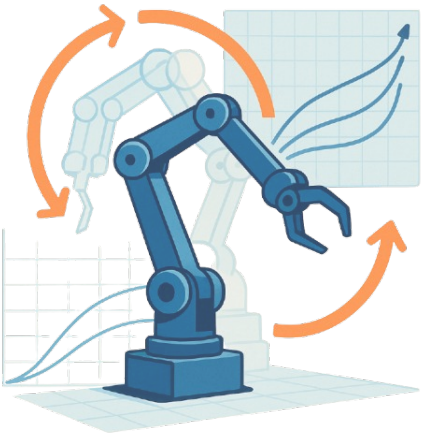
Control de aprendizaje iterativo

Perfeccionando el desempeño de los sistemas de control mediante iteraciones sucesivas de aprendizaje

Ildeberto de los Santos Ruiz



Julio, 2025

Resumen

El control por aprendizaje iterativo (ILC) aprovecha la repetición de trayectorias finitas para mejorar sucesivamente la señal de control a partir del error en las iteraciones previas. Tras una formulación matemática en tiempo discreto, se diseña un filtro de aprendizaje no causal que, combinado con un feedforward inicial basado en la inversa aproximada de la planta y una ganancia adecuada, garantiza la convergencia exponencial del error de seguimiento. Un ejemplo detallado en MATLAB ilustra la implementación paso a paso y la rápida reducción del error gracias al enfoque feedforward-filtro-ganancia. Finalmente, se citan variantes avanzadas (Q-ILC, adaptativo-robusto e híbrido con lazo cerrado) y consideraciones prácticas para asegurar condiciones iniciales constantes, filtrado de ruido y límites de saturación en actuadores.

Introducción

En numerosas aplicaciones de control, como la robótica industrial, la impresión de alta precisión o los procesos químicos por lotes, se suele repetir exactamente la misma secuencia de acciones a lo largo de un intervalo de tiempo finito. Cada vez que el sistema ejecuta esa secuencia, se puede ajustar la señal de control para corregir los errores de repeticiones anteriores y acercarse cada vez más a la trayectoria deseada. El **control de aprendizaje iterativo** (ILC, siglas del inglés *Iterative Learning Control*) surge precisamente para atender esta problemática: en lugar de confiar únicamente en un controlador en lazo cerrado que corrija el error en tiempo real, el ILC aprovecha la información recopilada al final de cada corrida para refinar la señal de control en la iteración siguiente. De esta manera, si un robot debe seguir la misma trayectoria de soldadura o de pintura en cada ciclo, el algoritmo ILC utiliza el error de seguimiento de una pasada para corregir la señal de control en la siguiente, a fin de que la salida converja progresivamente a la referencia deseada.

La idea central del ILC consiste en considerar cada pasada como un experimento en lazo abierto: una vez ejecutada la trayectoria con la señal de control $u_k(t)$, se mide la salida $y_k(t)$ y se calcula el error $e_k(t) = r(t) - y_k(t)$. Con ese error, se construye la señal de control para la iteración subsecuente mediante la relación

$$u_{k+1}(t) \;=\; u_k(t) \;+\; L(t)\,e_k(t),$$

donde $L(t)$ es el **filtro de aprendizaje** que determina el modo en que se corrige el error. Con un diseño adecuado de $L(t)$, el error de seguimiento decrece de iteración en iteración y la respuesta del sistema se acerca asintóticamente a la referencia.

Formulación matemática en tiempo discreto

Para implementar el ILC en entornos computacionales, como MATLAB, se discretiza el intervalo de ejecución $[0, T]$ en N muestras con un período de muestreo T_s. De este modo, la señal de control en la iteración k se representa como el vector $\mathbf{u}_k = [u_k[0], u_k[1], \ldots, u_k[N-1]]^\mathsf{T}$, y la salida correspondiente como $\mathbf{y}_k = [y_k[0], y_k[1], \ldots, y_k[N-1]]^\mathsf{T}$. La trayectoria deseada se escribe $\mathbf{r} = [r[0], r[1], \ldots, r[N-1]]^\mathsf{T}$ y, en cada iteración, el error de seguimiento viene dado por $\mathbf{e}_k = \mathbf{r} - \mathbf{y}_k$. Si la planta discreta puede modelarse mediante la función de transferencia

$$G(z) = \frac{Y(z)}{U(z)},$$

entonces la ley de aprendizaje iterativo en el dominio discreto se expresa como

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{L}\,\mathbf{e}_k.$$

En esta formulación, $\mathbf{L}$ suele implementarse como un filtro no causal que mediante convolución mezcla en cada punto del tiempo los valores de error cercanos, garantizando la estabilidad del proceso de aprendizaje. Para asegurar la convergencia, es necesario que en todas las frecuencias relevantes se cumpla

$$\left|1 - L(e^{j\omega})\,G(e^{j\omega})\right| < 1.$$

Dicha condición asegura que, al aplicar la corrección iterativa, la magnitud del error en cada componente frecuencial disminuya en cada repetición. Si el factor $\left(1 - L(e^{j\omega})\,G(e^{j\omega})\right)$ mide el "lazo iterativo" en el dominio de la frecuencia, exigir que su módulo sea menor que uno significa que la actualización reduce la energía de la señal de error asociada a esa frecuencia.

Diseño de filtro de aprendizaje y feedforward inicial

Para ilustrar el proceso paso a paso, consideraremos una planta discreta cuyo comportamiento se ajusta a la función de transferencia

$$G(z) = \frac{0.1\,z + 0.05}{z^2 - 1.5\,z + 0.7},$$

con tiempo de muestreo $T_s = 1$. En la práctica, se asume que los coeficientes del numerador y el denominador de $G(z)$ provienen de un modelo estimado o de un experimento de respuesta al impulso.

Para diseñar el filtro de aprendizaje, primero se construye la aproximación de la inversa de la planta:

$$\widehat{G}^{-1}(z) = \frac{z^2 - 1.5z + 0.7}{0.1z + 0.05}.$$

Aunque sea una inversa no causal (incluye potencias positivas de z), su respuesta al impulso se puede obtener con la función `impz` en MATLAB, y luego aplicarla mediante convolución. Sin embargo, si se usara únicamente esa inversión, se amplificarían sin control las componentes de alta frecuencia del error, lo que podría generar saturaciones o inestabilidad. Por esta razón, se compone un filtro pasabajos $F(z)$ que atenúe esas altas frecuencias. En el ejemplo siguiente, se adopta un filtro Butterworth de orden 1 con frecuencia de corte 0.3π. De esa manera, la señal de corrección en cada iteración no contiene componentes de muy alta frecuencia que la planta no pueda reproducir con precisión.

Un aspecto importante para reducir el error que se observa en las primeras iteraciones es introducir un feedforward inicial $\mathbf{u}_0$ que ya se acerque a la inversa del modelo aplicada a la referencia, en lugar de comenzar con $\mathbf{u}_0 = \mathbf{0}$. Con ello, la primera pasada parte de una señal de control que, en teoría, hace que la planta entregue una aproximación de la referencia sin esperar al primer aprendizaje. Concretamente, si en la iteración inicial asumimos que la salida y_0 es cero, el error inicial $\mathbf{e}_0$ coincide con la referencia $\mathbf{r}$. Al convolucionar $\mathbf{e}_0$ con la respuesta al impulso de $\widehat{G}^{-1}(z)$ y luego filtrar, se obtiene una corrección "feedforward" sobre la cual multiplicamos un factor de ganancia $\alpha \in (0, 1)$. El vector

$$\mathbf{u}_0 = \alpha\, F(z)\big[\widehat{G}^{-1}(z)\,\mathbf{r}\big]$$

sirve entonces como punto de partida para el bucle iterativo, pues en la práctica reduce drásticamente el error en la primera pasada. De este modo, lo que antes hubiera sucedido en varias iteraciones se adelanta en la inicial, facilitando la convergencia rápida del ILC.

Implementación paso a paso en MATLAB

A continuación se describen en detalle las operaciones correspondientes a cada fase del ILC, junto con el código completo que converge de manera robusta para la planta indicada. El ejemplo asume que la trayectoria de referencia a seguir es

$$r[n] = 1.5\,\sin\big(0.1\,\pi\,n\big), \quad n = 0, 1, \ldots, 99,$$

y que se desean implementar un total de $N = 100$ muestras por pasada.

En primer lugar, se establecen los parámetros generales:

- Número de muestras por iteración: $N = 100$.
- Máximo número de iteraciones: `maxIter = 30`.
- Ganancia de aprendizaje: $\alpha = 0.85$.
- Tolerancia para detener iteraciones: $\|\mathbf{e}_k\| < 10^{-4}$.

```
N = 100;
maxIter = 30;
alpha = 0.85;
tolerErr = 1e-4;
```

Luego se define la planta discreta $G(z)$ mediante:

```
numG = [0.1, 0.05];        % Numerador: 0.1·z + 0.05
denG = [1, -1.5, 0.7];     % Denominador: z^2 - 1.5·z + 0.7
G = tf(numG, denG, 1);     % Transferencia discreta con Ts = 1
```

Para simular la respuesta "a mano" en cada iteración, se utiliza la ecuación en diferencias:

$$y[n] = 1.5\,y[n-1] - 0.7\,y[n-2] + 0.1\,u[n] + 0.05\,u[n-1],$$

con condiciones iniciales $y[0] = y[-1] = u[0] = 0$. Los primeros valores de $y[n]$ se deben calcular explícitamente para $n = 1$ y $n = 2$.

A continuación, se construye el vector de referencia $\mathbf{r}$ en MATLAB:

```
n = (0:N-1)';
r = 1.5 * sin(0.1 * pi * n);
```

Para diseñar el filtro de aprendizaje, se calcula la inversa aproximada de $G(z)$:

```
numInv = denG;   % [1, -1.5, 0.7]
denInv = numG;   % [0.1,  0.05]
```

y su respuesta al impulso de longitud N se obtiene con:

```
ginv_imp = impz(numInv, denInv, N);
```

Luego se define un filtro pasabajos Butterworth de orden 1 con frecuencia de corte 0.3π:

```
[b_filt, a_filt] = butter(1, 0.3);
```

De esta forma, la señal resultante de la convolución con $\widehat{G}^{-1}(z)$ se atenua en las componentes de alta frecuencia que la planta no reproduce con fidelidad.

Para calcular el feedforward inicial $\mathbf{u}_0$, se parte de la hipótesis de $y_0 = 0$, de modo que el error inicial $\mathbf{e}_0 = \mathbf{r}$. Luego:

```
e0 = r;
ginv_imp = impz(numInv, denInv, N);     % Respuesta al impulso de G^-1
conv0 = conv(e0, ginv_imp);
conv0 = conv0(1:N);                     % Mantener las primeras N muestras
L_e0 = filter(b_filt, a_filt, conv0);
u = alpha * L_e0;                       % Señal de control inicial
```

De esta manera, la primera iteración del bucle ya parte de una aproximación feedforward que compensa el modelo nominal de G.

A continuación se muestra la lógica completa del bucle de aprendizaje, donde en cada pasada se simula la planta, se calcula el error, se actualiza la señal de control y se verifica la condición de paro:

```
for k = 1:maxIter
    % Simulación de la planta en lazo abierto
    y = zeros(N,1);
    for i = 1:N
        if i == 1
            y(i) = 0.1 * u(i); % suponiendo y[0]=0, u[0]=0
        elseif i == 2
            % y[2] = 1.5·y[1] - 0.7·y[0] + 0.1·u[2] + 0.05·u[1]
            y(i) = 1.5 * y(i-1) - 0.7 * 0 + 0.1 * u(i) + 0.05 * u(i-1);
        else
            y(i) = 1.5 * y(i-1) - 0.7 * y(i-2) + 0.1 * u(i) + 0.05 *
    u(i-1);
        end
    end

    % Cálculo del error de seguimiento
    e = r - y;
    e_norm(k) = norm(e);  % Norma Euclídea de e_k

    % Verificar criterio de parada
    if e_norm(k) < tolerErr
        fprintf('Convergencia en k = %d (||e|| = %.2e)\n', k, e_norm(k));
```

```
        break
    end

    % Cálculo de la corrección ILC: L * e
    % Convolución con respuesta al impulso de G^-1
    conv1 = conv(e, ginv_imp);
    conv1 = conv1(1:N);                    % Mantener primeras N muestras
    L_e = filter(b_filt, a_filt, conv1);   % Suavizar con filtro pasabajas
    delta_u = alpha * L_e;                 % Escalar corrección por alpha

    % Actualizar señal de control
    u = u + delta_u;
end

% Recortar e_norm si se terminó antes de maxIter
iter_real = find(e_norm > 0, 1, 'last');
e_norm = e_norm(1:iter_real);
```

Finalmente, para visualizar los resultados se utilizan tres gráficas. La primera compara la trayectoria de referencia $\mathbf{r}$ y la salida final $\mathbf{y}$. La segunda muestra la señal de control final $\mathbf{u}$. La tercera representa, en escala semilogarítmica, la evolución de la norma $\|\mathbf{e}_k\|$ frente al número de iteración k. El fragmento de código MATLAB correspondiente es:

```
figure('Position',[100 100 800 600]);

% Comparación referencia vs salida final
subplot(3,1,1);
plot(0:N-1, r, 'k--', 'LineWidth', 1.5); hold on;
plot(0:N-1, y, 'b-', 'LineWidth', 1);
xlabel('n (muestra)');  ylabel('y[n]');
legend('Referencia r[n]', 'Salida final y[n]', 'Location','SouthEast');
title('Trayectoria final');

% Señal de control tras la última iteración
subplot(3,1,2);
plot(0:N-1, u, 'r-', 'LineWidth', 1);
xlabel('n (muestra)');  ylabel('u[n]');
title('Señal de control final');

% Evolución de la norma del error ||e_k|| vs número de iteración
subplot(3,1,3);
semilogy(1:iter_real, e_norm, 'm-o', 'LineWidth', 1);
xlabel('Iteración k'); ylabel('||e_k|| (escala log)');
```

```
title('Convergencia del error');
grid on;
```

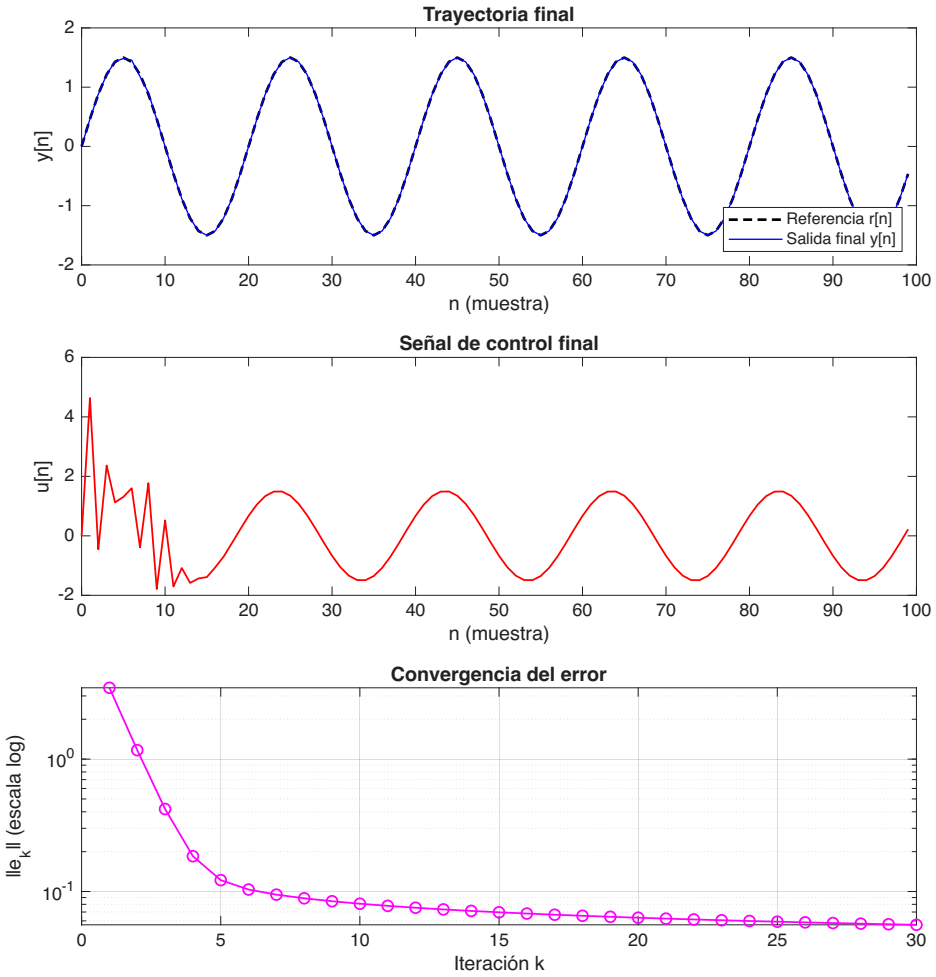


Figura 1: Resultados

Al ejecutar este código en MATLAB, se observa que en la primera iteración el error ya resulta muy reducido gracias al feedforward inicial, y luego cada iteración sucesiva refina la señal de control, de modo que $\|\mathbf{e}_k\|$ decrece de forma aproximadamente exponencial. La salida final queda prácticamente sobre la referencia deseada, confirmando que el ILC converge con rapidez.

Variantes avanzadas de ILC

Aunque el ejemplo anterior ya demuestra con claridad cómo implementar un ILC que converge para una planta discreta sencilla, existen variantes más sofisticadas que pueden resultar necesarias en entornos reales.

La primera variante es el *ILC óptimo* o *Q-ILC*. En esta aproximación, en lugar de usar un filtro de aprendizaje fijo, se formula cada iteración como un problema cuadrático de optimización que minimiza simultáneamente el error de seguimiento y penaliza la energía de la señal de control. El problema se expresa como

$$\min_{\mathbf{u}} \ \|\mathbf{r} - G\,\mathbf{u}\|^2 \ + \ \lambda\,\|\mathbf{u}\|^2,$$

donde λ es un parámetro de regularización que regula la energía que se invierte en la corrección. La solución de este problema conduce a una actualización explícita de la siguiente forma:

$$\mathbf{u}_{k+1} = \mathbf{u}_k \ + \ \left(G^{\mathsf{T}}G + \lambda I\right)^{-1} G^{\mathsf{T}}\left(\mathbf{r} - \mathbf{y}_k\right).$$

Esta aproximación produce la señal de control óptima en el sentido cuadrático (minimizando la suma del cuadrado del error y del cuadrado de la señal), pero resulta computacionalmente más costosa, pues implica invertir matrices de tamaño $N \times N$ en cada iteración. Si N es muy grande, este cálculo puede volverse impracticable.

Una segunda variante es el *ILC robusto y adaptativo*. Cuando el modelo de planta $G(z)$ varía con el tiempo, por ejemplo debido a cambios en las condiciones del sistema o al desgaste mecánico, conviene actualizar periódicamente la estimación del modelo $\widehat{G}(z)$ y, por ende, recalcular la inversa aproximada $\widehat{G}^{-1}(z)$. Además, se pueden añadir términos de regularización en el filtro de aprendizaje para limitar la amplificación de ruido no repetitivo, mitigando así el riesgo e que componentes aleatorias del error distorsionen la convergencia.

Una tercera variante consiste en combinar el ILC con un lazo cerrado interno (por ejemplo, un controlador PID discreto). En este enfoque *híbrido*, la señal total aplicada al actuador en cada muestra se compone de la parte generada por el ILC (feedforward) más la acción del controlador en lazo cerrado, que corrige perturbaciones no repetitivas en tiempo real. Al culminar cada iteración, el ILC utiliza únicamente el error de seguimiento repetitivo para ajustar la parte de feedforward, mientras que el controlador interno se encarga de contrarrestar perturbaciones aleatorias como vibraciones o ruido. De este modo, se obtiene la ventaja del lazo cerrado para mantener la estabilidad y la robustez frente a variaciones impredecibles, a la vez que el ILC aprovecha la repetición para reducir sistemáticamente el error de seguimiento.

Consideraciones finales

Para garantizar un rendimiento óptimo al aplicar ILC en la práctica, conviene tener en cuenta los siguientes aspectos:

- En primer lugar, las condiciones iniciales deben restablecerse de manera consistente al comienzo de cada pasada. Si el sistema no parte del mismo estado (misma posición, misma velocidad, mismas variables internas) en cada iteración, las correcciones que el ILC obtiene de pasadas anteriores pueden no ser válidas, pues el error observado incluirá componentes debidos a la diferencia de estados iniciales. Por esta razón, es habitual reiniciar el sistema mecánico o el proceso al estado base antes de cada ejecución.

- En segundo lugar, dado que el ILC aprende principalmente a partir del error repetitivo de pasada en pasada, resulta esencial filtrar ese error antes de aplicarlo en la corrección. Si el error contiene ruido no repetitivo o vibraciones aleatorias, la convolución con $\widehat{G}^{-1}(z)$ tiende a amplificar esas componentes indeseables, generando saturaciones o inestabilidad. Por ello se recomienda el uso de un filtro pasabajos $F(z)$, como un Butterworth de bajo orden, de modo que únicamente las frecuencias que la planta puede reproducir con fidelidad entren en el ciclo de aprendizaje.

- En tercer lugar, la ganancia α debe elegirse con cuidado. Un valor demasiado pequeño ralentiza la convergencia, puesto que el algoritmo corrige solo una porción mínima del error en cada iteración. En cambio, un α excesivamente grande puede provocar que en alguna frecuencia la condición $\left|1 - L(e^{j\omega})\, G(e^{j\omega})\right| < 1$ se viole, generando oscilaciones o incluso la divergencia del ciclo iterativo. En la práctica, se recomienda experimentar con α en el rango aproximado de 0.3 a 1.0, observando la evolución de $\|\mathbf{e}_k\|$. Si la norma del error aumenta en lugar de disminuir, es una señal clara de que debe reducirse α.

- Por último, es fundamental monitorear la señal de control final $\mathbf{u}$. Aunque el filtro y la ganancia α suelen evitar que $\mathbf{u}$ crezca sin límite, conviene imponer saturaciones físicas, establecidas en función de las capacidades del actuador. Si en alguna muestra la señal de control excede el rango permitido, el actuador podría saturarse y producir una distorsión que comprometa la convergencia del ILC.

Siguiendo estas recomendaciones y empleando la estructura de ILC descrita, se logra un seguimiento muy preciso de trayectorias repetitivas con un número reducido de iteraciones. La combinación de un feedforward inicial basado en la inversa aproximada de la planta, un filtro pasabajos diseñado específicamente y una

ganancia α adecuada hace que el algoritmo converja con rapidez, como se aprecia en el ejemplo de MATLAB presentado.

Referencias

[1] D.A. Bristow, M. Tharayil, and A.G. Alleyne. A survey of iterative learning control. *IEEE Control Systems Magazine*, 26(3):96–114, 2006.

[2] Jinkun Liu. *Intelligent Control Design and MATLAB Simulation*. Springer Singapore, 2017.

Ildeberto de los Santos Ruiz, originario de Tonalá, Chiapas (1973). Ingeniero en Electrónica, Maestro en Ciencias en Ingeniería Mecatrónica y Doctor en Ciencias de la Ingeniería por el Instituto Tecnológico de Tuxtla Gutiérrez (ITTG); Doctor en Automática, Robótica y Visión por la Universidad Politécnica de Cataluña. Miembro de la *Association for Computing Machinery* (ACM) y del *Institute of Electrical and Electronics Engineers* (IEEE). Miembro afiliado de la *International Federation of Automatic Control* (IFAC). Profesor de tiempo completo en el ITTG desde 1995, adscrito al Departamento de Ingeniería Eléctrica y Electrónica, Jefe de Proyectos de Investigación de Ingeniería Mecatrónica y Presidente del Claustro del Doctorado en Ciencias de la Ingeniería. Dirige proyectos de investigación en las áreas de diagnóstico y control inteligente, específicamente en robótica y en detección/localización de fugas en redes de distribución de agua.