Efficient Calibration of Embedded MPC

Marco Forgione[1], Dario Piga[1], and Alberto Bemporad [2]

[1]IDSIA Dalle Molle Institute for Artificial Intelligence SUPSI-USI, Lugano, Switzerland

[2]IMT School for Advanced Studies Lucca, Lucca, Italy

IFAC 2020 World Congress
Berlin, Germany

Motivations

Calibration of model predictive controllers is costly and time-consuming. One has to choose model, cost function weights, prediction/control horizon, design a state estimator, etc.

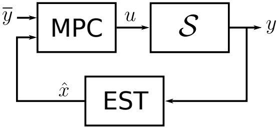
The design is even more involved for embedded applications, where the hardware is also a limit. Need to choose carefully:

- Solution strategy (QP solver?)
- Low-level solver settings (e.g., tolerances)
- A feasible MPC sampling time

In this paper, we introduce an approach to ease embedded MPC calibration based on global optimization and repeated experiements.

Control architecture

Model Predictive Controller



Linear constrained MPC for a (non-linear) plant with model

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t);\ \beta) \\ y(t) &= g(x(t);\ \beta) \end{aligned} \qquad \Rightarrow \qquad \begin{aligned} x_{t+1} &= A(T_s^{\mathrm{MPC}}, \beta)x_t + B(T_s^{\mathrm{MPC}}, \beta)u_t \\ y_t &= C(\beta)x_t \end{aligned}$$

input u chosen on-line as the solution (applied in receding horizon) of:

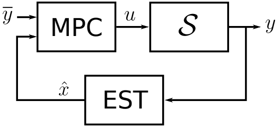
$$\min_{\{u_{t+k|t}\}_{k=1}^{N_c}} \sum_{k=1}^{N_p} \left\| \hat{y}_{t+k|t} - \overline{y}_{t+k} \right\|_{Q_y}^2 + \left\| u_{t+k|k} - u_{t+k-1|t} \right\|_{Q_{\Delta u}}^2$$

s.t. model equations + constraints on y, u, Δu

Control architecture

Design variables

The MPC design variables to be tuned are



- Model parameters: β
- MPC sampling time: T_s^{MPC}
- MPC cost function: $Q_y, Q_{\Delta u}, N_c, N_p$
- MPC implementation: QP solver tolerances $\epsilon_{\mathrm{abs}}, \epsilon_{\mathrm{rel}}$
- State estimator: Kalman covariances W_v, W_w

MPC calibration procedure

Overview

In our optimization-based MPC calibration, we define

- Tunable design parameters collected in a design vector $\theta \in \Theta$.
- A closed-loop performance index J defined in terms of measured input/outputs during the calibration experiment: $J = J(y_{1:T}, u_{1:T};\ \theta)$
- An procedure to perform calibration experiments (or SIL simulations) representative of the intended closed-loop operation

MPC calibration is seen as a global optimization problem:

$$\theta^{\text{opt}} = \arg\min_{\theta \in \Theta} J(y_{1:T}, u_{1:T};\ \theta)$$

$$\text{s.t. } T_{\text{calc}}^{\text{MPC}}(\theta) \leq \eta T_s^{\text{MPC}}$$

each (noisy) function evaluation correspond to a calibration experiment. Both J and $T_{\text{calc}}^{\text{MPC}}$ (worst-case MPC computation time) are observed!

MPC calibration procedure

Overview

In our optimization-based MPC calibration, we define

- Tunable design parameters collected in a design vector $\theta \in \Theta$.
- A closed-loop performance index J defined in terms of measured input/outputs during the calibration experiment: $J = J(y_{1:T}, u_{1:T};\ \theta)$
- An procedure to perform calibration experiments (or SIL simulations) representative of the intended closed-loop operation

MPC calibration is seen as a global optimization problem:

$$\theta^{\text{opt}} = \arg\min_{\theta \in \Theta} J(y_{1:T}, u_{1:T};\ \theta)$$
$$\text{s.t. } T_{\text{calc}}^{\text{MPC}}(\theta) \leq \eta T_s^{\text{MPC}}$$

each (noisy) function evaluation correspond to a calibration experiment. Both J and $T_{\text{calc}}^{\text{MPC}}$ (worst-case MPC computation time) are observed!

Global Optimization Algorithm

Overview

Several global optimization algorithms exist:

- Response surface methods
- Bayesian Optimization
- Genetic algorithms
- Particle Swarm Optimization
- ...

Here, we adopt GLIS: a method recently introduced in (Bemporad, 2019).

- Radial basis function surrogate + inverse distance weighting (IDW) for exploration
- Performs better than BO on standard benchmarks, at a lower computational cost

Global Optimization Algorithm

Overview

Several global optimization algorithms exist:

- Response surface methods
- Bayesian Optimization
- Genetic algorithms
- Particle Swarm Optimization
- . . .

Here, we adopt GLIS: a method recently introduced in (Bemporad, 2019).

- Radial basis function surrogate + inverse distance weighting (IDW) for exploration
- Performs better than BO on standard benchmarks, at a lower computational cost

GLIS Algorithm
Surrogate function

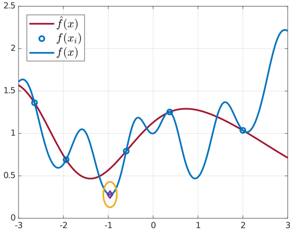
Goal: solve global optimization problem:

$$\min_{\theta \in \Theta} J(\theta)$$

1. Collect n_{in} initial observations $\{(\theta_1, J_1), (\theta_2, J_2), \ldots, (\theta_{n_{\text{in}}}, J_{n_{\text{in}}})\}$
2. Build a surrogate function

$$\hat{J}(\theta) = \sum_{i=1}^{n} \alpha_i \phi(\|\theta - \theta_i\|_2)$$

True $J(\theta)$, surrogate $\hat{J}(\theta)$



ϕ = radial basis function

Example: $\phi(d) = \frac{1}{1+(\epsilon d)^2}$

Minimizing $\hat{J}(\theta)$ to find θ_{n+1} may easily miss the global optimum!

GLIS Algorithm

Surrogate function

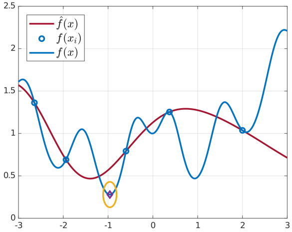
Goal: solve global optimization problem:

$$\min_{\theta \in \Theta} J(\theta)$$

1. Collect n_{in} initial observations $\{(\theta_1, J_1), (\theta_2, J_2), \ldots, (\theta_{n_{\text{in}}}, J_{n_{\text{in}}})\}$
2. Build a surrogate function

$$\hat{J}(\theta) = \sum_{i=1}^{n} \alpha_i \phi(\|\theta - \theta_i\|_2)$$

True $J(\theta)$, surrogate $\hat{J}(\theta)$



ϕ = radial basis function

Example: $\phi(d) = \frac{1}{1+(\epsilon d)^2}$

Minimizing $\hat{J}(\theta)$ to find θ_{n+1} may easily miss the global optimum!

GLIS Algorithm

Exploration vs. Exploitation

3. Construct the IDW exploration function

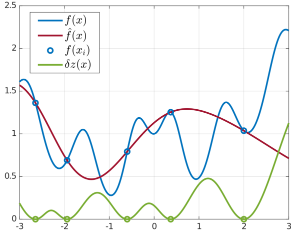
$$z(\theta) = \frac{2}{\pi}\Delta F \tan^{-1}\left(\frac{1}{\sum_{i=1}^{n} w_i(\theta)}\right)$$

where $w_i(\theta) = \frac{e^{-\|\theta-\theta_i\|^2}}{\|\theta-\theta_i\|^2}$

4. Optimize the acquisition function:

$$\theta_{n+1} = \arg\min_{\theta\in\Theta} \hat{J}(\theta) - \delta z(x)$$

Exploration function $z(\theta)$



δ: exploitation vs. exploration trade-off

to get the query point θ_{n+1}. Execute experiment with θ_{n+1}, measure J_{n+1}.

Iterate the procedure for $n+2, n+3 \ldots$.

GLIS Algorithm

Exploration vs. Exploitation

3. Construct the IDW exploration function

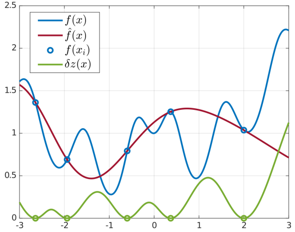
$$z(\theta) = \frac{2}{\pi}\Delta F \tan^{-1}\left(\frac{1}{\sum_{i=1}^{n} w_i(\theta)}\right)$$

where $w_i(\theta) = \frac{e^{-\|\theta-\theta_i\|^2}}{\|\theta-\theta_i\|^2}$

4. Optimize the acquisition function:

$$\theta_{n+1} = \arg\min_{\theta\in\Theta} \hat{J}(\theta) - \delta z(x)$$

Exploration function $z(\theta)$



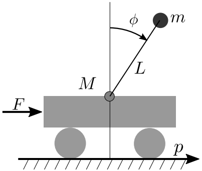
δ: exploitation vs. exploration trade-off

to get the query point θ_{n+1}. Execute experiment with θ_{n+1}, measure J_{n+1}.

Iterate the procedure for $n+2, n+3 \ldots$.

MPC Calibration Example

Cart-pole system



- Input: force F with fast disturbance (5 rad/sec)
- Output: noisy position p
- Objective: follow trajectory for p, keep ϕ small.

Optimization-based calibration of

1. MPC sample time T_s^{MPC}
2. MPC weights Q_y and $Q_{\Delta u}$
3. Prediction and control horizon
4. Kalman filter covariance matrices
5. QP solver's abs. and rel. tolerances

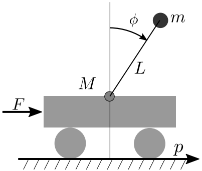
$n_\theta = 14$ design parameters optimized using $n = 500$ iterations of GLIS

$$J = \int_0^{T_{\exp}} |\overline{p}(t) - p(t)| + 30|\phi(t)| \, dt$$

Python codes on-line, 100% open source (including all dependencies).
https://github.com/forgi86/efficient-calibration-embedded-MPC

MPC Calibration Example

Cart-pole system



- Input: force F with fast disturbance (5 rad/sec)
- Output: noisy position p
- Objective: follow trajectory for p, keep ϕ small.

Optimization-based calibration of

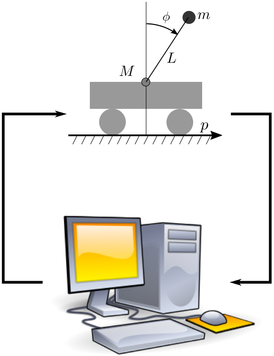
1. MPC sample time T_s^{MPC}
2. MPC weights Q_y and $Q_{\Delta u}$
3. Prediction and control horizon
4. Kalman filter covariance matrices
5. QP solver's abs. and rel. tolerances

$n_\theta = 14$ design parameters optimized using $n = 500$ iterations of GLIS

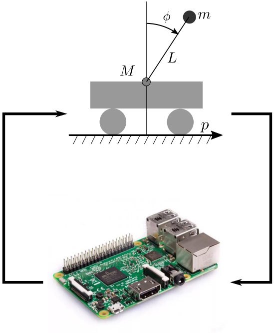
$$J = \int_0^{T_{\exp}} |\bar{p}(t) - p(t)| + 30|\phi(t)| \, dt$$

Python codes on-line, 100% open source (including all dependencies).
`https://github.com/forgi86/efficient-calibration-embedded-MPC`

MPC Calibration Example



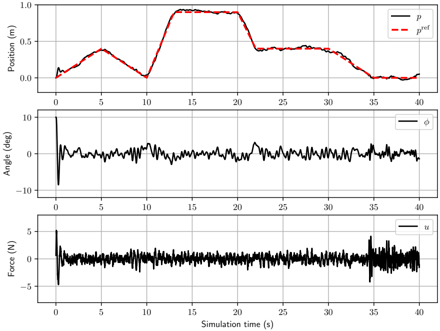
vs.



- An Intel i5 PC (left) vs. an ARM-based Raspberry PI 3 (right)
- PI is about 10 time slower than the PC for MPC computations

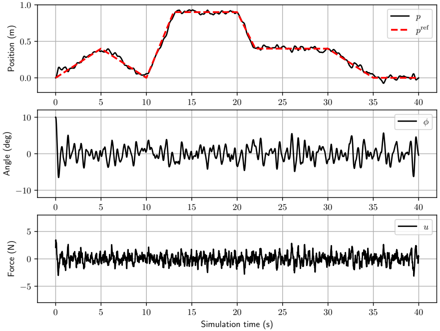
MPC Calibration Example

Optimized MPC on the PC



$$T_s^{\mathrm{MPC}} = 6 \text{ ms}$$

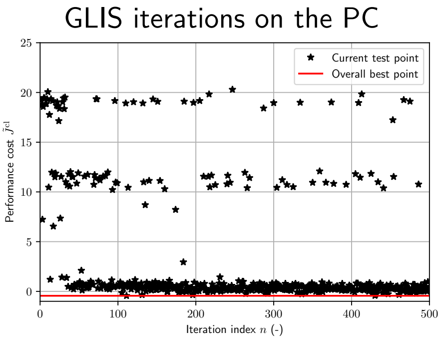
Optimized MPC on the Raspberry PI

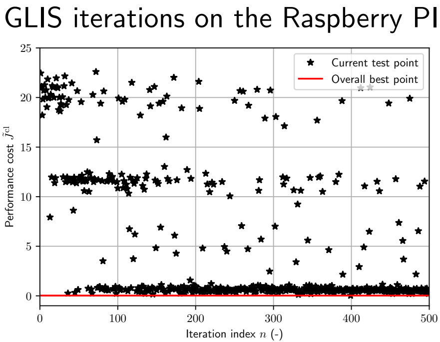


$$T_s^{\mathrm{MPC}} = 22 \text{ ms}$$

- Position and angle control tighter on the PC
- Faster loop update on the PC $\Rightarrow$ more effective disturbance rejection
- Calibration squeezes max performance out of the hardware!

MPC Calibration Example



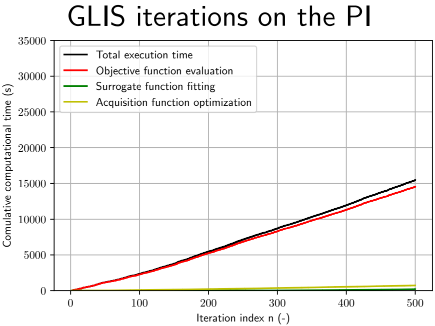


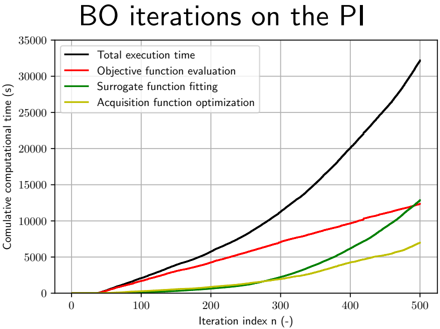
- High cost for interrupted tests/unfeasible design
- For increasing n, more points have "low" cost
- Experiments with high cost still appear for large n (exploration)
- Optimum on PC is slightly better than on the PI. Some configurations are only feasible on PC!

Simulation Example

Cart-pole system

Similar calibration results are obtained using Bayesian Optimization (BO). However, BO is a much heavier algorithm!





- With GLIS, the overall computation time is dominated by function evaluations (= running MPC and simulating the system).

Conclusions

An approach for embedded MPC calibration approach based on global optimization and repeated experiments

- Higher- and lower-level settings tuned altogether to maximize performance, while keeping the design feasible
- Tested in simulation on two hardware platforms

Current/future works

- Application to robotic systems
- Preference-based MPC calibration

Conclusions

An approach for embedded MPC calibration approach based on global optimization and repeated experiments

- Higher- and lower-level settings tuned altogether to maximize performance, while keeping the design feasible
- Tested in simulation on two hardware platforms

Current/future works

- Application to robotic systems
- Preference-based MPC calibration

Thank you.
Questions?