Learning to Track Trajectories

Okan Koç

MPI for Intelligent Systems, Tübingen
Robot Learning Lab

okan.koc@tuebingen.mpg.de

August 27, 2014

Table of Contents

Iterative Learning Control (ILC)

- Task: Follow a trajectory under unknown repeating disturbances and model mismatch. [BTA06]

Iterative Learning Control (ILC)

- Task: Follow a trajectory under unknown repeating disturbances and model mismatch. [BTA06]
- In ILC, control inputs are adjusted at each iteration.

Iterative Learning Control (ILC)

- Task: Follow a trajectory under unknown repeating disturbances and model mismatch. [BTA06]
- In ILC, control inputs are adjusted at each iteration.
  - *Feedforward* (open-loop) adjustment.

Iterative Learning Control (ILC)

- Task: Follow a trajectory under unknown repeating disturbances and model mismatch. [BTA06]
- In ILC, control inputs are adjusted at each iteration.
  - *Feedforward* (open-loop) adjustment.
  - The goal is drive the deviations from the trajectory to zero.

Iterative Learning Control (ILC)

- Task: Follow a trajectory under unknown repeating disturbances and model mismatch. [BTA06]
- In ILC, control inputs are adjusted at each iteration.
  - *Feedforward* (open-loop) adjustment.
  - The goal is drive the deviations from the trajectory to zero.

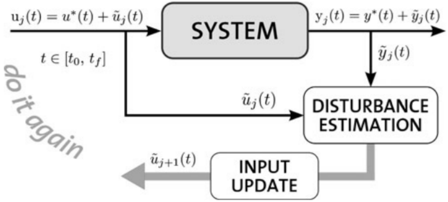

Figure : ILC Framework [SMD12]

Problem Setting

- Problem: Continuous time trajectory tracking under the *nonlinear* system dynamics:

[1] See [ZWC10] for a spline-based method for differential-flat dynamics

Problem Setting

- Problem: Continuous time trajectory tracking under the *nonlinear* system dynamics:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$$
$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(\mathbf{t}))$$

- Goal: Track a reference trajectory $\mathbf{y}^*(t),\ 0 \leq t \leq T$ by applying the control inputs $\mathbf{u}^*(t)$.

[1] See [ZWC10] for a spline-based method for differential-flat dynamics

Problem Setting

- Problem: Continuous time trajectory tracking under the *nonlinear* system dynamics:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$$
$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(\mathbf{t}))$$

- Goal: Track a reference trajectory $\mathbf{y}^*(t),\ 0 \leq t \leq T$ by applying the control inputs $\mathbf{u}^*(t)$.
- Use: Any trajectory generation algorithm [1] that comes up with $\mathbf{u}^*(t)$.

[1] See [ZWC10] for a spline-based method for differential-flat dynamics

Problem Setting

- Problem: Continuous time trajectory tracking under the *nonlinear* system dynamics:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$$
$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(\mathbf{t}))$$

- Goal: Track a reference trajectory $\mathbf{y}^*(t),\ 0 \leq t \leq T$ by applying the control inputs $\mathbf{u}^*(t)$.
- Use: Any trajectory generation algorithm [1] that comes up with $\mathbf{u}^*(t)$.

[1] See [ZWC10] for a spline-based method for differential-flat dynamics

Problem Setting

- However $\mathbf{f}$ is not known very well!

Problem Setting

- However $\mathbf{f}$ is not known very well!
- We will fail to track $\mathbf{y}^*(t)$ just by applying $\mathbf{u}^*(t)$.

Problem Setting

- However $\mathbf{f}$ is not known very well!
- We will fail to track $\mathbf{y}^*(t)$ just by applying $\mathbf{u}^*(t)$.
- We can use ILC to correct for deviations!

Problem Setting

- However $\mathbf{f}$ is not known very well!
- We will fail to track $\mathbf{y}^*(t)$ just by applying $\mathbf{u}^*(t)$.
- We can use ILC to correct for deviations!
- We will follow the method of Schöllig et al. [SMD12] in the following slides.

Linearize Around Trajectory

- Linearize model around trajectory:

Linearize Around Trajectory

- Linearize model around trajectory:

$$\dot{\tilde{\mathbf{x}}}(t) = A(t)\tilde{\mathbf{x}}(t) + B(t)\tilde{\mathbf{u}}(t)$$
$$\tilde{\mathbf{y}}(t) = C(t)\tilde{\mathbf{x}}(t)$$

Linearize Around Trajectory

- Linearize model around trajectory:

$$\dot{\tilde{\mathbf{x}}}(t) = A(t)\tilde{\mathbf{x}}(t) + B(t)\tilde{\mathbf{u}}(t)$$
$$\tilde{\mathbf{y}}(t) = C(t)\tilde{\mathbf{x}}(t)$$

- The time variant matrices are:

Linearize Around Trajectory

- Linearize model around trajectory:

$$\dot{\tilde{\mathbf{x}}}(t) = A(t)\tilde{\mathbf{x}}(t) + B(t)\tilde{\mathbf{u}}(t)$$
$$\tilde{\mathbf{y}}(t) = C(t)\tilde{\mathbf{x}}(t)$$

- The time variant matrices are:

$$A(t) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{(\mathbf{x}^*(t),\mathbf{u}^*(t))}$$
$$B(t) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right|_{(\mathbf{x}^*(t),\mathbf{u}^*(t))}$$
$$C(t) = \left.\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right|_{\mathbf{x}^*(t)}$$

Discretize linearized model

- For k $\in \{0, 1, \ldots, N-1\}$,

Discretize linearized model

- For k $\in \{0, 1, \ldots, N-1\}$,

$$\begin{aligned}
\tilde{\mathbf{x}}(k+1) &= A_D(k)\tilde{\mathbf{x}}(k) + B_D(k)\tilde{\mathbf{y}}(k) \\
\tilde{\mathbf{y}}(k+1) &= C_D(k+1)\tilde{\mathbf{x}}(k+1)
\end{aligned}$$

Discretize linearized model

- For k $\in \{0, 1, \ldots, N-1\}$,

$$\begin{aligned} \tilde{\mathbf{x}}(k+1) &= A_D(k)\tilde{\mathbf{x}}(k) + B_D(k)\tilde{\mathbf{y}}(k) \\ \tilde{\mathbf{y}}(k+1) &= C_D(k+1)\tilde{\mathbf{x}}(k+1) \end{aligned}$$

- The discretized matrices can be found using:

Discretize linearized model

- For k $\in \{0, 1, \ldots, N-1\}$,

$$\begin{aligned}\tilde{\mathbf{x}}(k+1) &= A_D(k)\tilde{\mathbf{x}}(k) + B_D(k)\tilde{\mathbf{y}}(k)\\ \tilde{\mathbf{y}}(k+1) &= C_D(k+1)\tilde{\mathbf{x}}(k+1)\end{aligned}$$

- The discretized matrices can be found using:

$$\exp^{h\left[\begin{array}{c|c} A(k) & B(k) \\ \hline 0 & 0 \end{array}\right]} = \left[\begin{array}{c|c} A_D(k) & B_D(k) \\ \hline 0 & I \end{array}\right]$$

$$C_D(k) = C(k)$$

Lifted Vector Representation

- Stack inputs together, $\mathrm{u} = (\tilde{\mathbf{u}}(1), \ldots, \tilde{\mathbf{u}}(N-1))$ to obtain the lifted vector form:

Lifted Vector Representation

- Stack inputs together, $\mathrm{u} = (\tilde{\mathbf{u}}(1), \ldots, \tilde{\mathbf{u}}(N-1))$ to obtain the lifted vector form:

$$
\begin{aligned}
\mathrm{x} &= F\mathrm{u} + \mathrm{d}^{0} \\
\mathrm{y} &= G\mathrm{x}
\end{aligned}
$$

Lifted Vector Representation

- Stack inputs together, $\mathrm{u} = (\tilde{\mathbf{u}}(1), \ldots, \tilde{\mathbf{u}}(N-1))$ to obtain the lifted vector form:

$$\mathrm{x} = F\mathrm{u} + \mathrm{d}^{0}$$
$$\mathrm{y} = G\mathrm{x}$$

- where the submatrices are:

Lifted Vector Representation

- Stack inputs together, $\mathrm{u} = (\tilde{\mathbf{u}}(1), \ldots, \tilde{\mathbf{u}}(N-1))$ to obtain the lifted vector form:

$$\mathrm{x} = F\mathrm{u} + \mathrm{d}^0$$
$$\mathrm{y} = G\mathrm{x}$$

- where the submatrices are:

$$F_{(i,j)} = \begin{cases} A_D(i-1)\ldots A_D(j)B_D(j-1), & j < i \\ B_D(j-1), & j = i \\ 0, & j > i \end{cases}$$
$$G_{(i,j)} = C_D$$

Disturbance Model

- The model for the evolution of disturbance over iterations:

Disturbance Model

- The model for the evolution of disturbance over iterations:

$$
\begin{aligned}
\mathrm{x}_l &= F\mathrm{u}_l + \mathrm{d}_l + \epsilon \\
\mathrm{y}_l &= G\mathrm{x}_l + \nu \\
\mathrm{d}_l &= \mathrm{d}_{l-1} + \omega_{l-1} \\
\epsilon &\sim \mathcal{N}(0, \Sigma_\epsilon) \\
\nu &\sim \mathcal{N}(0, \Sigma_\nu) \\
\omega &\sim \mathcal{N}(0, \Sigma_\omega)
\end{aligned}
$$

Disturbance Estimation

- The disturbance model

Disturbance Estimation

- The disturbance model

$$
\begin{aligned}
\mathrm{d}_l &= \mathrm{d}_{l-1} + \omega_{l-1} \\
\mathrm{y}_l &= G\mathrm{d}_l + GF\mathrm{u}_l + \eta \\
\omega &\sim \mathcal{N}(0, \Sigma_\omega) \\
\eta &\sim \mathcal{N}(0, \Sigma_\eta = G\Sigma_\epsilon G^{\mathrm{T}} + \Sigma_\nu)
\end{aligned}
$$

Disturbance Estimation

- The disturbance model

$$
\begin{aligned}
\mathrm{d}_l &= \mathrm{d}_{l-1} + \omega_{l-1} \\
\mathrm{y}_l &= G\mathrm{d}_l + GF\mathrm{u}_l + \eta \\
\omega &\sim \mathcal{N}(0, \Sigma_\omega) \\
\eta &\sim \mathcal{N}(0, \Sigma_\eta = G\Sigma_\epsilon G^{\mathrm{T}} + \Sigma_\nu)
\end{aligned}
$$

- is estimated with a Kalman filter.

Disturbance Estimation

- The disturbance model

$$
\begin{aligned}
\mathrm{d}_l &= \mathrm{d}_{l-1} + \omega_{l-1} \\
\mathrm{y}_l &= G\mathrm{d}_l + GF\mathrm{u}_l + \eta \\
\omega &\sim \mathcal{N}(0, \Sigma_\omega) \\
\eta &\sim \mathcal{N}(0, \Sigma_\eta = G\Sigma_\epsilon G^{\mathrm{T}} + \Sigma_\nu)
\end{aligned}
$$

- is estimated with a Kalman filter.

Kalman Filter

- Kalman filter equations:

Kalman Filter

- Kalman filter equations:

$$
\begin{aligned}
\bar{\mu}_t &= A_t \mu_{t-1} + B_t u_t \\
\bar{\Sigma}_t &= A_t \Sigma_{t-1} A_t^{\mathrm{T}} + \Sigma_\omega \\
K_t &= \bar{\Sigma}_t C_t^{\mathrm{T}} (C_t \bar{\Sigma}_t C_t^{\mathrm{T}} + \Sigma_\eta)^{-1} \\
\mu_t &= \bar{\mu}_t + K_t (y_t - C_t \bar{\mu}_t) \\
\Sigma_t &= (I - K_t C_t) \bar{\Sigma}_t
\end{aligned}
$$

Kalman Filter

- Kalman filter equations:

$$
\begin{aligned}
\bar{\mu}_t &= A_t \mu_{t-1} + B_t u_t \\
\bar{\Sigma}_t &= A_t \Sigma_{t-1} A_t^{\mathrm{T}} + \Sigma_\omega \\
K_t &= \bar{\Sigma}_t C_t^{\mathrm{T}} (C_t \bar{\Sigma}_t C_t^{\mathrm{T}} + \Sigma_\eta)^{-1} \\
\mu_t &= \bar{\mu}_t + K_t (y_t - C_t \bar{\mu}_t) \\
\Sigma_t &= (I - K_t C_t) \bar{\Sigma}_t
\end{aligned}
$$

- for $A_t = I$, $B_t = 0$ become:

Kalman Filter

- Kalman filter equations:

$$\begin{aligned}
\bar{\mu}_t &= A_t \mu_{t-1} + B_t u_t \\
\bar{\Sigma}_t &= A_t \Sigma_{t-1} A_t^{\mathrm{T}} + \Sigma_\omega \\
K_t &= \bar{\Sigma}_t C_t^{\mathrm{T}} (C_t \bar{\Sigma}_t C_t^{\mathrm{T}} + \Sigma_\eta)^{-1} \\
\mu_t &= \bar{\mu}_t + K_t (y_t - C_t \bar{\mu}_t) \\
\Sigma_t &= (I - K_t C_t) \bar{\Sigma}_t
\end{aligned}$$

- for $A_t = I$, $B_t = 0$ become:

$$\begin{aligned}
\hat{\mathrm{d}}_l &= \hat{\mathrm{d}}_{l-1} + K_l (y_l - G \hat{\mathrm{d}}_{l-1} - GF \mathrm{u}_l) \\
\bar{\Sigma}_l &= \Sigma_{l-1} + \Sigma_\omega \\
K_l &= \bar{\Sigma}_l G^{\mathrm{T}} (G \bar{\Sigma}_l G^{\mathrm{T}} + \Sigma_\eta)^{-1} \\
\Sigma_l &= (I - K_l G) \bar{\Sigma}_l
\end{aligned}$$

Input update

- The goal is to compensate for the disturbances by updating u adequately: $\min_{\mathrm{u}_{l+1}} \mathbb{E}[\mathrm{x}_{l+1}|\mathrm{y}_1, \ldots, \mathrm{y}_l] = \min_{\mathrm{u}_{l+1}} F\mathrm{u}_{l+1} + \hat{\mathrm{d}}_l$

Input update

- The goal is to compensate for the disturbances by updating u adequately: $\min_{\mathrm{u}_{l+1}} \mathbb{E}[\mathrm{x}_{l+1}|\mathrm{y}_1, \ldots, \mathrm{y}_l] = \min_{\mathrm{u}_{l+1}} F\mathrm{u}_{l+1} + \hat{\mathrm{d}}_l$
- We can also add (lifted vector) constraints in a convex optimization framework:

Input update

- The goal is to compensate for the disturbances by updating u adequately: $\min_{\mathrm{u}_{l+1}} \mathbb{E}[\mathrm{x}_{l+1}|\mathrm{y}_1, \ldots, \mathrm{y}_l] = \min_{\mathrm{u}_{l+1}} F\mathrm{u}_{l+1} + \hat{\mathrm{d}}_l$
- We can also add (lifted vector) constraints in a convex optimization framework:

$$\mathrm{u}_{l+1} = \arg\min_{u} \|F\mathrm{u} + \hat{\mathrm{d}}_l\|_2$$

$$\text{s.t.}$$

$$L\mathrm{u} \leq q$$

Input update

- The goal is to compensate for the disturbances by updating u adequately: $\min_{\mathrm{u}_{l+1}} \mathbb{E}[\mathrm{x}_{l+1}|\mathrm{y}_1, \ldots, \mathrm{y}_l] = \min_{\mathrm{u}_{l+1}} F\mathrm{u}_{l+1} + \hat{\mathrm{d}}_l$
- We can also add (lifted vector) constraints in a convex optimization framework:

$$\mathrm{u}_{l+1} = \arg\min_u \|F\mathrm{u} + \hat{\mathrm{d}}_l\|_2$$
$$\text{s.t.}$$
$$L\mathrm{u} \leq q$$

- $\mathbf{u}(t) = \mathbf{u}^*(t) + \mathrm{u}_{l+1}(t)$

Example

- Wind disturbance during quadrocopter operation.

Example

- Wind disturbance during quadrocopter operation.
- 2D Quadrocopter dynamics modified as follows:

Example

- Wind disturbance during quadrocopter operation.
- 2D Quadrocopter dynamics modified as follows:

$$\ddot{y} = -f_{\text{coll}} \sin\phi + P_{wind} Asin(\theta + \phi)cos\theta$$
$$\ddot{z} = f_{\text{coll}} \cos\phi - g + P_{wind} Asin(\theta + \phi)sin\theta$$
$$\dot{\phi} = \omega_x$$

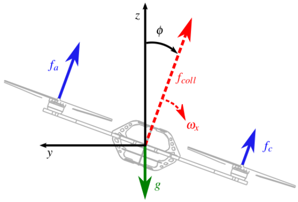

Figure : 2D Quadrocopter model

Example

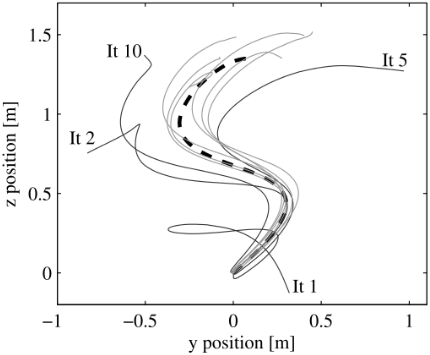

Figure : Learning an S-shaped trajectory

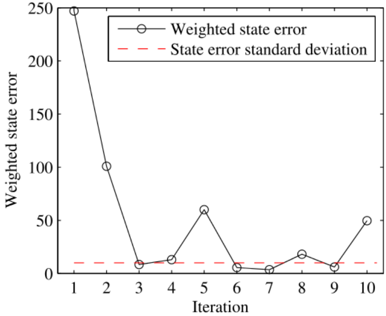

Figure : Errors vs. iterations

Analysis of the method

- Works great, especially with a feedback controller!

Analysis of the method

- Works great, especially with a feedback controller!
- No guarantee of stability!

Analysis of the method

- Works great, especially with a feedback controller!
- No guarantee of stability!
- No guarantee of convergence!

Analysis of the method

- Works great, especially with a feedback controller!
- No guarantee of stability!
- No guarantee of convergence!
  - Modelling disturbances d_I as 'random walk' is not accurate!

Analysis of the method

- Works great, especially with a feedback controller!
- No guarantee of stability!
- No guarantee of convergence!
  - Modelling disturbances d_l as 'random walk' is not accurate!
  - $\mathrm{x}_l = F_{\mathrm{nom}}\mathrm{u}_l + \Delta F \mathrm{u}_l + \epsilon$

Analysis of the method

- Works great, especially with a feedback controller!
- No guarantee of stability!
- No guarantee of convergence!
  - Modelling disturbances d_l as 'random walk' is not accurate!
  - $\mathrm{x}_l = F_{\mathrm{nom}}\mathrm{u}_l + \Delta F \mathrm{u}_l + \epsilon$
  - Is it possible to estimate ΔF from the inputs $(\mathbf{u}_1, \ldots, \mathbf{u}_l)$ and output trajectories $(\mathbf{y}_1, \ldots, \mathbf{y}_l)$?

Analysis of the method

- Works great, especially with a feedback controller!
- No guarantee of stability!
- No guarantee of convergence!
  - Modelling disturbances d_l as 'random walk' is not accurate!
  - $\mathrm{x}_l = F_{\mathrm{nom}}\mathrm{u}_l + \Delta F \mathrm{u}_l + \epsilon$
  - Is it possible to estimate ΔF from the inputs $(\mathbf{u}_1, \ldots, \mathbf{u}_l)$ and output trajectories $(\mathbf{y}_1, \ldots, \mathbf{y}_l)$?
- In other nonlinear ILC approaches [Xu11], contraction mappings and Lyapunov analysis is used to show convergence and stability.

Transfer Learning with ProMPs

- ILC has to learn from scratch each time a trajectory changes!

Transfer Learning with ProMPs

- ILC has to learn from scratch each time a trajectory changes!
- Guilherme and Geri's idea:

Transfer Learning with ProMPs

- ILC has to learn from scratch each time a trajectory changes!
- Guilherme and Geri's idea:
  - Train a ProMP with all of the ILC iterations as demonstrations.

Transfer Learning with ProMPs

- ILC has to learn from scratch each time a trajectory changes!
- Guilherme and Geri's idea:
  - Train a ProMP with all of the ILC iterations as demonstrations.
  - Generalize to different endpoints or trajectories.

Conclusion

- Thank you for listening!

References I

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

AngelaP. Schoellig, FabianL. Mueller, and Raffaello DAndrea.
Optimization-based iterative learning for precise quadrocopter trajectory tracking.
Autonomous Robots, 33:103–127, 2012.

Jian-Xin Xu.
A survey on iterative learning control for nonlinear systems.
International Journal of Control, 84(7):1275–1294, 2011.

References II

Chaojie Zhang, Nan Wang, and Jing Chen.
Trajectory generation for aircraft based on differential flatness and spline theory.
In *Information Networking and Automation (ICINA), 2010 International Conference on*, volume 1, pages V1–110 –V1–114, oct. 2010.