॥ श्री गणेशाय नमः ॥

Optimal Control

Varad Vaidya

November 29, 2025

Abstract

Contents

Lecture 17: Iterative Learning Control (ILC)

We address a fundamental reality in robotics: **models are never perfect**. Even with the best physics engines and parameter identification, there will always be a "sim-to-real" gap due to unmodeled friction, aerodynamic effects, flexibility, or sensor noise. We will explore strategies to handle these errors, culminating in **Iterative Learning Control (ILC)**, a powerful technique for improving performance over repeated executions of a task.

1.1 Strategies for Model Mismatch

When our nominal model $f_{\text{nom}}(\mathbf{x}, \mathbf{u})$ differs from the true system dynamics $f_{\text{true}}(\mathbf{x}, \mathbf{u})$, the optimal policy computed for the model will be suboptimal or even unstable on the real system. Feedback control (like LQR or MPC) provides some robustness, but it reacts to errors rather than anticipating them. If we need high performance (e.g., tracking a trajectory very aggressively), feedback gains alone may not be sufficient.

There are several standard approaches to close this gap:

1. **Parameter Estimation (System ID):** Assumes a "gray-box" model structure (e.g., a rigid body chain) and fits physical parameters (masses, lengths, friction coefficients) from data.
   - **Pros:** Interpretable, generalizes well to new tasks.
   - **Cons:** Limited by the assumed model structure. If the physics model doesn't include a specific phenomenon (e.g., stiction), fitting parameters won't fix it.
2. **Learn a Model (Black-box / Residual):** Fits a generic function approximator (like a Neural Network or Gaussian Process) to the dynamics $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$ or the error residual $\mathbf{e}_{k+1} = \mathbf{x}_{k+1}^{\text{true}} - f_{\text{nom}}(\mathbf{x}_k, \mathbf{u}_k)$.
   - **Pros:** Flexible, can capture complex nonlinear effects.
   - **Cons:** Data-hungry, often fails to generalize outside the training distribution, "black-box" nature makes stability analysis hard.
3. **Learn a Policy (Model-Free RL):** Optimizes the policy $\pi(\mathbf{x})$ directly based on rewards, ignoring the dynamics model entirely.
   - **Pros:** Requires very few assumptions.
   - **Cons:** Extremely sample inefficient (requires millions of trials), typically does not generalize to new tasks.
4. **Transfer / Iterative Learning Control (ILC):** Assumes we have a decent nominal model and a specific reference trajectory we want to track. We use data from the real system to iteratively refine the **feedforward control** for that specific task.
   - **Pros:** Very sample efficient (often converges in $<$10 iterations), high precision.
   - **Cons:** Task-specific (doesn't generalize to new trajectories).

1.2 Iterative Learning Control (ILC)

ILC is based on the idea that if we perform the same task repeatedly, the errors we see are likely consistent (systematic). Instead of just reacting to them with feedback, we should "learn" from the previous iteration's error to adjust our plan for the next iteration.

We can view ILC as a specialized form of policy optimization. Consider a tracking controller of the form:

$$\mathbf{u}_k(\mathbf{x}) = \bar{\mathbf{u}}_k - \mathbf{K}_k(\mathbf{x} - \bar{\mathbf{x}}_k)$$

Standard LQR/MPC keeps the feedforward term $\bar{\mathbf{u}}_k$ constant (from the nominal plan). ILC updates $\bar{\mathbf{u}}_k$ based on the error observed in the previous rollout.

1.2.1 Derivation via Sequential Quadratic Programming (SQP)

We can rigorously derive the ILC update by viewing the trajectory optimization problem as a constrained nonlinear program (NLP):

$$\begin{aligned} \min_{\mathbf{z}} \quad & J(\mathbf{z}) = \sum_{k=1}^{N-1} \frac{1}{2}\|\mathbf{x}_k - \bar{\mathbf{x}}_k\|^2_{\mathbf{Q}} + \frac{1}{2}\|\mathbf{u}_k - \bar{\mathbf{u}}_k\|^2_{\mathbf{R}} + \frac{1}{2}\|\mathbf{x}_N - \bar{\mathbf{x}}_N\|^2_{\mathbf{Q}_N} \\ \text{subject to} \quad & \mathbf{c}(\mathbf{z}) = 0 \quad (\text{Dynamics: } \mathbf{x}_{k+1} - f(\mathbf{x}_k, \mathbf{u}_k) = 0) \end{aligned}$$

where $\mathbf{z} = [\mathbf{x}_1, \mathbf{u}_1, \ldots, \mathbf{x}_N]^\top$ is the full decision vector.

To solve this using SQP (which is equivalent to Newton's method on the KKT conditions), we look for a step $\delta\mathbf{z}$ by solving the following linear system:

$$\begin{bmatrix} \mathbf{H} & \mathbf{C}^\top \\ \mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \delta\mathbf{z} \\ \delta\lambda \end{bmatrix} = \begin{bmatrix} -\nabla J(\mathbf{z}) \\ -\mathbf{c}(\mathbf{z}) \end{bmatrix} \tag{1.1}$$

where:

- $\mathbf{H} \approx \nabla^2_{zz}\mathcal{L}$ is the Hessian of the Lagrangian (often approximated by the Gauss-Newton Hessian of the cost J).
- $\mathbf{C} = \frac{\partial \mathbf{c}}{\partial \mathbf{z}}$ is the Jacobian of the constraints (dynamics).
- $\nabla J(\mathbf{z})$ is the gradient of the cost.
- $\mathbf{c}(\mathbf{z})$ is the constraint violation residual.

The ILC Strategy: In ILC, we exploit the difference between the **nominal model** and the **real system** to populate the terms in Equation 1.1:

1. **RHS (Gradient/Residual):** We perform a rollout on the *real* system.
   - Since the rollout happens on the physical system, it is dynamically feasible with respect to the true dynamics, so $\mathbf{c}_{\text{true}}(\mathbf{z}) = 0$.
   - We compute the gradient $\nabla J(\mathbf{z})$ using the trajectory data $(\mathbf{X}, \mathbf{U})$ collected from the real system. This captures the true tracking error.
2. **LHS (Curvature/Jacobian):** We cannot compute the true system Hessians $\mathbf{H}_{\text{true}}$ or Jacobians $\mathbf{C}_{\text{true}}$.
   - Instead, we approximate them using the *nominal model*:

     $$\mathbf{H} \approx \mathbf{H}_{\text{nom}}, \quad \mathbf{C} \approx \mathbf{C}_{\text{nom}}$$

   - Since our nominal model is a reasonable approximation, and assuming the current trajectory is close to the reference, these matrices (which can be computed offline) are sufficient to determine a good search direction.

Solving this KKT system for $\delta\mathbf{z}$ (specifically the control part $\delta\mathbf{u}$) yields the feedforward update $\mathbf{u}_{\text{new}} = \mathbf{u}_{\text{old}} + \delta\mathbf{u}$. This explains why ILC works: it uses the **true gradient** to drive the optimization, conditioned by the **model's curvature**. In the context of robotics with model mismatch:

- **The Gradient ∇J:** Depends on the *true* system behavior. If we run a rollout on the real robot, we can compute the cost and its sensitivity to the controls around the actual trajectory. This gives us the "true" gradient direction (or a very good approximation of it).
- **The Hessian $\nabla^2 J$:** Computing the true Hessian on a physical system is impossible. However, we have a *nominal model*. We can approximate the curvature of the problem using our model: $\mathbf{H}_{\text{nom}} \approx \nabla^2 J_{\text{true}}$.

Convergence theory tells us that as long as the nominal model is "close enough" (specifically, if $\|\mathbf{I} - \mathbf{H}_{\text{nom}}^{-1}\mathbf{H}_{\text{true}}\| < 1$), this iteration will converge to the local optimum of the *true* system.

1.3 ILC as a Quadratic Program (QP)

We can implement this update step by solving a Linear-Quadratic problem (similar to the subproblem in SQP or iLQR), but mixing real data with model derivatives.

Algorithm 1: Model-Based ILC Algorithm

Input: Nominal model f_{nom}, Initial guess $\mathbf{U}^{(0)}$, Reference $\mathbf{X}^{\text{ref}}$

for *iteration* $i = 0, 1, \ldots$ **do**

`// 1. Rollout on Real System`

Run $\mathbf{U}^{(i)}$ on the robot (possibly with feedback stabilization).

Record actual trajectory $\mathbf{X}^{(i)}$.

Compute tracking error $\mathbf{e}_k = \mathbf{x}_k^{(i)} - \mathbf{x}_k^{\text{ref}}$.

`// 2. Linearize Nominal Model`

Compute $\mathbf{A}_k, \mathbf{B}_k$ from f_{nom} around $\mathbf{x}_k^{(i)}, \mathbf{u}_k^{(i)}$.

`// 3. Formulate Optimization Step`

We want to find corrections $\delta\mathbf{x}, \delta\mathbf{u}$ that reduce the error. We minimize the approximate cost of the *next* iteration:

$$\min_{\delta\mathbf{X}, \delta\mathbf{U}} \sum \frac{1}{2}\|(\mathbf{x}_k^{(i)} + \delta\mathbf{x}_k) - \mathbf{x}_k^{\text{ref}}\|_{\mathbf{Q}}^2 + \frac{1}{2}\|\delta\mathbf{u}_k\|_{\mathbf{R}}^2$$

Expanding the quadratic term $\|\mathbf{x} + \delta\mathbf{x} - \mathbf{x}^{\text{ref}}\|_{\mathbf{Q}}^2$ gives a linear term $\delta\mathbf{x}^\top\mathbf{Q}(\mathbf{x}^{(i)} - \mathbf{x}^{\text{ref}})$. The QP becomes:

$$\begin{aligned} \min_{\delta\mathbf{X}, \delta\mathbf{U}} \quad & \sum_{k=1}^{N-1} \left(\frac{1}{2}\delta\mathbf{x}_k^\top\mathbf{Q}\delta\mathbf{x}_k + \mathbf{q}_k^\top\delta\mathbf{x}_k + \frac{1}{2}\delta\mathbf{u}_k^\top\mathbf{R}\delta\mathbf{u}_k \right) \\ \text{subject to} \quad & \delta\mathbf{x}_{k+1} = \mathbf{A}_k\delta\mathbf{x}_k + \mathbf{B}_k\delta\mathbf{u}_k \\ & \mathbf{u}_{\min} \le \mathbf{u}_k^{(i)} + \delta\mathbf{u}_k \le \mathbf{u}_{\max} \end{aligned}$$

where $\mathbf{q}_k = \mathbf{Q}(\mathbf{x}_k^{(i)} - \mathbf{x}_k^{\text{ref}})$ is the gradient driving the update based on real errors.

`// 4. Update`

Solve QP for $\delta\mathbf{U}$.

$\mathbf{U}^{(i+1)} \leftarrow \mathbf{U}^{(i)} + \delta\mathbf{U}$.

Intuition. The QP asks: "According to my *nominal* model $(\mathbf{A}, \mathbf{B})$, how should I change my inputs $\delta\mathbf{u}$ to eliminate the tracking error $\mathbf{x}^{(i)} - \mathbf{x}^{\text{ref}}$ that I observed on the real system?"

1.4 Code Analysis: `cartpole-ilc.ipynb`

The notebook simulates this process on a Cartpole system with parameter mismatch.

Code 1.1 (Julia Notebook: ILC for Cartpole Swing-up).
Problem Setup:

- **Nominal Model:** Standard Cartpole parameters.
- **True Model:** Perturbed masses ($m_c + 0.02$, $m_p - 0.01$), length ($l + 0.005$), and crucially, **nonlinear friction** (tanh damping) which is completely absent in the nominal model.

Step 1: Nominal Plan (ALTRO)

- Solves the swing-up problem using `Altro.jl` on the nominal model.
- Produces a reference trajectory $\mathbf{X}^{\text{ref}}, \mathbf{U}^{\text{nom}}$.

Step 2: Baseline Performance

- Runs the nominal feedforward $\mathbf{U}^{\text{nom}}$ (plus LQR feedback) on the **True Model**.
- Result: The cartpole fails to reach the top or drift significantly due to the unmodeled friction and mass errors.

Step 3: ILC Update

- **Linearization:** Computes $\mathbf{A}_k, \mathbf{B}_k$ along the *nominal* trajectory using the *nominal* model.
- **QP Formulation:** Uses `OSQP` to solve for $\delta\mathbf{u}$.
- **Cost Vector q:** The linear cost term in the QP is set to `Qilc * (xtraj - Xopt)`. This vector encodes the error between the rollout on the **true** physics (`xtraj`) and the **desired** plan (`Xopt`).
- **Constraints:** Enforces nominal linearized dynamics and torque limits.
- **Result:** The computed $\delta\mathbf{u}$ effectively learns an inverse dynamics term to cancel out the unmodeled friction and gravity errors. After the update, the trajectory tracking on the true system improves dramatically.