2018 European Control Conference (ECC)
June 12-15, 2018. Limassol, Cyprus

Introducing a Model-based Learning Control Software for Nonlinear Systems: ROFALT*

Armin Steinhauser and Jan Swevers

Abstract— **This paper introduces ROFALT, an open-source, model-based iterative learning control (ILC) tool for nonlinear systems, that aims at closing the gap between the theory of nonlinear ILC and successful applications. Providing a simple yet powerful syntax in MATLAB, ROFALT supports all phases of the design of a nonlinear ILC—from modeling, tuning and execution, to analysis. The theoretical basis is an optimization-based two-step approach that allows an easy trade-off between fast convergence and robustness for generic nonlinear systems. To demonstrate the efficiency of the developed tool, a simulation study on an overhead crane is performed, where a model with introduced parameter deviations is used to iteratively learn the open-loop control inputs. Special attention is paid to the comparison of different possible ways to realize the learning effect and the resulting performance. Moreover, the simplicity of considering constraints and their compliance is demonstrated, while fast convergence is observed.**

I. INTRODUCTION

The ever present drive in industry to enhance performance of mechatronic systems—be it by improving accuracy, increasing speed or by other means—pushes research not only to revise and improve existing technologies, but also to investigate new approaches. In the 1980s, iterative learning control (ILC) was considered to be such a new approach, enabled by the rise of computers, yielding improved performance for repetitive tasks by only manipulating the actuation based on measurements from the preceding executions [1], [2]. The fact that ILC exploits available resources and that the achieved improvement therefore comes at little to no cost makes this control strategy very appealing for industry, which in turn attracted additional attention of the scientific community.

While the initial idea of ILC involved a rather simple gain-based update of the actuation signals [3], a plausible extension was to consider linear systems and analyze the properties of this control strategy, e.g. stability or convergence speed. Over the last 30 years, this branch of ILC —called linear ILC— was well elaborated by both theoretical proofs and experimental validations [4], showing its capabilities and applications. In the case of nonlinear system dynamics, however, the restriction to linear systems in general implicates a suboptimal control performance [5]. Crucial knowledge of nonlinear effects is disregarded and the nonlinearities are captured by an unnecessarily big uncertainty bound, which is reflected in a conservative controller design. This in turn can yield bad convergence properties, a suboptimal converged error result or even lead to unexpected instability.

A number of approaches to expand the idea of ILC to nonlinear systems was presented [6], but they all share the disadvantage of requiring a particular system structure. A fully generic approach was first presented in [7] and elaborated in [8]. It is shown that norm-optimal ILC can be interpreted as a two-step procedure: At first computing an explicit model correction and subsequently inverting the corrected system dynamics. Within this framework, no restrictions are imposed on the system structure or its complexity and both steps are formulated as optimization problems, which additionally allows to consider actuator limits or dynamic constraints, such as limited velocity or acceleration.

Following the approach of a model-based two-step algorithm, this paper presents ROFALT, which stands for *robust and fast learning tool*—a novel learning control tool for MATLAB that implements such an optimization-based approach to nonlinear ILC. As a universal nonlinear ILC package, ROFALT is the first of its kind. Moreover, it is independent of any other MATLAB toolbox, freely available[1] and open-source, licensed under the permissive GNU LGPLv3 [9] to allow use in proprietary software. The main purpose of the development of ROFALT is to close the gap between the theory of nonlinear ILC and its application, by eliminating the need for knowledge of the underlying concepts, e.g. nonlinear optimization. In the end, the goal is to enable a controller design with just a handful of simple commands, where the trade-off between fast convergence and robustness can easily be adjusted. At the same time ROFALT allows profound adaptations, such that conditions and limitations of individual applications can be taken into account.

The remainder of this paper is organized as follows. Section II gives an overview of the theoretical background of the implemented software tool. Section III deals with the software itself by describing the steps required to obtain a learning controller. A simulation study of an overhead crane setup is given together with a comparison of different model correction methods of the software in Section IV and Section V concludes this paper.

*This work has been carried out within the framework of Flanders Make's ICON project 'RoFaLC' (Robust and Fast Learning Control, IWT.150531) funded by the agency Flanders Innovation & Entrepreneurship (VLAIO) and Flanders Make. This work also benefits from the projects G0A6917N and G.0915.14 of the Research Foundation - Flanders (FWO) and KU Leuven-BOF PFV/10/002 Centre of Excellence: Optimization in Engineering (OPTEC). Flanders Make is the Flemish strategic research centre for the manufacturing industry.

The authors are with the MECO Research Team within the Department of Mechanical Engineering, KU Leuven, Belgium, and the Flanders Make DMMS lab, Leuven, Belgium. Contact: armin.steinhauser@kuleuven.be

[1] https://gitlab.mech.kuleuven.be/meco-software/rofalt/

II. THEORETICAL BACKGROUND

A. Notation

In this paper we consider discrete-time dynamics and denote a generic nonlinear, multiple-input, multiple-output (MIMO) model as

$$\begin{aligned} x_{k+1} &= f\left(x_k, u_k, \alpha_k\right), \\ y_k &= h\left(x_k, u_k, \alpha_k\right), \end{aligned} \tag{1}$$

with states $x_k \in \mathbb{R}^{n_x}$, inputs $u_k \in \mathbb{R}^{n_u}$, outputs $y_k \in \mathbb{R}^{n_y}$ and correction terms $\alpha_k \in \mathbb{R}^{n_a}$ for $k \in \{0, \ldots, N-1\}$, where discrete time instants are denoted by a subscript k and N represents the number of considered samples. Remark that while most ILC controllers rely on output corrections, the generic notation of correction terms in (1) captures all possible configurations w.r.t. where and how the correction terms appear in the model. We speak of *parametric correction* when the correction terms directly affect system parameters and of *nonparametric correction* in all other formulations, where in both cases the influence can be additive, multiplicative or of any other kind. The initial state vector x_0 is assumed to be known and we refer to

$$\begin{aligned} f &: \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_a} \to \mathbb{R}^{n_x} \quad \text{and} \\ h &: \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_a} \to \mathbb{R}^{n_y} \end{aligned}$$

as the state transition and the output map, respectively. We distinguish the model (1) from the so-called plant that provides input and output data from either a simulation or an experiment, denoted as $\tilde{u}_k$ and $\tilde{y}_k$, respectively. Whenever a particular iteration i of the ILC is considered, it is indicated by an additional subscript, e.g. $u_{k,[\mathrm{i}]}$, which will be omitted for the sake of readability if unambiguous. Local minimizers of presented optimization problems will be denoted by a star, e.g. $u^\star_{[\mathrm{i}]}$.

B. ILC Algorithm

The software introduced in this paper implements a nonlinear iterative learning control algorithm that consists of a model correction step and a consecutive control step, each of them comprising a nonlinear program (NLP). The advantage of this generic formulation as optimization problems is that the algorithm can theoretically handle any nonlinear MIMO model, while on the downside one must bear in mind that these NLPs yield local minima and a globally optimal solution can therefore not be claimed. As shown in Fig. 1, every iteration i of the ILC algorithm is initiated by gathered input and output data. Note that this implies that input and output data $\tilde{u}_{k,[1]}$ and $\tilde{y}_{k,[1]}$ is assumed to be at hand prior to the first iteration.

1) Model Correction Step: Given a model described by (1), the correction step aims to find correction terms α_k that first and foremost minimize the difference of the gathered output and the output of the model. This general nonlinear, non-convex problem is cast into an NLP and can be stated as

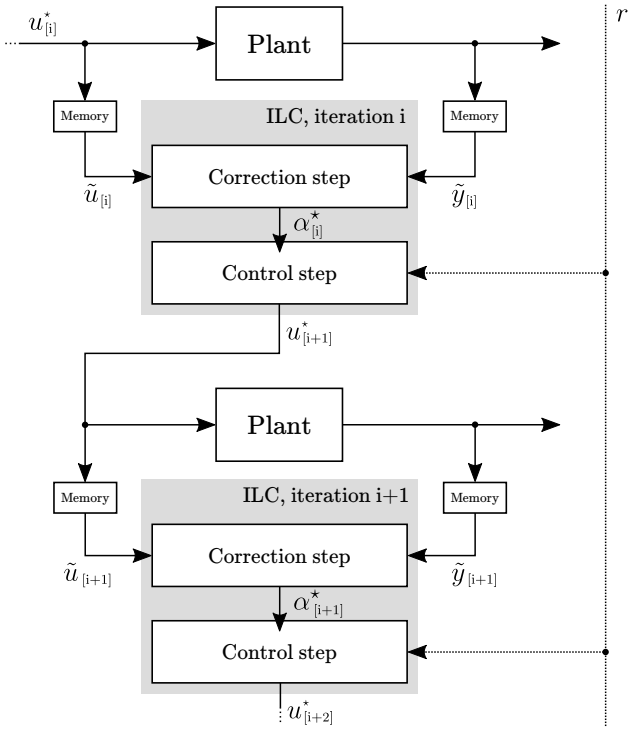

Fig. 1. Schematic overview of the considered two-step learning algorithm

$$\min_{x,\alpha} \quad \frac{1}{2}\sum_{k=0}^{N-1}\Big[\|\tilde{y}_{k,[\mathrm{i}]} - h\left(x_k, \tilde{u}_{k,[\mathrm{i}]}, \alpha_k\right)\|_2^2 + \tag{2a}$$

$$+ \|\alpha_k\|^2_{\mathbf{R}_1} + \|\Delta_i \alpha_k\|^2_{\mathbf{W}_1}\Big] + \tag{2b}$$

$$+ \frac{1}{2}\sum_{k=0}^{N-2}\|\Delta_t \alpha_k\|^2_{\mathbf{W}_2} \tag{2c}$$

$$\text{s.t.} \quad x_{k+1} = f\left(x_k, \tilde{u}_{k,[\mathrm{i}]}, \alpha_k\right), \tag{2d}$$

$$\Delta_t \alpha_k = \alpha_{k+1} - \alpha_k, \quad k = 0, \ldots, N-2, \tag{2e}$$

$$\Delta_i \alpha_k = \alpha_k - \alpha^\star_{k,[\mathrm{i\text{-}1}]}, \tag{2f}$$

$$\underline{\alpha} \le \alpha_k \le \overline{\alpha}, \quad k = 0, \ldots, N-1. \tag{2g}$$

The main objective (2a) represents a nonlinear least squares regression, while the constraints (2d) ensure that the solution complies with the state transition map of the model. Additionally, regularization terms (2b) and (2c) for the corrections and their rate of change w.r.t. time (2e) and iterations (2f) are added, weighted by positive semidefinite matrices $\mathbf{R}_1, \mathbf{W}_1$ and $\mathbf{W}_2$, respectively. These matrices are typically subject to tuning and influence the convergence rate and robustness of the model correction. Finally, bounds on the correction terms can be included as constraints (2g). As a result, the optimal

set of correction terms $\alpha^\star_{[\mathrm{i}]}$ and thus a corrected model

$$\begin{aligned} x_{k+1} &= f\left(x_k, u_k, \alpha^\star_{k,[\mathrm{i}]}\right), \\ y_k &= h\left(x_k, u_k, \alpha^\star_{k,[\mathrm{i}]}\right), \end{aligned} \tag{3}$$

is obtained, which is subsequently passed on to the control step.

2) Control Step: Starting from the corrected model (3), the control step is supposed to provide the optimal set of inputs $u^\star$ such that the difference of the model output and a given output reference $r_k \in \mathbb{R}^{n_y}$ is minimized. In the same way as in the first step, this problem is formulated as an NLP,

$$\min_{x,u} \quad \frac{1}{2}\sum_{k=0}^{N-1}\Bigg[\|r_k - h\left(x_k, u_k, \alpha^\star_{k,[\mathrm{i}]}\right)\|_2^2 + \tag{4a}$$

$$+ \|u_k\|^2_{\mathbf{R}_2} + \|\Delta_i u_k\|^2_{\mathbf{W}_3}\Bigg] + \tag{4b}$$

$$+ \frac{1}{2}\sum_{k=0}^{N-2}\|\Delta_t u_k\|^2_{\mathbf{W}_4} \tag{4c}$$

$$\text{s.t.} \quad x_{k+1} = f\left(x_k, u_k, \alpha^\star_{k,[\mathrm{i}]}\right), \tag{4d}$$

$$\Delta_t u_k = u_{k+1} - u_k, \tag{4e}$$

$$k = 0, \ldots, N-2,$$

$$\Delta_i u_k = u_k - u^\star_{k,[\mathrm{i}]}, \tag{4f}$$

$$\underline{u} \le u_k \le \overline{u}, \quad k = 0, \ldots, N-1. \tag{4g}$$

The main objective (4a) again constitutes a nonlinear least squares problem, while the system dynamics are enforced by (4d). While the regularization of the inputs weighted by $\mathbf{R}_2$ is helpful to make the model inversion more robust, the remaining regularization terms in (4b) and (4c), weighted by $\mathbf{W}_3$ and $\mathbf{W}_4$, penalize the change of the inputs w.r.t. time (4e) and iterations (4f), respectively. Note that this regularization can be interpreted as low-pass filtering in iteration and time domain with its use shown in the application in Section IV. Since every actuation will be limited, (4g) ensures that the computed input signals are realizable. As a result of (4) the locally minimizing inputs $u^\star_{[\mathrm{i+1}]}$ are obtained and applied in the next iteration.

C. Adaptations

Due to the separation into two independent steps and the formulation as optimization problems, the specified algorithm can easily be adapted to tackle other problems that do not fit into the framework of reference tracking. The objective (4a) of the control step can be augmented or replaced to allow e.g. minimization of the power consumption or execution time of a given task. In addition, various ways of model-plant mismatch compensation can be implemented in the model correction step.

III. RoFaLT Software

The purpose of RoFaLT is to simplify the required steps to design and implement an optimization-based learning controller by providing a user-friendly interface, while still allowing customizations and therefore providing a maximum of versatility. To formulate and solve the optimization problems, RoFaLT makes use of the open-source tool CasADi[2] [10], which comes as a self-contained MATLAB package.

A. Modeling

The design of a model-based learning controller is naturally initiated by the choice and definition of a model which will be used for the learning process. RoFaLT accepts continuous- and discrete-time models given in symbolic form, defined either with CasADi or MATLAB's Symbolic Math Toolbox. An example of using a continuous-time model—defined by CasADi functions for ordinary differential equations `ode` and output equations `y`—is shown in Code 1, with the sample time `ts` as the only additionally required argument.

```
rofalt = RoFaLT(); % instantiate RoFaLT object
model = rofalt.setModel('ode', ode, ...
                        'output', y, ...
                        'ts', ts);
```

Code 1. Initialization of RoFaLT and model definition

While these functions have a predefined input argument order, an interface defining all signals manually with their symbolic expressions is also provided. If a continuous-time model is given, it is automatically discretized using either a forward Euler or a Runge-Kutta integrator, where the integration scheme can be chosen by adapting the settings of the returned `model` object handle. Aside from serving as the internal model of the controller, specific methods for simulation, linearization and sensitivity analysis can also be called directly.

B. Controller Design

The core functionality of RoFaLT is the controller design, which starts from the generic configuration described in Section II-B. Assuming that a standard tracking problem with given reference `r` and initial states `x0` is considered, all required lines of code to initialize the controller are shown in Code 2.

```
ilc = rofalt.setController('ilc');
ilc.setReference(r);
ilc.setInitialState(x0);
```

Code 2. Initializing a controller in RoFaLT

The returned handle `ilc` is used to change or add properties of the created nonlinear iterative learning controller. RoFaLT by default automatically scales the optimization problem by means of a sensitivity analysis of the provided model and subsequently adds scaling factors to input and correction terms. This automatic scaling is supposed to normalize the problem such that the weighting matrices introduced in (2) and (4) can be interpreted independently of the model. Note that this functionality can be deactivated in the settings of the controller as shown in Code 3, where also other properties can be adapted.

[2] `http://www.casadi.org`

```
ilc.setSettings('auto_scale', false);
```

Code 3. Setting controller properties in RoFaLT

Examining the formulations (2) and (4), one can identify a number of structural similarities. This resemblance is exploited by implementing a generic NLP class which is then used to instantiate the NLPs specific to the two steps of the algorithm—`ilc.correction` and `ilc.control`. Moreover, the created NLPs share symbolic expressions for the variables used to formulate the optimization problems. These expressions are publicly accessible and can be used to define further properties of the ILC, e.g. considering actuator limits in the control step as shown in Code 4.

```
usym = ilc.symbolics.u; % retrieve symbolic input
ilc.control.addInequalityConstraints(usym, ...
                                     u_min, ...
                                     u_max);
```

Code 4. Adding inequality constraints in RoFaLT

This also allows to replace or augment the default objectives and constraints of both steps, as discussed in Section II-C. For the solution of the NLPs IPOPT [11] is utilized, which is included in CasADi.

C. Execution

By the time all required properties of the controller are set, the final task is to iteratively solve the two optimization problems by taking the latest obtained measurements of a plant into account. Assuming the initial data $\tilde{u}_{k,[1]}$ and $\tilde{y}_{k,[1]}$ is given as `umeas` and `ymeas`, respectively, Code 5 shows a minimalistic for-loop to execute the ILC.

```
for i = 1:1:trials
  % run single iteration
  unext = ilc.runIteration(struct('u', umeas, ...
                                  'y', ymeas));
  % execute a measurement with updated inputs
  [umeas, ymeas] = executeMeasurement(unext);
end

% fetch stored results
results = ilc.getResults();
```

Code 5. Iterative learning with RoFaLT

RoFaLT internally stores data like supplied measurements, results of the individual steps, tracking error and more for each iteration, which can be retrieved in a structured format to easily analyze the individual iterations and their intermediate results. To speed up the solution of the optimization problems they are by default automatically warm-started. This is done by initializing the optimization variables of each step to the corresponding result of the preceding iteration. Similar to the automatic scaling functionality, this warm-starting can be deactivated in the settings of the controller or be manually done by providing initial values for the optimization variables.

IV. Application

To demonstrate the capabilities of RoFaLT and the difference between parametric and nonparametric corrections, we consider a tracking problem for an overhead crane setup as shown in Fig. 2.

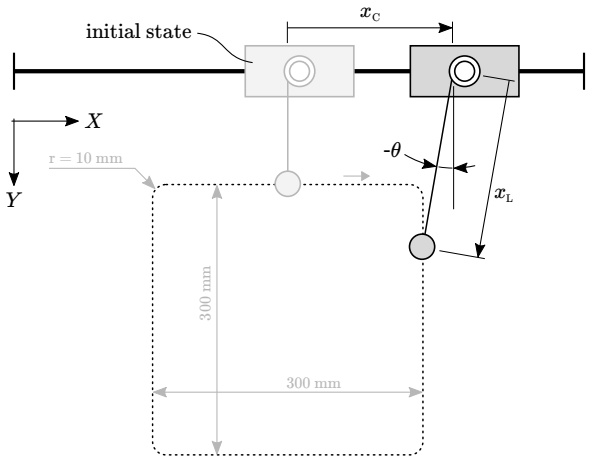

Fig. 2. Sketch of the overhead crane setup, its variables and the considered trajectory (dashed) that it is supposed to track

A. Modeling

1) Nominal Model: To obtain simulation results we introduce a nominal plant $\mathcal{P}_{\text{nom}}$ which is adopted from [12], where the states are gathered as

$$x = [\, x_C,\ v_C,\ x_L,\ v_L,\ \theta,\ \omega \,]^\intercal \tag{5}$$

with the cart position x_C, its velocity v_C, the cable length x_L and its time derivative v_L and the cable angle θ and its time derivative ω. Unlike in [12], to reduce the complexity of the given example, we assume identical gains and time constants for the second-order transfer functions from the voltage references of the cart position and cable length control loop to cart position and cable length

$$\frac{\mathcal{L}\{x_C(t)\}}{\mathcal{L}\{u_C(t)\}} = \frac{\mathcal{L}\{x_L(t)\}}{\mathcal{L}\{u_L(t)\}} = \frac{A}{s\,(\tau s + 1)}. \tag{6}$$

Defining the voltage references as inputs

$$u = [\, u_C,\ u_L \,]^\intercal, \tag{7}$$

the continuous-time dynamics in state space representation can be written as

$$\dot{x} = \begin{bmatrix} v_C \\ -\tau^{-1}(v_C - A\,u_C) \\ v_L \\ -\tau^{-1}(v_L - A\,u_L) \\ \omega \\ -x_L^{-1}\left(\dot{v}_C \cos\theta + g \sin\theta + 2\,v_L\omega\right) \end{bmatrix} \tag{8}$$

with g the gravitational constant. Considering the horizontal and vertical coordinates of the load as the output, the corresponding output equations are given by

$$y = \begin{bmatrix} x_C + x_L \sin\theta \\ x_L \cos\theta \end{bmatrix}. \tag{9}$$

These dynamics are simulated using the `ode45` integrator of MATLAB with the nominal parameter set p_{nom} as stated in Table I, where the applied inputs and the obtained outputs are considered the measurements $\tilde{u}_{k,[i]}$ and $\tilde{y}_{k,[i]}$, respectively.

TABLE I

SETS OF PARAMETERS FOR SIMULATION AND INITIAL MODEL ESTIMATE

Set	$A\ \left[\frac{\mathrm{m}}{\mathrm{s\,V}}\right]$	τ [s]
p_{nom}	$5 \cdot 10^{-2}$	$3 \cdot 10^{-2}$
p_{ilc}	$6 \cdot 10^{-2}$	$7 \cdot 10^{-2}$

2) Model for Controller: The models used by the learning controller have the same structure as the plant $\mathcal{P}_{\text{nom}}$ but start from the parameters p_{ilc} (see Table I) that differ from the nominal values, and include correction terms. The introduced parameter deviations can be physically motivated, for example as an erroneous gain of the hoisting dynamics that reflects an uncertain winder drum diameter. To show the characteristics of parametric and nonparametric corrections, three different scenarios of including correction terms are considered:

a) Scenario I: The first scenario considers parametric corrections of the gains A such that we obtain a model structure that is identical to the plant $\mathcal{P}_{\text{nom}}$ but substitutes $A = A|_{p_{\text{ilc}}} + \alpha$ and $\tau = \tau|_{p_{\text{ilc}}}$, resulting in a total of two correction terms. Note that parametric correction terms like the one introduced here can change over iterations but are time invariant within one trial.

b) Scenario II: The second scenario covers a parametric correction of all parameters, namely the gains A and the time constants τ, such that the model replaces the parameters with $A = A|_{p_{\text{ilc}}} + \alpha_{\text{A}}$ and $\tau = \tau|_{p_{\text{ilc}}} + \alpha_{\tau}$ for each second-order transfer function, resulting in a total of four correction terms.

c) Scenario III: The third scenario considers nonparametric corrections of the outputs such that we obtain a model that replaces the parameters by $A = A|_{p_{\text{ilc}}}$ and $\tau = \tau|_{p_{\text{ilc}}}$, but features adapted output equations

$$y = \begin{bmatrix} x_{\text{C}} + x_{\text{L}} \sin\theta \\ x_{\text{L}} \cos\theta \end{bmatrix} + \alpha \tag{10}$$

where the time-variant nonparametric correction terms are

$$\alpha = [\,\alpha_x,\ \alpha_y\,]^{\intercal} \in \mathbb{R}^{2\times 1}. \tag{11}$$

All models are defined in continuous time and subsequently discretized by RoFaLT with a Runge-Kutta integrator to obtain models in the form of (1).

B. Task Description

The goal of this application is to improve the tracking performance on a square-shaped path as shown in Fig. 2 through iterative updates of the open-loop voltage rate inputs by the learning controller. The reference is given in global coordinates as

$$r_k = [\,X_{\text{ref}},\ Y_{\text{ref}}\,]_k^{\intercal} \tag{12}$$

and spans a period of 7 s—including some standstill at the end—which together with the chosen sampling frequency of 100 Hz results in 700 samples. Additionally, to show the possibility of including input constraints, both voltages u_{C} and u_{L} are limited to ± 10 V.

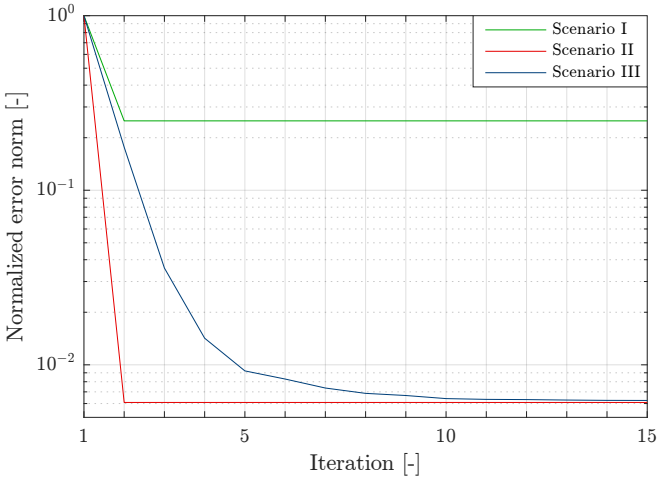

Fig. 3. Normalized tracking error norm evolution over iterations for the considered scenarios

C. Results

As a starting point, the initial couple of measured inputs and outputs is obtained by executing a control step of the proposed ILC algorithm without any correction,

$$\alpha^{\star}_{[0]} = 0 \quad \Rightarrow \quad u^{\star}_{[1]} \quad \Rightarrow \quad \left(\tilde{u}_{[1]},\ \tilde{y}_{[1]}\right), \tag{13}$$

that yields identical results for all considered scenarios. An obvious criterion to assess the control performance is the tracking error or the evolution of its norm over iterations

$$\epsilon_{[\mathrm{i}]} = \sum_{k=0}^{N-1} \|r_k - \tilde{y}_{k,[\mathrm{i}]}\|_2, \tag{14}$$

since this is the main objective in the control step. This evolution can be influenced by the weighting matrices introduced in (2) and (4), which essentially represents a trade-off between convergence speed and robustness. Since noise or external disturbances are not considered in this simulation example, increasing the robustness is not important and we set the weights to

$$\mathbf{R}_1 = \mathbf{R}_2 = 1 \cdot 10^{-3}, \tag{15}$$

$$\mathbf{W}_1 = \mathbf{W}_2 = \mathbf{W}_3 = 0, \tag{16}$$

$$\mathbf{W}_4 = 1 \cdot 10^{-2}, \tag{17}$$

where the non-zero $\mathbf{W}_4$ is used to smoothen the computed feed-forward inputs. The resulting error norm evolutions for the three considered scenarios are shown in Fig. 3, where two distinct properties can be observed: First, scenario II and III yield almost identical converged error norms while scenario I shows a significantly higher value. This results from the inability to find a model correction that captures the introduced parameter deviation, whereas suitable corrections are found for the full parametric or the nonparametric correction. Secondly, the learning controllers considering parametric corrections—scenario I and II—converge in a single iteration, while the nonparametric corrections take longer to converge. This only applies to the noiseless case where parameters can be perfectly estimated based on a

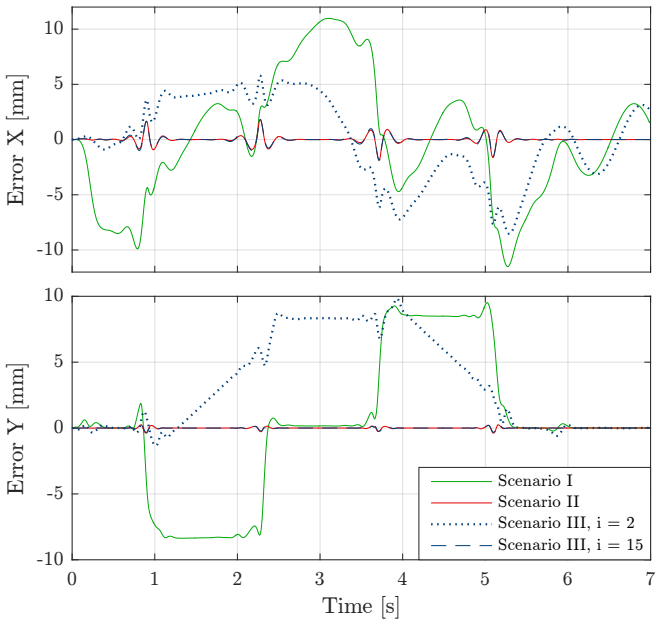

Fig. 4. Tracking error over time in global coordinates; iteration independent results for scenario I & II, while for scenario III iteration 2 and 15 show the convergence; initial tracking error (iteration 1) is an order of magnitude larger and therefore not shown

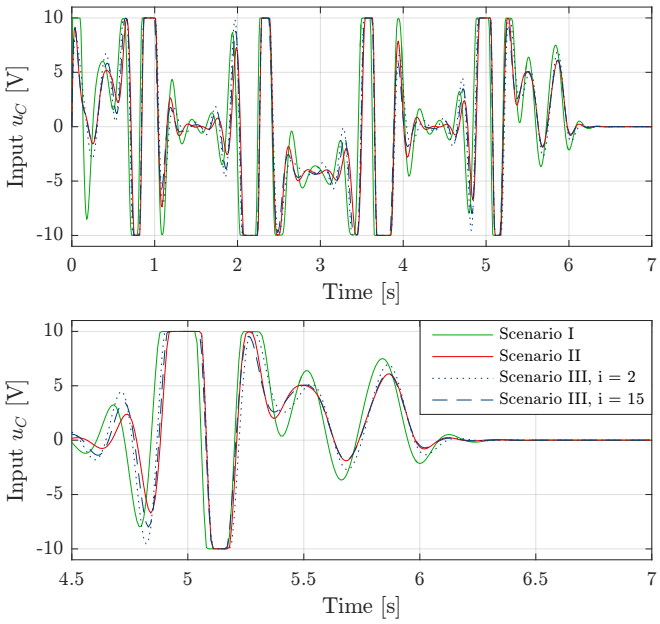

Fig. 5. Cart velocity reference (input u_C) of full trial (top) and a magnified detail (bottom); iteration independent results for scenario I & II, while for scenario III iteration 2 and 15 show the convergence; second input u_L shows the same properties and is therefore omitted

single measurement. Similar conclusions can be drawn from observing the tracking performance in time domain, as shown in Fig. 4. While the learning controllers of scenario II and III converge to approximately the same remaining error, the controller using a single parametric correction in scenario I shows a significantly larger remaining error that is irreducible given the limited ways of model correction. Furthermore, one can observe remaining errors for all scenarios whenever maximal voltages are required, as it is the case in the corners of the given trajectory. This is confirmed by the result for the inputs, see Fig. 5, where the effectiveness of the considered input constraints and the concurrence with the aforementioned remaining error can be seen. While the solutions for scenario II and III differ only slightly, the input for scenario I shows not only a larger difference but also that the deviation of the time constant cannot be compensated.

V. CONCLUSIONS

This paper presents a novel model-based learning tool for MATLAB, called ROFALT, that supports all phases of the design of a nonlinear ILC. Unlike any other available software, this tool implements an optimization-based two-step algorithm which represents a generic approach to nonlinear ILC. The efficiency of ROFALT is demonstrated by improving the tracking performance of an overhead crane in a simulation study. The ILC learns to compensate for the introduced parameter deviations through model correction, where specific consequences of the choice of correction method are shown in the results. Although a parametric correction seems appealing due to its physical interpretation and reusability, limiting the model correction to parameters can lead to significant performance loss. A nonparametric correction constitutes a more generic approach, however, at the expense of slower convergence and validity only for the considered trajectory. ROFALT is actively developed further and future work will include applications to real setups.

REFERENCES

[1] K. L. Moore, *Iterative learning control for deterministic systems*. London: Springer-Verlag, 1993.

[2] W. Messner, R. Horowitz, W.-W. Kao, and M. Boals, "A new adaptive learning rule," *IEEE Trans. Autom. Control*, vol. 36, no. 2, pp. 188–197, February 1991.

[3] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering operation of dynamic systems by learning : A new control theory for servomechanism or mechatronic systems," in *Proc. 23rd Conf. on Decision and Control*, Las Vegas, NV, 1984.

[4] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE Control Syst. Mag.*, vol. 26, no. 3, pp. 96–114, 2006.

[5] J.-X. Xu and Y. Tan, *Linear and Nonlinear Iterative Learning Control*. Berlin Heidelberg: Springer-Verlag, 2003.

[6] J.-X. Xu, "A survey on iterative learning control for nonlinear systems," *Int. J. Control*, vol. 84, no. 7, pp. 1275–1294, 2011.

[7] M. Volckaert, J. Swevers, and M. Diehl, "A two step optimization based iterative learning control algorithm," in *Proc. ASME 2010 Dynamic Systems and Control Conf.*, Cambridge, MA, 2010.

[8] M. Volckaert, M. Diehl, and J. Swevers, "Generalization of norm optimal ilc for nonlinear systems with constraints," *Mechanical Systems and Signal Processing*, vol. 39, no. 1-2, pp. 280–296, 2013.

[9] Free Software Foundation (FSF), "Gnu lesser general public license v3.0," accessed 10 October 2017. [Online]. Available: http://www.gnu.org/licenses/lgpl.html

[10] J. Andersson, "A general-purpose software framework for dynamic optimization," Ph.D. dissertation, Arenberg Doctoral School, KU Leuven, 2013.

[11] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Math. Program.*, vol. 106, no. 1, pp. 25–57, 2006.

[12] M. Vukov, W. Van Loock, B. Houska, H. J. Ferreau, J. Swevers, and M. Diehl, "Experimental validation of nonlinear mpc on an overhead crane using automatic code generation," in *Proc. American Control Conference (ACC)*, Montreal, Canada, 2012, pp. 6264–6269.