LEARNING THROUGH EXPERIENCE

Optimizing Performance by Repetition

**GOAL**
Enabling a system to improve its own performance by
- repeatedly executing a given task and
- updating the input after each execution

**STRATEGY**
Adapting the input based on
- data of previous trials and
- a-priori knowledge about the system’s basic dynamics

**APPROACH**

**Prepare your model**

System Dynamics
Deriving a mathematical model of the real-world system

\[ x(t) = A x(t) + B u(t) + w(t) \]
\[ y(t) = C x(t) + e(t) \]

Taking input and state constraints explicitly into account
\[ x_{\text{min}} \leq x(t) \leq x_{\text{max}} \]
\[ u_{\text{safe}} \leq u(t) \leq u_{\text{safe}} \]

Defining a desired trajectory (feasible in the context above)
\[ x_{\text{d}}(t), y_{\text{d}}(t), t \in [0, T] \]

Considering only small deviations from the nominal trajectory
\[ \delta x(t) = x(t) - x_{\text{d}}(t), \delta y(t) = y(t) - y_{\text{d}}(t) \]

**Lifted-Domain Representation**

Linearizing and discretizing with
\[ \delta x(k+1) = A \delta x(k) + B \delta y(k) + \delta w(k), k \in \{0, \ldots, N\} \]

Lifting the system assuming \( \delta y(0) = 0 \)

\[ \begin{bmatrix} \delta x(0) \\ \delta y(0) \\ \delta x(1) \\ \vdots \\ \delta x(N) \\ \delta y(N) \end{bmatrix} = \begin{bmatrix} A & 0 & 0 & \cdots & 0 \\ 0 & A & 0 & \cdots & 0 \\ 0 & 0 & A & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A \end{bmatrix} \begin{bmatrix} \delta x(0) \\ \delta y(0) \\ \delta x(1) \\ \vdots \\ \delta x(N) \\ \delta y(N) \end{bmatrix} \]

where \( \Phi_{i,j} = A^i \delta x(j), i = 0, \ldots, N, j = 0, \ldots, N \)

**Perform the algorithm**

Execution
Performing a new trial by applying the updated input
\[ u_j \in \mathbb{R}^{n_u} \]

Starting the first trial, \( u_0 \) with the reference input obtained from the nominal model, i.e., \( u_0 = u^* \)

Measuring the output
\[ y_j \in \mathbb{R}^{n_y} \]

Estimation
Using a discrete-time Kalman filter for the system
\[ \hat{\delta y}_j = \hat{\delta y}_{j-1} + \hat{w}_j-1 \]
\[ y_j = C \hat{\delta x}_j + (C \delta x + \delta w_j), j \in \{1, \ldots, N\} \]

Estimating the modeling error \( \hat{\delta y}_j \) along the desired trajectory

Control
Compensating for the error estimate \( \hat{\delta y}_j \)
\[ \min \left\{ \| Fu_{j+1} + d_j \| \right\} \]

subject to
\[ u_{\text{min}} - u^* \leq u_{j+1} = \hat{w}_j + \hat{d}_j \leq u_{\text{max}} - u^* \]
\[ x_{\text{safe}} - x^* \leq \hat{w}_j + \hat{d}_j \leq x_{\text{safe}} - x^* \]

where \( d_{j+1} \) is approximated by \( d_{j+1} = \Phi_{j+1} d_j \)

**Get results**

A benchmark problem: Swing-up a pendulum

**CONCLUSION**

What are the advantages of the presented approach?
Input and state constraints directly taken into account.
Noise characteristics explicitly considered.
Different optimization goals possible by choosing appropriate norm and weighting.
Extending horizon guarantees that linearization assumptions hold all times

Why is machine learning a good approach?

ADA**P**T**I**ON by repetition

LEARNING through experience

monch**A**tch between model and reality

system UNCERTAINTY and environment COMPLEXITY