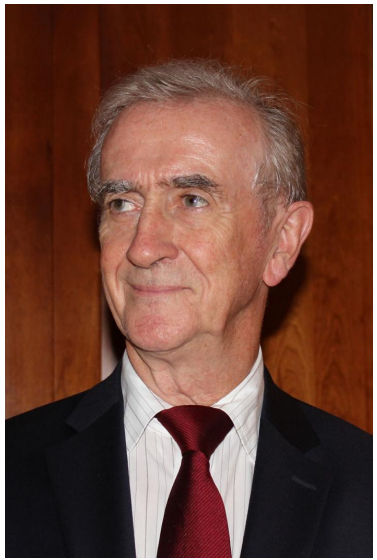
Advantage: Works in free space AND at contact!

Use case: Assembly, human collaboration, uncertain contact

Key Distinction

Force control specifies what force to apply.
Impedance control specifies how force should relate to position error—the robot's mechanical behavior.
This is fundamentally about controlling dynamics, not just statics.

Hogan's Fundamental Insight (1985)



Key Insight

"Manipulation fundamentally requires the control of dynamic behavior—the controlled system's response to externally applied forces. Neither motion control alone nor force control alone can accomplish this."

Historical Context: Published as a trilogy in ASME Journal of Dynamic Systems (1985). Introduced impedance control as a unified framework for manipulation. Still the foundation of compliant manipulation 40 years later.

The Implication: Instead of controlling position or force, we should control the relationship between them—the impedance. The robot should behave like a physical system with chosen mass, damping, and stiffness properties.

The Impedance Control Law

Variable Definitions

Symbol	Domain	Meaning
$\tilde{x} = x - x_d$	$\mathbb{R}^3$ or $\mathbb{R}^6$	Position error (actual minus desired)
M_d	$\mathbb{R}^{n \times n}$	Desired inertia matrix
B_d	$\mathbb{R}^{n \times n}$	Desired damping matrix
K_d	$\mathbb{R}^{n \times n}$	Desired stiffness matrix
F_{ext}	$\mathbb{R}^3$ or $\mathbb{R}^6$	External force/torque from environment

Desired Impedance Behavior

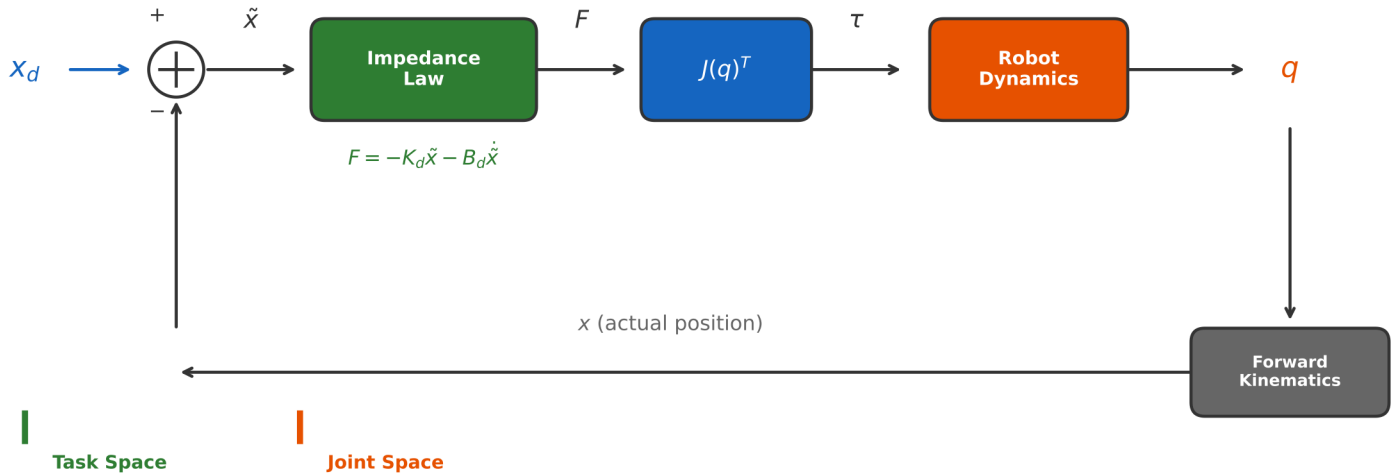
$$M_d \ddot{\tilde{x}} + B_d \dot{\tilde{x}} + K_d \tilde{x} = F_{\text{ext}}$$

Interpretation: The robot should respond to external forces as if it were a mass-spring-damper system with parameters Md, Bd, Kd that we choose.

Physical Analogy: Kd = virtual spring stiffness (position error → force), Bd = virtual damping (velocity → force), Md = virtual inertia (acceleration → force).
We're programming the robot to feel like a mechanical system of our choosing.

Implementing Impedance Control

Joint-space implementation using the Jacobian



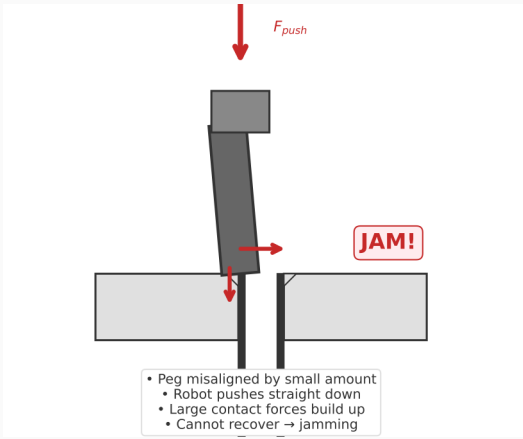
Implementation Steps

1. Compute position error in task space: x̃ = x − xd
2. Compute desired task-space force: F = −Kd·x̃ − Bd·x̃̇ (+ optional inertia shaping)
3. Map to joint torques: τ = J(q)ᵀ·F (Jacobian transpose maps task forces to joint torques)

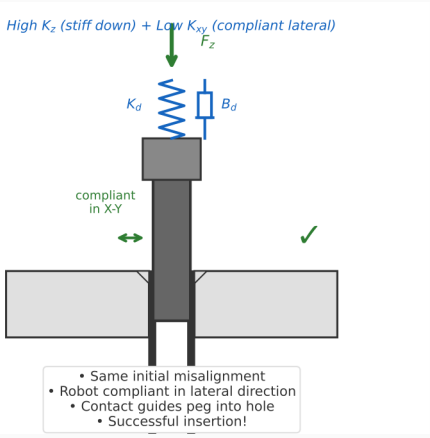
Example: Peg-in-Hole Insertion

The classic benchmark for contact-rich manipulation

Position Control: FAILS

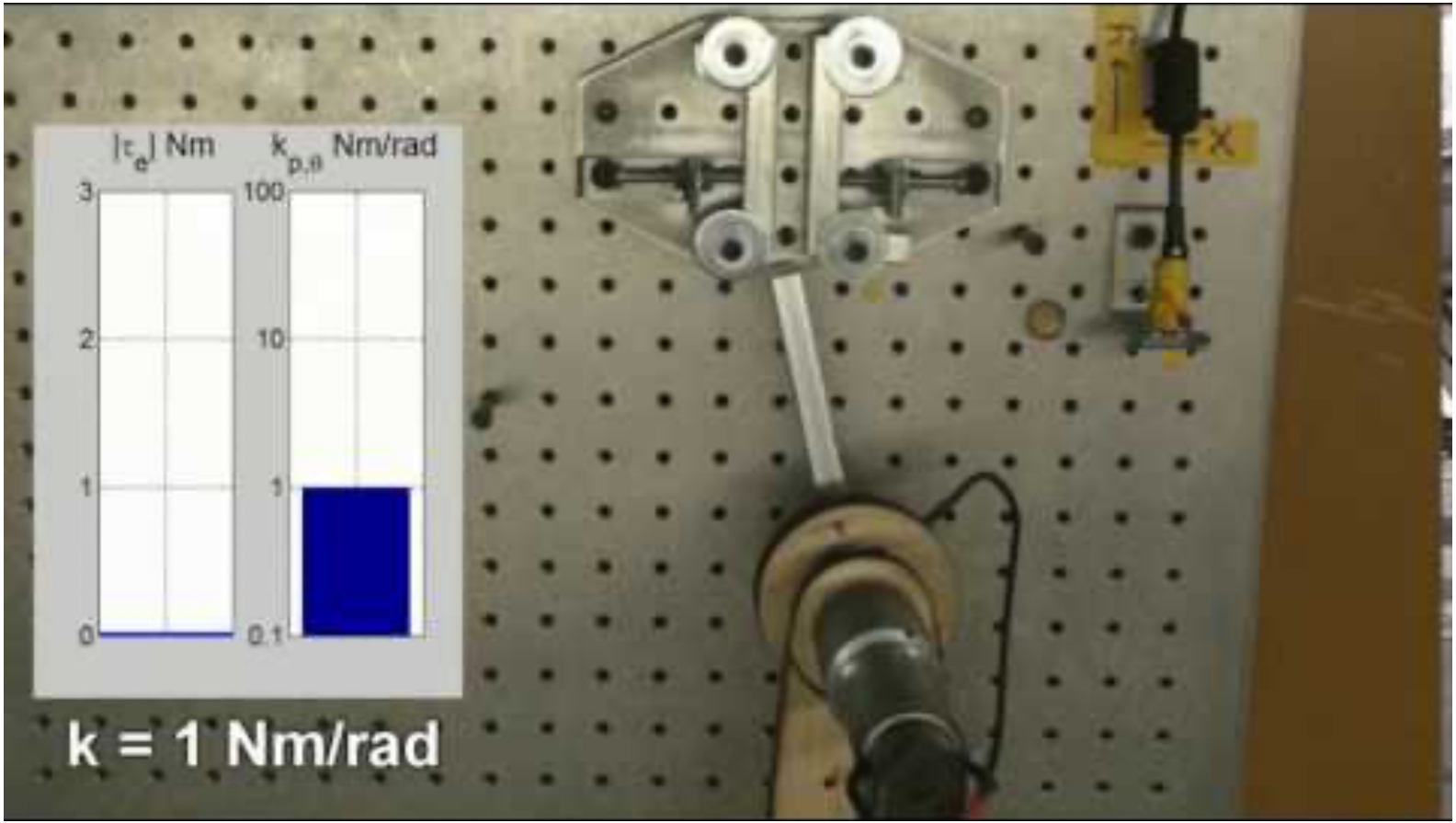


Impedance Control: SUCCEEDS



Why It Works

Directional stiffness: High stiffness in Z (insertion direction) maintains downward force.
Low stiffness in X-Y (lateral) allows the peg to comply with the hole geometry.
The environment itself guides the peg into place—we use contact rather than fighting it.

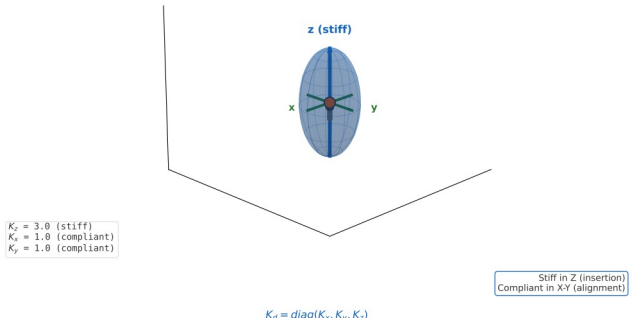

|τe| Nm
kp,θ Nm/rad
3
2
1
0
100
10
1
0.1
k = 1 Nm/rad

Choosing Impedance Parameters

The Stiffness Trade-off

High Stiffness (large Kd)
✓ Accurate positioning
✓ Precise trajectory tracking
✗ High contact forces
✗ Fragile at unexpected contact

Low Stiffness (small Kd)
✓ Safe at contact
✓ Robust to uncertainty
✗ Imprecise positioning
✗ Susceptible to disturbances



Directional Stiffness

The key insight is that Kd can be a matrix, not just a scalar. This lets us be:
• Stiff in task-critical directions
• Compliant in contact directions

Example: For peg-in-hole, Kz >> Kx, Ky

Learning Connection: Diffusion Policy exhibits impedance-like behavior—compliant near contacts because training data shows compliant human demos. This is why it succeeds where MSE-based BC fails. (Why?)

$$\mathcal{L}_{\text{MSE}} = \mathbb{E}\left[\|\pi_\theta(s) - a_{\text{demo}}\|^2\right]$$

Operational Space Control

Thinking in Task Space

Oussama Khatib, 1987

Joint Space vs. Task Space

We specify tasks in task space but command motors in joint space

Joint Space

Variables: $q \in \mathbb{R}^n$ (joint angles)

Control: $\tau \in \mathbb{R}^n$ (joint torques)

Perspective: How each motor should move

Task Space (Operational Space)

Variables: $x \in \mathbb{R}^m$ (end-effector pose)

Control: $F \in \mathbb{R}^m$ (task-space force)

Perspective: What the end-effector should do

The Connection: Jacobian

$$\dot{x} = J(q)\dot{q} \quad \text{and} \quad \tau = J^T(q)F$$

Khatib's Insight: We can formulate dynamics and control directly in task space—no need to convert back and forth!

Operational Space Dynamics

From joint-space dynamics to task-space dynamics

Starting Point: Joint-Space Dynamics

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau$$

↓ Transform using $x = f(q)$, $\dot{x} = J(q)\dot{q}$, $\tau = J^T F$

Symbol	Definition	Meaning
$\Lambda(x)$	$(JM^{-1}J^\top)^{-1}$	Task-space inertia matrix
$\mu(x,\dot{x})$	$\Lambda(JM^{-1}C - \dot{J})\dot{q}$	Task-space Coriolis/centrifugal forces
$p(x)$	$\Lambda JM^{-1}g$	Task-space gravity vector
F	$\in \mathbb{R}^m$	Task-space force/torque vector

Symbol	Meaning
J	Jacobian matrix: $\dot{x} = J(q)\dot{q}$
M	Joint-space mass/inertia matrix
C	Joint-space Coriolis matrix
g	Joint-space gravity vector
$\dot{J}$	Time derivative of the Jacobian
x	Task-space position (end-effector pose)
$\ddot{x}$	Task-space acceleration

Operational Space Dynamics

$$\Lambda(x)\ddot{x} + \mu(x,\dot{x}) + p(x) = F$$

Key: Same structure as joint-space dynamics, but expressed in task-space variables!

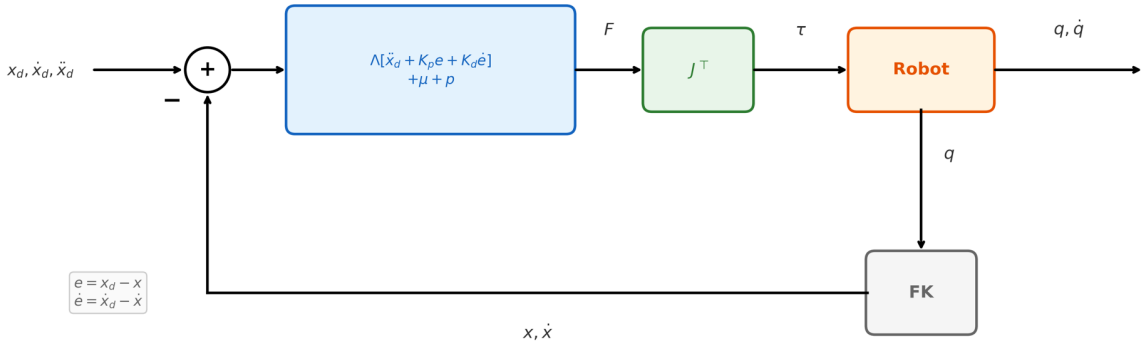
Operational Space Control Law

Computed torque in task space: specify what the end-effector should do

Operational Space Control

$$F = \Lambda(x)\left[\ddot{x}_d + K_p(x_d - x) + K_d(\dot{x}_d - \dot{x})\right] + \mu(x, \dot{x}) + p(x)$$

Then map to joint torques: $\tau = J^\top(q)F$



Compare to Computed Torque: Same idea—cancel nonlinearities, achieve linear error dynamics—but now in task space. Specify *where the hand should go*, not how each joint should move.

Handling Redundancy

7-DOF arms: more joints than needed for 6-DOF task-space control

The Situation

- Task space: 6 DOF (position + orientation)
- Joint space: 7 DOF (e.g., Franka Panda)
- Redundancy: ∞ joint configurations for same end-effector pose
- Question: How to use this freedom?

Null-Space Motion

Null space = joint motions that don't affect end-effector
Use it for:

- Avoid joint limits
- Avoid obstacles
- Minimize energy

$$\tau = \underbrace{J^\top F}_{\text{primary task}} + \underbrace{N(q)\tau_0}_{\text{null-space task}}$$

Full Control Law with Null-Space Optimization

$$\tau = J^\top F + \left(I - J^\top (J^+)^\top\right)\tau_0$$

where τ_0 = secondary objective (e.g., posture control)

Learning Connection

RT-1, Octo, and OpenVLA output end-effector poses (position + orientation), not joint angles.
This is exactly operational space thinking—specify what the end-effector should do, let the low-level controller determine how the joints achieve it. The classical framework enables learned high-level policies to be embodiment-agnostic.

Locomotion Control

The Challenge of Walking

Key Question: How do we control a robot that spends its time falling and catching itself?

Why Locomotion is Different

Manipulation vs. locomotion: fundamentally different control challenges

Manipulation	Locomotion
Base: Fixed (bolted down)	**Base:** Floating (free in space)
Actuation: Fully actuated	**Actuation:** Underactuated!
Contacts: Continuous (gripper)	**Contacts:** Switching (foot strikes)
CoM Control: Direct (via joints)	**CoM Control:** Indirect (via contacts)
Stability: Static (always stable)	**Stability:** Dynamic (actively maintained)

The Core Challenge

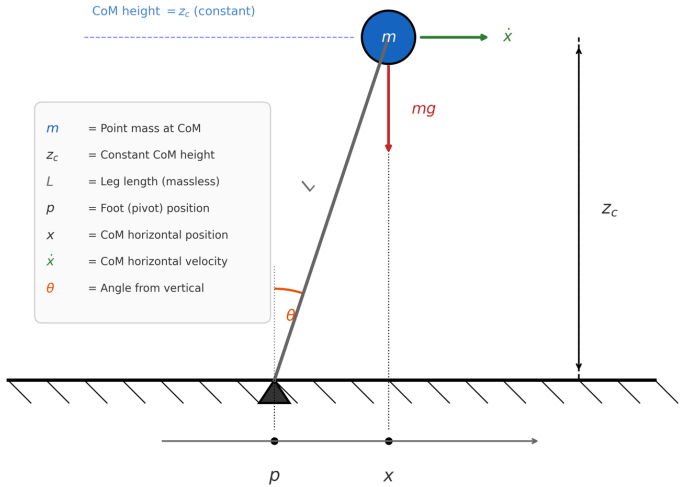
We cannot directly command the robot's center of mass (CoM)—we can only control it indirectly through contact forces with the ground. The robot is constantly in a state of controlled falling. This requires completely different control strategies than manipulation.

The Linear Inverted Pendulum Model (LIPM)

The simplest useful model of bipedal walking

Simplifying Assumptions

1. Point mass at CoM
   (ignore body rotation)
2. Massless legs
   (no leg inertia)
3. CoM moves on horizontal plane
   (constant height zc)
4. Point contact at foot
   (no foot rotation)



Why This Model?

Despite these drastic simplifications, LIPM captures the essential dynamics of walking:
- The unstable nature of bipedal balance
- The relationship between CoM and foot placement
- The fundamental physics that all walking robots must obey

This model underlies most classical walking control methods (ZMP, Capture Point).

LIPM Dynamics: Unstable Equilibrium

The mathematics of controlled falling

Variable Definitions

Symbol	Units	Meaning
x	m	Horizontal position of Center of Mass (CoM)
$\ddot{x}$	$\mathrm{m/s^2}$	Horizontal acceleration of CoM
p	m	Foot (pivot) position on ground
z_c	m	Constant height of CoM above ground
g	$\mathrm{m/s^2}$	Gravitational acceleration (≈ 9.81)
ω_0	rad/s	Natural frequency of the inverted pendulum

LIPM Dynamics

$$\ddot{x} = \omega_0^2(x - p)$$

Critical Insight: This System is Unstable!

- When x > p (CoM ahead of foot): ẍ > 0 → accelerates forward → falls forward
- When x < p (CoM behind foot): ẍ < 0 → accelerates backward → falls backward
- Only at x = p is there equilibrium, but it's unstable (inverted pendulum)

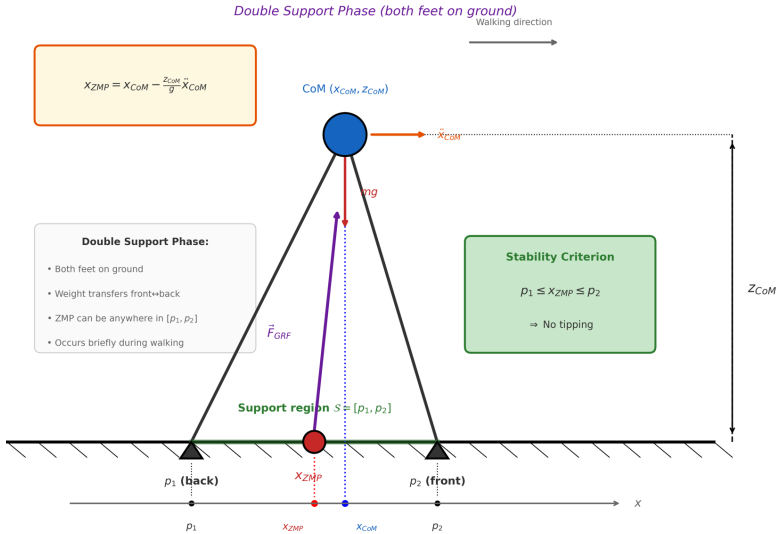
Walking = continuously moving p (foot placement) to manage this instability

The Zero Moment Point (ZMP)

Where the ground reaction force must act for stability

Definition

The ZMP is the point on the ground where the total moment of all forces (gravity + inertia) about the horizontal axes equals zero. **Physical meaning:** It's where the ground reaction force vector intersects the ground plane.



ZMP Location (2D)

$$x_{ZMP} = x_{CoM} - \frac{z_{CoM}}{g}\ddot{x}_{CoM}$$

ZMP Stability Criterion

If ZMP stays inside the support polygon → robot won't tip over
If ZMP reaches the edge → robot is about to tip
If ZMP would go outside → physically impossible → robot falls
Walking control goal: Plan CoM trajectory so ZMP stays safely inside the support polygon

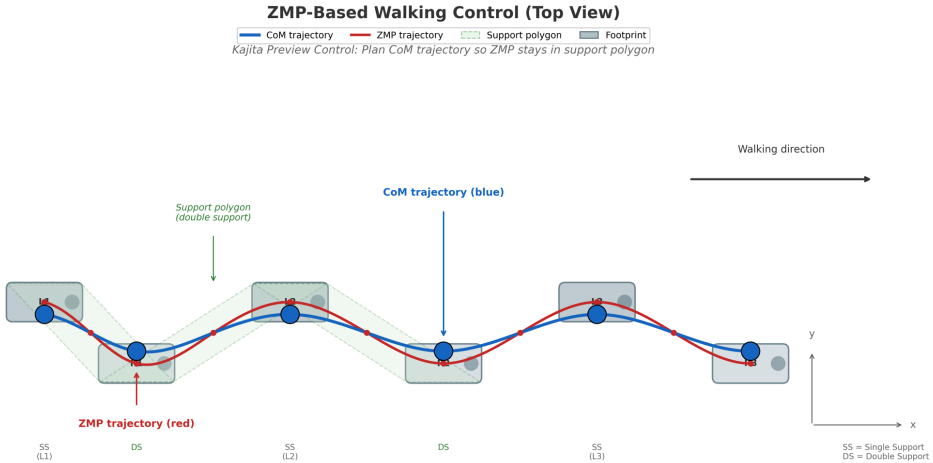
Symbol	Units	Meaning
x_{ZMP}	m	Horizontal position of Zero Moment Point
x_{CoM}	m	Horizontal position of Center of Mass
$\ddot{x}_{CoM}$	m/s^2	Horizontal acceleration of Center of Mass
z_{CoM}	m	Vertical height of Center of Mass
g	m/s^2	Gravitational acceleration (≈ 9.81)

ZMP-Based Walking Control

Kajita's preview control (2003) — the foundation of industrial humanoids

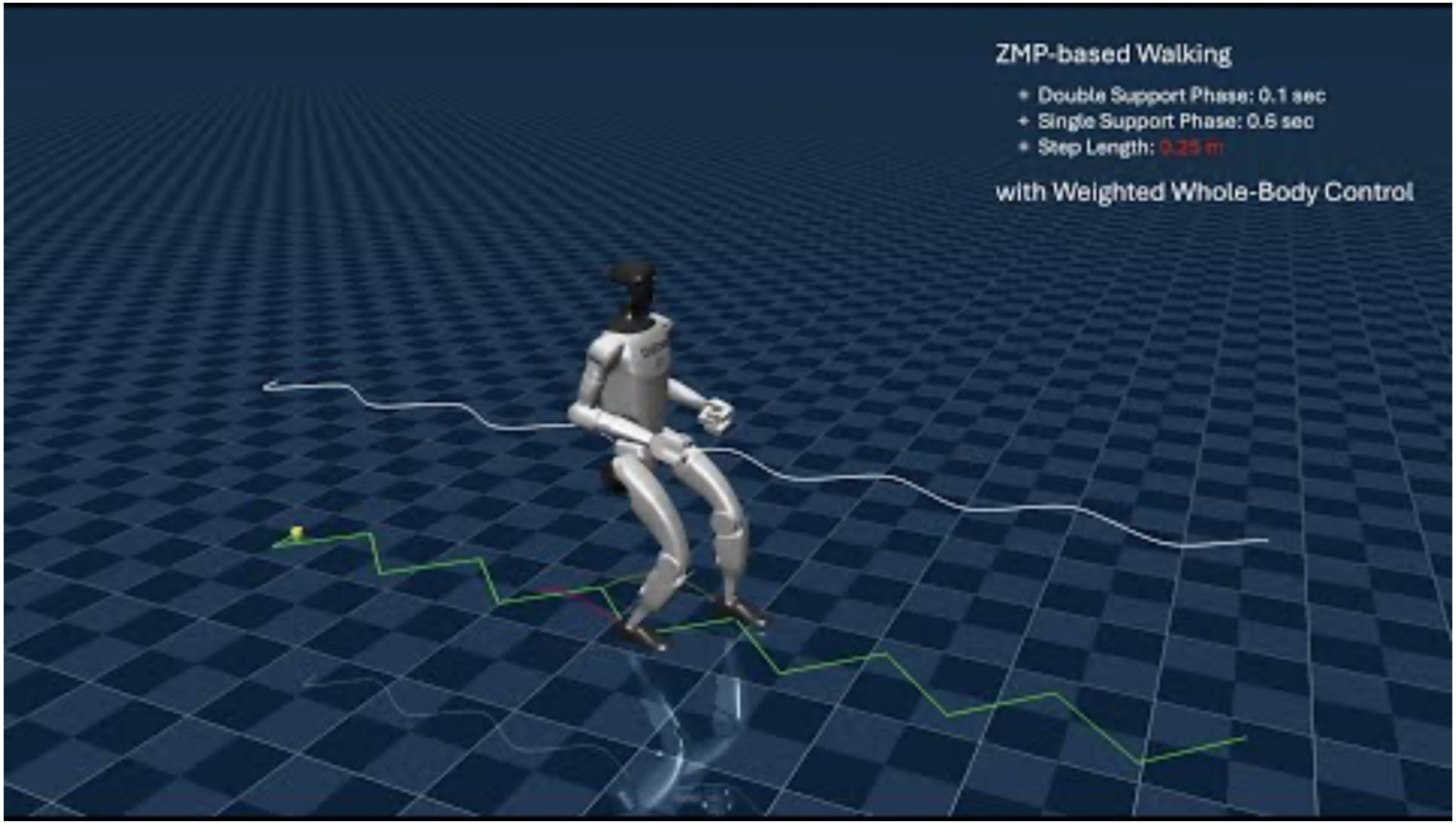
The Approach

1. Define desired footstep sequence (where feet will land)
2. Define desired ZMP trajectory (stays in support polygon)
3. Solve for CoM trajectory that produces this ZMP trajectory
4. Track CoM with whole-body control



Success Stories: Honda ASIMO, HRP series, early humanoids. This approach dominated industrial humanoid walking from 2000-2015.

But there's a problem... What happens when the robot gets pushed unexpectedly? The pre-planned trajectory can't adapt fast enough.

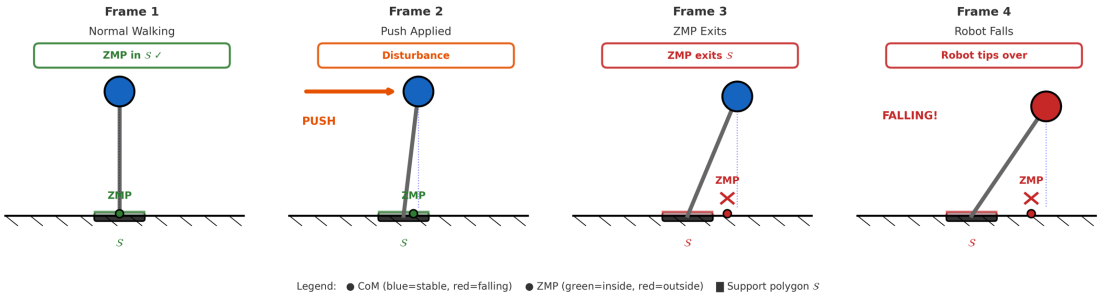

ZMP-based Walking
Double Support Phase: 0.1 sec
Single Support Phase: 0.6 sec
Step Length: 0.25 m
with Weighted Whole-Body Control

Limitations of ZMP-Based Control

Why we need reactive balance strategies

Key Limitations

1. **Requires pre-planned trajectories**
   Cannot react in real-time to disturbances
2. **Limited disturbance rejection**
   Small pushes OK, large pushes → fall
3. **Conservative walking**
   Must keep ZMP well inside polygon
4. **No stepping reaction**
   Can't take recovery step if pushed



The Key Question

When you're falling, where should you put your foot to recover?
This is exactly the question Capture Point theory answers...

Capture Point: Where to Step

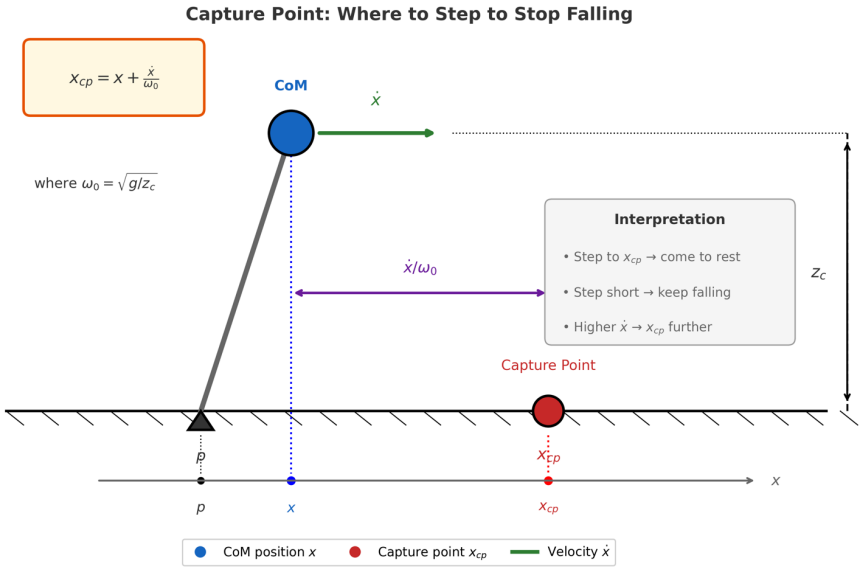
Pratt et al., 2006 — A simple answer to a critical question

The Idea

The **Capture Point** is the location where, if you place your foot instantaneously, you will come to a stop without falling.
Intuition: It's "where the CoM is heading" accounting for current velocity.

Capture Point (for LIPM)

$$x_{cp} = x + \frac{\dot{x}}{\omega_0}$$



Interpretation

- Capture point = current position + velocity scaled by pendulum time constant
- Higher velocity → capture point further ahead
- Step exactly to capture point → come to rest. Step short → continue falling forward.

Physical Interpretation

When you place your foot exactly at the capture point, you eliminate the divergent mode of the inverted pendulum. The solution becomes:

$$x(t) = x_{cp} + Be^{-\omega_0 t}$$

As $t \to \infty$, the CoM asymptotically approaches x_{cp} and comes to rest directly above the foot. The robot stops falling without any additional control effort.

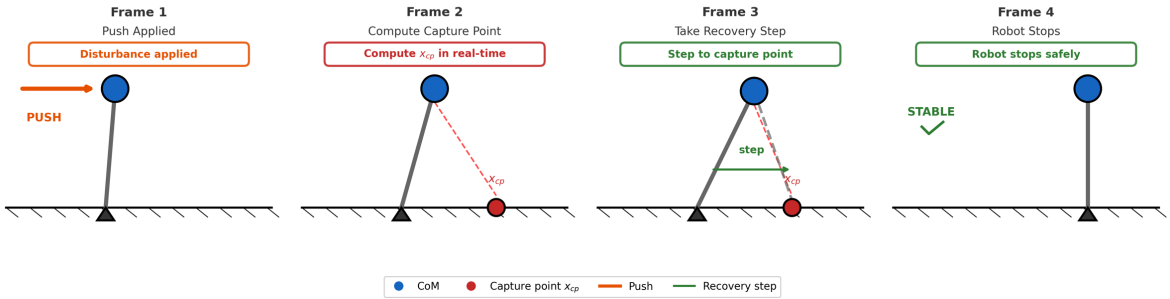
Capture Point for Push Recovery

Reactive balance without replanning

The Recovery Strategy

1. Robot gets pushed
2. Compute capture point in real-time
3. Take recovery step to capture point
4. Robot comes to stable stop

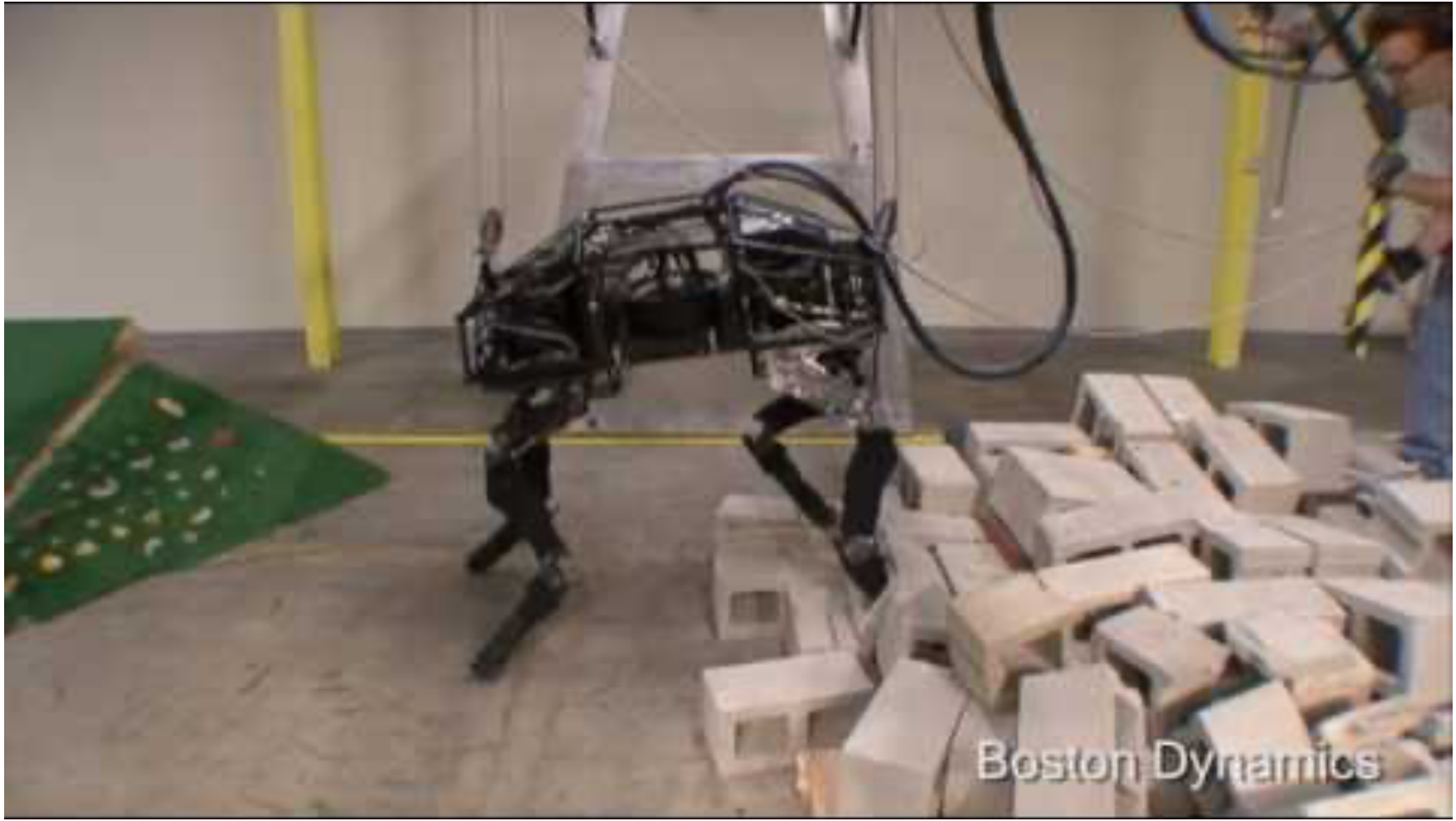
No trajectory replanning needed! Just simple geometry based on current state.



Extensions: Capture regions (where to step given bounds on step size), instantaneous capture point dynamics, 3D capture point for full humanoids. Forms the basis of many modern humanoid controllers.

Learning Connection

Learned locomotion policies must implicitly satisfy these stability criteria. When we see a learned policy recover from a push, it has discovered something like capture point control—placing the foot where it needs to go to arrest the fall. Understanding ZMP and capture point helps us interpret what the network has learned and diagnose failure modes.


Boston Dynamics

Model Predictive Control

Optimization Meets Real-Time Control

Key Question: How do we combine trajectory optimization with feedback control for real-time performance?

The MPC Concept

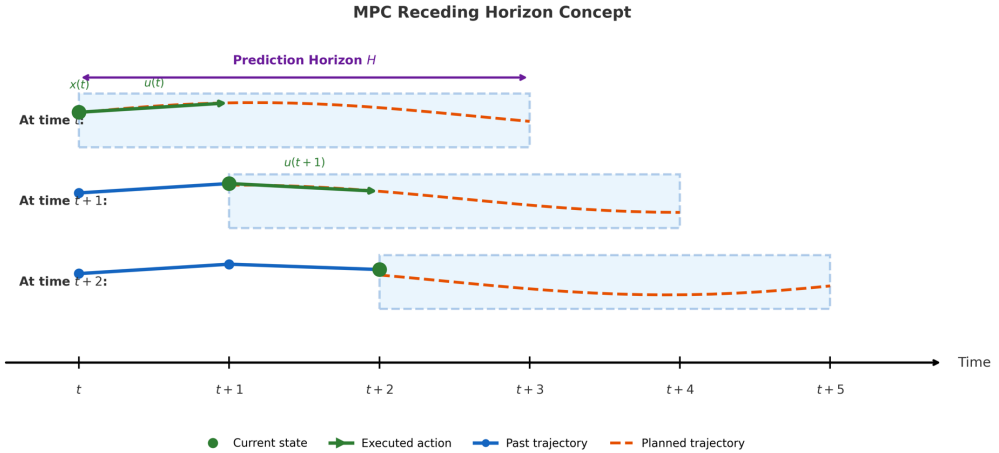
Optimize, execute, repeat — the receding horizon approach

The MPC Algorithm

At each timestep:

1. Measure current state x(t)
2. Optimize trajectory over horizon [t, t+H]
3. Execute only the first action u(t)
4. Repeat at next timestep

Key insight: Re-plan constantly with updated state information. Never fully commit to a long trajectory.



Why MPC Works

- Handles constraints naturally (joint limits, torque limits, obstacle avoidance)
- Incorporates dynamics model (predicts future states)
- Provides feedback (re-plans from actual state, not planned state)
- Bridges planning and control (optimization at control frequency)

Convex MPC for Quadrupeds

MIT Cheetah 3 (Di Carlo et al., 2018) — State of the art in practice

The Key Insight

Simplify the model to make optimization fast:
- Treat body as single rigid body
- Ignore leg dynamics (massless legs)
- Linearize rotation dynamics
- Result: Convex QP solvable in <1ms!

Fast enough to run at 500Hz control rate

Results: Running at 3 m/s, jumping over obstacles, recovering from pushes—all using this simplified convex MPC. Now the standard approach for quadruped locomotion (Unitree, ANYmal, etc.).

Why This Matters: Shows that the right model simplification can make optimization-based control practical. This is the baseline that learning-based methods now try to match or exceed.

MPC: Limitations and Opportunities

Where learning can help

Limitations

- Requires accurate model
  (model errors → suboptimal control)
- Computational cost
  (limits horizon length, update rate)
- Struggles with contacts
  (discontinuities in dynamics)

Learning Opportunities

- Learn better models
  (neural network dynamics)
- Learn to warm-start
  (predict good initial solutions)
- Learn residual corrections
  (MPC baseline + learned adjustment)

Learning Connection: Hybrid MPC Approaches

Neural MPC: Train a network to approximate the MPC solution. 1000x faster inference, but loses guarantees.
Residual RL + MPC: Run MPC as baseline, learn a policy that adds corrections. Gets MPC's structure + learning's adaptability.
Learned Dynamics + MPC: Use neural network for dynamics model inside MPC. Better predictions, same optimization framework.

Classical Meets Learning

Hybrid Architectures for Embodied AI

Key Question: How do we combine the guarantees of classical control with the adaptability of learning?

Why Hybrid Architectures?

Neither pure classical nor pure learning is sufficient alone

Pure Classical	Hybrid	Pure Learning
✓ Stability guarantees	✓ Physics-based structure	✓ Handles complexity
✓ Interpretable	✓ Learned adaptability	✓ Adapts to reality
✓ Sample efficient	✓ Faster training	✓ End-to-end optimization
✗ Brittle to model errors	**Best of both worlds!**	✗ Sample inefficient
✗ Hard to tune	Classical provides baseline	✗ No guarantees
✗ Cannot adapt online	Learning fills the gaps	✗ Hard to debug

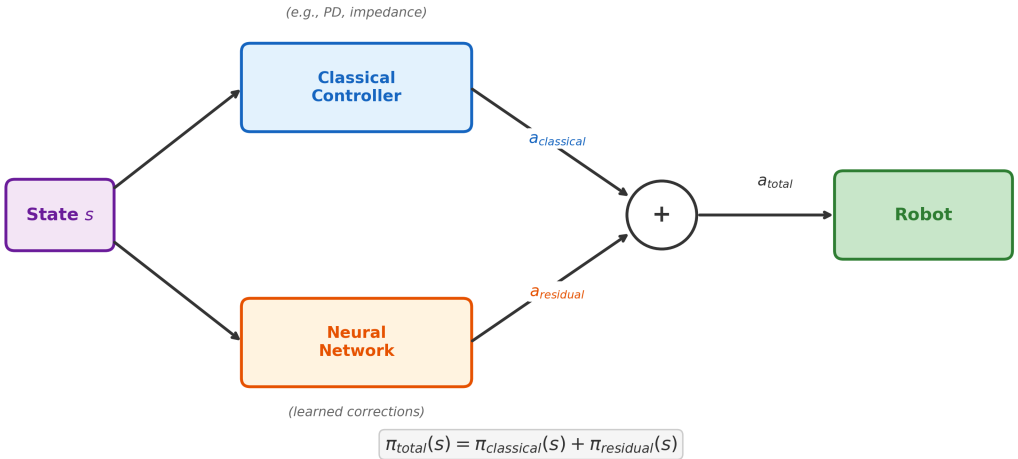
The Key Insight: Use physics-based structure where you understand the system, learn the parts you don't. Classical control provides a strong prior that makes learning faster, safer, and more generalizable.

Residual Policy Learning

Classical baseline + learned corrections

The Residual Policy Equation

$$\pi_{total}(s) = \pi_{classical}(s) + \pi_{residual}(s)$$



Why This Works

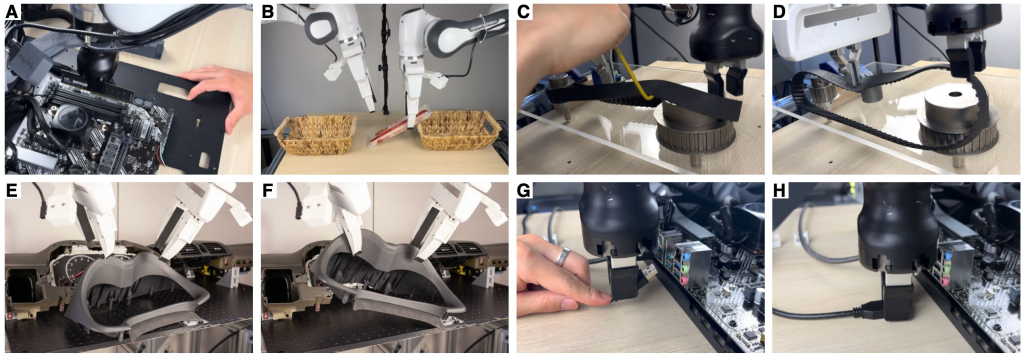
- Classical policy handles 80% of the problem → network only learns the 20% that's hard to model
- Training starts from competent behavior → faster convergence, safer exploration
- If learned policy fails, classical baseline provides safety fallback

HIL-SERL: Residual RL in Practice

Luo et al., Science Robotics 2024 — Near-perfect assembly in 1-2.5 hours

The Approach

1. Start with impedance controller baseline (compliant, safe, but imprecise)
2. Learn residual corrections online (human-in-the-loop for resets)
3. Achieve sub-mm precision assembly (impossible with impedance alone)



Results

Task	Training Time	Success Rate
PCB Insertion (0.2mm tolerance)	1 hour	Near 100%
Cable Routing	2 hours	Near 100%
Dynamic Handover	2.5 hours	Near 100%

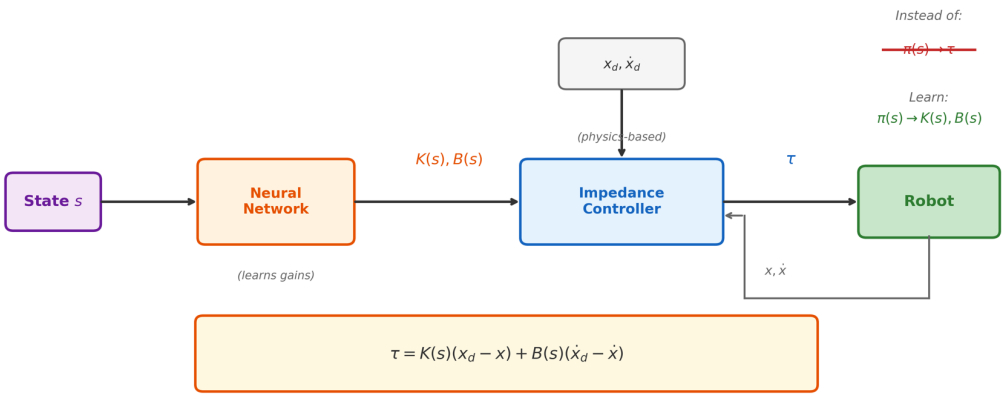
Key: Impedance control baseline made learning tractable. Without it, these tasks would need millions of samples.

Learned Impedance Parameters

Network outputs stiffness and damping, not torques

The Idea

Instead of: π(s) → τ (raw torques)
Learn: π(s) → K(s), B(s) (impedance gains)
Then use impedance controller:
τ = K(s)(xd − x) + B(s)(ẋd − ẋ)



Why This Is Better Than Learning Raw Torques

1. **Inherent compliance:** Robot is always compliant—safe by construction, not by training
2. **Structured output space:** K > 0, B > 0 constraints easy to enforce → always stable
3. **Interpretable:** Can inspect what stiffness/damping the network chooses in different situations
4. **Generalizable:** Same gains often work across similar tasks (transfer learning)

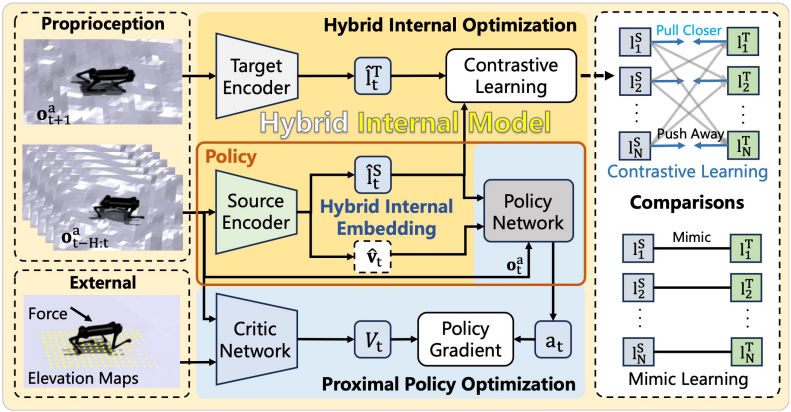
Hybrid Internal Model (HIM)

Long et al., ICLR 2024 — 1 hour of real-world training

Internal Model Control Principle

Classical idea (from process control):
1. Build internal model of the scene
2. Use model inverse for feedforward
3. Correct for model errors with feedback

HIM: Learn components 1-3 from data!



Results: Quadruped locomotion learned in ~1 hour of real-world training. Matches performance of policies trained with millions of simulation steps. Key: classical structure reduces what needs to be learned.

Why Classical Structure Helps Learning

The IMC structure separates: (1) learning the dynamics model, (2) computing the control from the model. Each subproblem is easier than learning end-to-end. The structure also provides inductive bias: the network learns "corrections" rather than "everything from scratch."

Sim-to-Real: Classical Control as Anchor

Reducing the reality gap with physics-based structure

The Sim-to-Real Problem

End-to-end policies trained in sim:
- Learn simulator quirks
- Exploit unrealistic physics
- Break in subtle ways on real robot

The "reality gap" limits transfer

Classical Control as Anchor

Task-space policy + classical low-level:
- Policy outputs end-effector goals
- Classical IK/control handles execution
- Same low-level works in sim and real!

Smaller gap = better transfer

Example: TRANSIC Approach

Train VLA in simulation to output task-space poses. Deploy with operational space control on real robot.
The classical controller "anchors" the policy to real physics—handles motor dynamics, friction, contact forces that simulation gets wrong.

Key Insight: The more physics you encode in classical structure, the less the learned policy needs to compensate for sim-to-real mismatch. Abstraction barriers help!

Real-to-Sim

Sim-to-Real

HELIX 02

SYSTEM 2 — 7B

Scene understanding, language comprehension

SYSTEM 1 — 80M

All sensors to all joints visuomotor learning

SYSTEM 0 — 10M

Human-like, stable motion tracking

1kHz

The Design Principle

A framework for hybrid control architectures

Classical Structure + Learned Parameters/Corrections

Use physics-based structure where you understand the system.
Learn the parts that are hard to model.

Examples We've Seen

Approach	Classical Structure	Learned Component
Residual Policy	Impedance/MPC baseline	Correction term
Learned Impedance	Impedance control law	K(s), B(s) parameters
HIM	Internal Model Control	Dynamics model, inverse
VLA + Op-Space	Operational space control	Task-space policy

This is why understanding classical control matters for modern embodied AI!

Summary: What We Learned

Reviewing the 7 learning objectives

1. **PID Control:** Simple but limited—works for slow, decoupled motion
2. **Computed Torque:** Model-based cancellation enables linear error dynamics
3. **Impedance Control:** Regulate force-motion relationship, essential for contact
4. **Operational Space:** Control in task space where policies naturally operate
5. **LIPM, ZMP, Capture Point:** Fundamental physics of bipedal balance
6. **MPC:** Optimization meets real-time control, state of practice
7. **Hybrid Architectures:** Classical structure + learned components = best of both

Big Picture: Classical control provides the foundation that makes modern learning-based approaches practical and safe.

Connections Forward

Where these concepts reappear in the course

Lecture 4: Planning & Navigation

MPC returns as motion planning backbone; operational space for manipulation planning

Lecture 7: Reinforcement Learning

Residual policies, reward shaping with classical controllers, impedance-based RL

Lecture 9: Diffusion Policies

Action representations (task-space vs. joint-space), control frequency considerations

Lecture 12: Humanoid Robots

Full-body control, ZMP/capture point, whole-body MPC

Q & A