2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 7-12, 2012. Vilamoura, Algarve, Portugal

Iterative Learning of Feed-Forward Corrections for High-Performance Tracking

Fabian L. Mueller, Angela P. Schoellig and Raffaello D'Andrea

***Abstract*— We revisit a recently developed iterative learning algorithm that enables systems to learn from a repeated operation with the goal of achieving high tracking performance of a given trajectory. The learning scheme is based on a coarse dynamics model of the system and uses past measurements to iteratively adapt the feed-forward input signal to the system. The novelty of this work is an identification routine that uses a numerical simulation of the system dynamics to extract the required model information. This allows the learning algorithm to be applied to *any* dynamic system for which a dynamics simulation is available (including systems with underlying feedback loops). The proposed learning algorithm is applied to a quadrocopter system that is guided by a trajectory-following controller. With the identification routine, we are able to extend our previous learning results to three-dimensional quadrocopter motions and achieve significantly higher tracking accuracy due to the underlying feedback control, which accounts for non-repetitive noise.**

I. INTRODUCTION

Current control systems usually regulate the behavior of dynamic systems by *reacting* to noise and unexpected disturbances. Typically, they are based on a mathematical model of the system dynamics. The performance of this approach is limited by the accuracy of the dynamics model and the causality of the control action that is compensating only for disturbances as they occur. Unfavorable effects of these limitations are observed especially when operating systems in regimes where feedback is not able to react in time and the dynamic behavior is difficult to identify. To achieve high tracking performance in such cases, we propose data-based control approaches that are able to store and interpret information from past executions, and infer the correct actions for future experiments.

We build upon the iterative learning scheme previously presented [1], [2], which relies on a coarse dynamics model of the system under consideration when interpreting past measurements and updating the feed-forward input after each iteration. In contrast to [1], [2], where the system model was derived analytically (e.g. from first principles), this work introduces a numerical identification routine that extracts the required model information from a dynamics simulation. The novel identification routine allows us to apply the learning algorithm to (complex) systems where no analytical model is available, but where a numerical simulation is. This generalized approach comes at a slightly higher computational cost due to the required numerical model extraction. The resulting learning framework is conceptually simple and allows for an acausal correction, which anticipates and compensates for recurring disturbances before they occur. The latter distinguishes our approach from other iterative approaches, which iteratively update the feedback law (cf. [3] and references therein) or adapt the reference input online [4].

The proposed learning algorithm is applied to quadrotor vehicles in the ETH Flying Machine Arena (FMA), cf. [5]. Quadrocopters offer exceptional agility, and complex effects such as aero- and motor-dynamics have a significant impact on the vehicle behavior. These effects are difficult to model but can be compensated for by iterative learning.

The approach presented in this paper can be characterized as an iterative learning control (ILC) technique. ILC became a popular research topic beginning with [6], and has since proven to be a powerful method for high-performance reference tracking. A recent overview of ILC is available in [7] and [8]. Work in [7], [9]–[11] has shown that ILC can be applied to systems with underlying feedback loops, and [12] first applied ILC to quadrocopter trajectory tracking.

Below we present the learning algorithm, which is introduced as a two-step process of first estimating the unknown repetitive disturbance (Sec. II-B) and later compensating for it (Sec. II-C). The numerical system identification routine is presented in Sec. III. In Sec. V, we apply the derived learning framework to quadrocopters and present the learning performance in actual experiments. A video of the quadrocopter results is found at `http://tiny.cc/SlalomLearning`. We conclude with a discussion of the approach in Sec. VII and summarize the results in Sec. VIII.

II. ITERATIVE LEARNING

The goal of the proposed learning scheme is to use iterative experiments to teach a dynamic system how to precisely follow a desired trajectory, which is defined by a sequence of output values $\mathrm{y}^*(k),\ k \in \{1, 2, \ldots, N\}$, with $N < \infty$ being the trial length in discrete time steps. To improve the tracking performance over iterations, the learning algorithm adapts the feed-forward input values after each experiment, where $\mathrm{u}_j(k),\ k \in \{0, 1, \ldots, N-1\}$, denotes the input of the jth experiment. Fig. 1 shows the basic setup. We use discrete-time representations of signals, which may be obtained by sampling the corresponding continuous-time signals.

The learning algorithm requires knowledge of the system's key dynamics around the desired trajectory, cf. Sec. II-A. The algorithm builds upon this knowledge when exploiting the data from past trials and updating the feed-forward

This research was funded in part by the Swiss National Science Foundation through the National Centre of Competence in Research Robotics.

F. L. Mueller, A. P. Schoellig and R. D'Andrea are with the Institute of Dynamic Systems and Control (IDSC), ETH Zurich, Switzerland. `(famuelle,aschoellig,rdandrea)@ethz.ch`

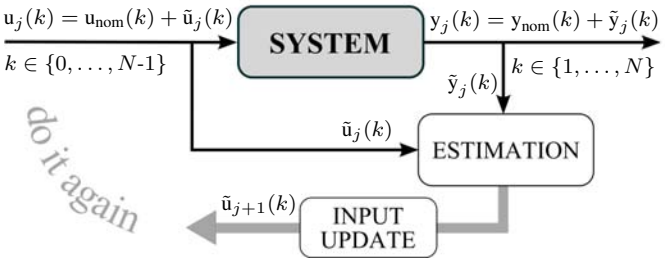

Fig. 1. The general iterative learning framework considered in this paper: A complete trial $\mathrm{u}_j(k), k \in \{0, 1, \ldots, N-1\}$ is performed at iteration j. Based on the output deviation $\tilde{\mathrm{y}}_j(k)$, a new input $\mathrm{u}_{j+1}(k)$ is calculated and applied during the next trial.

input signal for the next trial. A Kalman filter interprets the measurement of the last trial and incorporates it into the current estimate of the modeling error (Sec. II-B). The input update step takes the current estimate and returns a more adequate input for the next trial by solving an optimization problem (Sec. II-C).

The proposed learning scheme can be applied to dynamic systems with underlying feedback loops. In Sec. IV we show that the tracking accuracy of a quadrocopter guided by a trajectory-following controller can be improved through iterative experiments, cf. Fig. 3.

A. System Representation

Our learning scheme is a model-based approach in the sense that it incorporates knowledge about the key dynamics of the physical system under consideration. In particular, we consider a mapping $\mathcal{D}$,

$$\mathcal{D} : U \to (Y, C), \tag{1}$$

that relates the input series $\bar{\mathrm{u}} = (\mathrm{u}(0), \ldots, \mathrm{u}(N-1)) \in U \subset \mathbb{R}^{Nn_\mathrm{u}}$ to the output sequence $\bar{\mathrm{y}} = (\mathrm{y}(1), \ldots, \mathrm{y}(N)) \in Y \subset \mathbb{R}^{Nn_\mathrm{y}}$ and to the constrained sequence $\bar{\mathrm{c}} = (\mathrm{c}(1), \ldots, \mathrm{c}(N)) \in C \subset \mathbb{R}^{Nn_\mathrm{c}}$. The vector $\mathrm{c}(\cdot) \in \mathbb{R}^{n_\mathrm{c}}$ includes all quantities that are subject to constraints. The mapping (1) is not required to reproduce the real system's behavior in the minutest detail but is expected to approximate the dominating dynamics to first order. In Sec. IV-C we show that a first-principles model of the quadrocopter dynamics is sufficient to derive the mapping (1).

The learning algorithm presupposes an initial guess for the input, hereafter referred to as the ***nominal*** input $\bar{\mathrm{u}}_\mathrm{nom} \in U$, yielding $(\bar{\mathrm{y}}_\mathrm{nom}, \bar{\mathrm{c}}_\mathrm{nom}) = \mathcal{D}(\bar{\mathrm{u}}_\mathrm{nom})$. We choose $\bar{\mathrm{u}}_\mathrm{nom}$ such that the resulting $\bar{\mathrm{y}}_\mathrm{nom}$ is close to the desired output sequence $\bar{\mathrm{y}}^* = (\mathrm{y}^*(1), \ldots, \mathrm{y}^*(N))$. This is not required but represents a good starting point for the learning process. Subsequently, we consider deviations from the nominal trajectories,

$$\begin{aligned} \tilde{\mathrm{u}}(k) &= \mathrm{u}(k) - \mathrm{u}_\mathrm{nom}(k)\,, \;\; \tilde{\mathrm{y}}(k) = \mathrm{y}(k) - \mathrm{y}_\mathrm{nom}(k) \\ \tilde{\mathrm{c}}(k) &= \mathrm{c}(k) - \mathrm{c}_\mathrm{nom}(k), \end{aligned} \tag{2}$$

and introduce the lifted vector representation, cf. [13]:

$$\begin{aligned} \mathrm{u} &= [\,\tilde{\mathrm{u}}(0),\, \tilde{\mathrm{u}}(1),\, \ldots,\, \tilde{\mathrm{u}}(N-1)\,]^T \quad \in \mathbb{R}^{Nn_\mathrm{u}} \\ \mathrm{y} &= [\,\tilde{\mathrm{y}}(1),\, \ldots,\, \tilde{\mathrm{y}}(N)\,]^T \quad \in \mathbb{R}^{Nn_\mathrm{y}} \\ \mathrm{c} &= [\,\tilde{\mathrm{c}}(1),\, \ldots,\, \tilde{\mathrm{c}}(N)\,]^T \quad \in \mathbb{R}^{Nn_\mathrm{c}}. \end{aligned} \tag{3}$$

Note that we use u, y, c (in contrast to $\bar{\mathrm{u}}$, $\bar{\mathrm{y}}$, $\bar{\mathrm{c}}$) to represent ***deviations*** from the nominal input, output and constraint sequence (namely, $\bar{\mathrm{u}}_\mathrm{nom}$, $\bar{\mathrm{y}}_\mathrm{nom}$ and $\bar{\mathrm{c}}_\mathrm{nom}$).

The key assumption of the learning algorithm is that static linear mappings

$$\mathrm{y} = F\,\mathrm{u}\,, \qquad \mathrm{c} = L\,\mathrm{u}\,, \tag{4}$$

with $F \in \mathbb{R}^{Nn_\mathrm{y} \times Nn_\mathrm{u}}$ and $L \in \mathbb{R}^{Nn_\mathrm{c} \times Nn_\mathrm{u}}$ can be derived from (1), which capture the main dynamics of the real system along the nominal trajectories $(\bar{\mathrm{u}}_\mathrm{nom}, \bar{\mathrm{y}}_\mathrm{nom}, \bar{\mathrm{c}}_\mathrm{nom})$ by relating the input ***deviation*** time series $\tilde{\mathrm{u}}(k)$, $k \in \{0, 1, \ldots, N-1\}$, to the corresponding time series of output and constrained quantities ***deviations***, $\tilde{\mathrm{y}}(k)$, $\tilde{\mathrm{c}}(k)$, $k \in \{1, 2, \ldots, N\}$.

The mapping (4) is motivated by the fact that any nonlinear dynamics model of the form

$$\dot{\mathrm{x}}(t) = \mathrm{f}\,(\mathrm{x}(t), \mathrm{u}(t), t), \quad \mathrm{y}(t) = \mathrm{g}\,(\mathrm{x}(t), t), \tag{5}$$

can be written as (4) when time-discretized and linearized around the nominal trajectory, cf. [1]. In Sec. III we present an algorithm that identifies the matrices F and L from a dynamics simulation of (1). That is, we extract (4) from a numerical simulation and do not require an explicit analytical representation of (1).

The two steps of the proposed learning algorithm, the disturbance estimation and the input update, fundamentally rely on (4) and are explained in the following.

B. Disturbance Estimation

The purpose of the estimation step is to estimate a correction vector $\mathrm{d} \in \mathbb{R}^{Nn_\mathrm{y}}$ that is added to the first mapping in (4) with the goal of improving the mapping's accuracy. That is, the input-output relation in (4) now reads as

$$\mathrm{y} = F\,\mathrm{u} + \mathrm{d}. \tag{6}$$

The evolution of the learning over a sequence of consecutive trials is modeled by

$$\begin{aligned} \mathrm{d}_{j+1} &= \mathrm{d}_j + \omega_j \\ \mathrm{y}_j &= F\,\mathrm{u}_j + \mathrm{d}_j + \mu_j\,, \end{aligned} \tag{7}$$

where the subscript j indicates the jth execution of the desired task, $j \in \{0, 1, \ldots\}$. The vector d_j can be interpreted as repetitive disturbance along the trajectory, which is primarily caused by unmodeled dynamics. The disturbance vector is subject only to slight random changes ω_j from iteration to iteration. We account for process and measurement noise by adding the random variable μ_j. Both stochastic variables, $\omega_j \sim \mathcal{N}(0, \epsilon_j I)$ and $\mu_j \sim \mathcal{N}(0, \eta_j I)$, are trial-uncorrelated sequences of zero-mean Gaussian white noise and, moreover, assumed to be independent. The scalars μ_j, η_j represent the corresponding variances and I denotes the identity matrix.

We use an iteration-domain Kalman filter, which retains all available information from previous trials (namely the measured output deviations $\{\mathrm{y}_0, \mathrm{y}_1, \ldots, \mathrm{y}_j\}$) and calculates an updated estimate $\widehat{\mathrm{d}}_{j+1}$ of the disturbance vector after each iteration based on the relationship (7). Given initial values for the disturbance estimate and its covariance matrix, $\widehat{\mathrm{d}}_0$ and P_0, respectively, the disturbance estimate is updated according to

$$\widehat{\mathrm{d}}_{j+1} = \widehat{\mathrm{d}}_j \, + \, K_j \left(\mathrm{y}_j - F\,\mathrm{u}_j - \widehat{\mathrm{d}}_j \right) , \tag{8}$$

where K_j is the optimal Kalman gain, cf. [1].

C. Input Update

The learning algorithm is completed by the subsequent input update step. Making use of the information provided by the estimator, cf. Sec. II-B, we derive a model-based update rule that calculates a new input sequence $\overline{\mathrm{u}}_{j+1} \in \mathbb{R}^{N n_\mathrm{u}}$ in response to the estimated disturbance $\widehat{\mathrm{d}}_{j+1}$. The goal of the learning algorithm is to find an input *deviation* vector u_{j+1}, such that the system output of the next iteration $\overline{\mathrm{y}}_{j+1}$ follows the desired trajectory $\overline{\mathrm{y}}^*$ as closely as possible. In the context of (2) and (3), this is equivalent to finding input corrections u_{j+1} that minimize

$$\| \overline{\mathrm{y}}_\mathrm{nom} + \mathrm{y}_{j+1} - \overline{\mathrm{y}}^* \| = \| \check{\mathrm{y}} + \mathrm{y}_{j+1} \| , \tag{9}$$

where $\check{\mathrm{y}}$ is defined as $\check{\mathrm{y}} = \overline{\mathrm{y}}_\mathrm{nom} - \overline{\mathrm{y}}^*$. Since y_{j+1} is unknown, we use its expected value instead,

$$E \left[\mathrm{y}_{j+1} \,\middle|\, \mathrm{y}_0, \mathrm{y}_1, \ldots, \mathrm{y}_j \right] = F\,\mathrm{u}_{j+1} + \widehat{\mathrm{d}}_{j+1} . \tag{10}$$

The complete optimization problem that constitutes the update step is given as:

$$\begin{aligned} \min_{\mathrm{u}_{j+1}} \quad & \left\| S \left(\check{\mathrm{y}} + F \mathrm{u}_{j+1} + \widehat{\mathrm{d}}_{j+1} \right) \right\|_\ell + \, \alpha \left\| D\,\mathrm{u}_{j+1} \right\|_\ell \\ \text{s.t.} \quad & L\,\mathrm{u}_{j+1} \preceq \mathrm{c}_\mathrm{max} , \end{aligned} \tag{11}$$

where the first term accounts for (9) and $\alpha \geq 0$ weights an additional penalty term that was included into the objective function as a means of directly penalizing the input deviation ($D = I$) or, depending on the choice of D, approximations of its derivatives. The original error signal can be scaled and weighted via the diagonal matrix $S \in \mathbb{R}^{N n_\mathrm{y} \times N n_\mathrm{y}}$. The vector norm ℓ, $\ell \in \{1, 2, \infty\}$, in (11) affects the convergence behavior and the result of the learning algorithm. Further, constraints are taken explicitly into account. The lifted vector c_max denotes the maximum allowed *deviations* from nominal values $\overline{\mathrm{c}}_\mathrm{nom}$.

The update law (11) is a convex optimization problem, cf. [14], which can be solved very efficiently by existing software tools such as [15]. Moreover, if the optimization problem is feasible, then there exists a local minimum that is globally optimal.

III. SYSTEM IDENTIFICATION

Both the estimation step (Sec. II-B) and the input update step (Sec. II-C) rely on information about the system dynamics provided by the mapping (4). The goal of the system identification routine is to obtain the mapping matrices F and L from a numerical simulation of $\mathcal{D}$, cf. (1). Using the nominal input $\overline{\mathrm{u}}_\mathrm{nom}$, the simulation provides $(\overline{\mathrm{y}}_\mathrm{nom}, \overline{\mathrm{c}}_\mathrm{nom}) = \mathcal{D}(\overline{\mathrm{u}}_\mathrm{nom})$. The idea of the proposed routine is to identify F and L by running a sequence of simulations with inputs $\overline{\mathrm{u}}$ that differ from the nominal input in exactly one of their (scalar) elements, that is

$$\overline{\mathrm{u}} = \overline{\mathrm{u}}_\mathrm{nom} + \Delta u \cdot \mathrm{e}, \quad \Delta u \ll 1 , \tag{12}$$

where e is a vector of the form $(0, 0, \ldots, 1, 0, 0, \ldots)$ containing exactly one non-zero element. We denote the ith element of the lifted input vector by the superscript (i) and write $\overline{\mathrm{u}}^{(i)} \in \mathbb{R}$. The identification routine is summarized in Algorithm 1. By observing the changes in $\overline{\mathrm{y}}$ and $\overline{\mathrm{c}}$ caused by the change in the input according to (12), the matrices F and L are computed, allowing us to use (4) for predictions. However, these predictions are valid only around the nominal trajectory $\overline{\mathrm{y}}_\mathrm{nom}$. Thus, if $\overline{\mathrm{y}}_\mathrm{nom}$ is far from $\overline{\mathrm{y}}^*$, it might be necessary to *re*-identify the system after some iterations around the current trajectories $(\overline{\mathrm{u}}_j, \overline{\mathrm{y}}_j, \overline{\mathrm{c}}_j)$, $j > 0$, in order to ensure accurate predictions. Note that the mapping $\mathcal{D}$ must be continuous in $\overline{\mathrm{u}}$ to obtain useful approximations for F and L.

Algorithm 1 Identification routine

Require: Nominal input $\overline{\mathrm{u}}_\mathrm{nom}$ and mapping $\mathcal{D}$.

/* * * **Preliminary Step** * * */
Compute $(\overline{\mathrm{y}}_\mathrm{nom}, \overline{\mathrm{c}}_\mathrm{nom}) = \mathcal{D}(\overline{\mathrm{u}}_\mathrm{nom})$.
/* * * **Identification Loop** * * */
Allocate: $F \in \mathbb{R}^{N n_\mathrm{y} \times N n_\mathrm{u}}$, $L \in \mathbb{R}^{N n_\mathrm{c} \times N n_\mathrm{u}}$
Choose input increment $\Delta u \ll 1$, $\Delta u \in \mathbb{R}$.
for $i = 1 : N n_\mathrm{u}$ **do**
  Define simulation inputs: $\overline{\mathrm{u}}_1 = \overline{\mathrm{u}}_2 = \overline{\mathrm{u}}_\mathrm{nom}$
  Change ith element: $\overline{\mathrm{u}}_1^{(i)} = \overline{\mathrm{u}}_1^{(i)} + \Delta u$
  Change ith element: $\overline{\mathrm{u}}_2^{(i)} = \overline{\mathrm{u}}_2^{(i)} - \Delta u$
  Simulation $i.1$**:** apply $\overline{\mathrm{u}}_1$, store $(\overline{\mathrm{y}}_1, \overline{\mathrm{c}}_1) = \mathcal{D}(\overline{\mathrm{u}}_1)$
  Simulation $i.2$**:** apply $\overline{\mathrm{u}}_2$, store $(\overline{\mathrm{y}}_2, \overline{\mathrm{c}}_2) = \mathcal{D}(\overline{\mathrm{u}}_2)$
  Compute ith column of F and L:

$$\begin{aligned} F(:, i) &= (\overline{\mathrm{y}}_2 - \overline{\mathrm{y}}_1)/(2\Delta u) \\ L(:, i) &= (\overline{\mathrm{c}}_2 - \overline{\mathrm{c}}_1)/(2\Delta u) \end{aligned} \tag{13}$$

end for
return Lifted-domain mapping matrices F and L.

IV. EXPERIMENTAL SETUP

The iterative learning algorithm is applied to quadrocopter vehicles with the objective of precisely tracking trajectories in the three-dimensional space. The quadrocopters are operated in the ETH Flying Machine Arena (FMA), a dedicated testbed for motion control research. Similar to [16], [17], a motion capture system is used that provides precise position and attitude measurements of the vehicles. The localization data is sent to a PC that runs the control algorithms (including the iterative learning algorithm) and

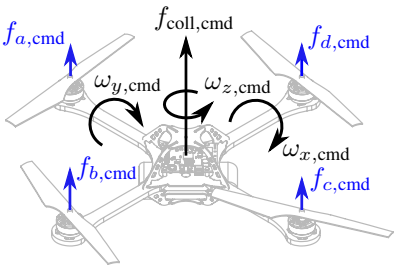

Fig. 2. The control inputs of the quadrocopter are the body rates $\omega_{x,\text{cmd}}$, $\omega_{y,\text{cmd}}$, and $\omega_{z,\text{cmd}}$ and the collective thrust $f_{\text{coll,cmd}}$. These inputs are converted by an onboard controller into motor forces $f_{i,\text{cmd}},\ i \in \{a,b,c,d\}$.

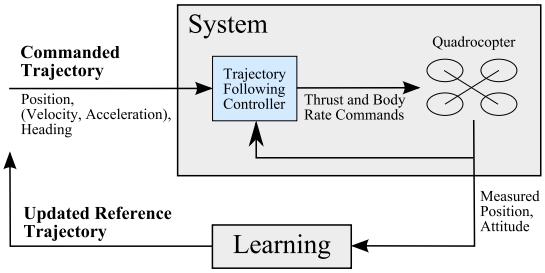

Fig. 3. We build the iterative learning scheme around a quadrocopter that is controlled by a trajectory-following controller.

sends commands back to the quadrocopters. More details about the test environment can be found in [18] and on the FMA web page, cf. [5].

A. Quadrocopter Control

The quadrotor vehicles are controlled by an onboard controller (OBC) and by a trajectory-following controller (TFC) that runs off board. The OBC accepts four inputs, the commanded collective thrust $f_{\text{coll,cmd}}$ and rotational body rates $(\omega_{x,\text{cmd}}, \omega_{y,\text{cmd}}, \omega_{z,\text{cmd}})$, and computes desired motor forces $f_{i,\text{cmd}}\ i \in \{a,b,c,d\}$ using feedback from rate gyros, cf. Fig. 2. The thrust and body rate commands are provided by the TFC, which takes desired position and yaw angle (and, optionally, corresponding derivatives) as an input, cf. Fig. 3. The TFC closes the loop using position and attitude measurements provided by the motion capture camera system. Refer to [19] for a detailed description of the TFC design.

The TFC is commonly used for trajectory tracking in the FMA. Applications include standard routines such as take-off and landing, as well as research projects like the 'Music in Motion' project, which builds upon the TFC and creates multi-vehicle performances that are designed and synchronized to music (see [5] for more information). We typically operate the TFC in two different modes: (C1) using desired quadrocopter position and yaw angle as inputs (later called *no feed-forward*), and (C2) additionally feeding velocity and acceleration values as inputs (called *with feed-forward*).

B. Applying ILC to Quadrocopters

The iterative learning scheme of Sec. II is applied to the quadrocopter system on the level of position and yaw angle input. That is, the input and output in the framework of (1) are

$$\bar{\mathrm{u}} = [x_{\text{cmd}}, y_{\text{cmd}}, z_{\text{cmd}}, \psi_{\text{cmd}}]^T \quad (14)$$

$$\bar{\mathrm{y}} = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi]^T, \quad (15)$$

where the quadrocopter's position and velocity are denoted by (x,y,z) and $(\dot{x},\dot{y},\dot{z})$, respectively, and (ϕ,θ,ψ) are the vehicle's roll, pitch and yaw angle.

The vehicles are subject to several constraints that result from both limited actuator action and limited range of sensor measurements. First, the thrust that each motor can provide is limited by $f_{\min} \le f_i \le f_{\max},\ i \in \{a,b,c,d\}$. Second, due to the motor dynamics, the rate of change of the thrust is also limited: $\left|\dot{f}_i\right| \le \dot{f}_{\max},\ i \in \{a,b,c,d\}$. Furthermore, the onboard rate gyroscopes have a limited measurement range, $|\omega_x|, |\omega_y|, |\omega_z| \le \Omega_{\max}$. The constrained quantities are summarized by $\mathrm{c}(k) = (f(k), \Delta f(k), \Omega(k))$ with $f(k) = (f_a(k), \ldots, f_d(k))$, $\Delta f = (\Delta f_a(k), \ldots, \Delta f_d(k))$ and $\Omega(k) = (\omega_x(k), \omega_y(k), \omega_z(k))$, where Δf_i represents a numerical approximation of the time derivative of f_i. The constraints are taken explicitly into account in the optimization problem (11).

C. Quadrocopter Dynamics

The simulated quadrocopter dynamics that are at the heart of the identification routine (cf. Sec. III) neglect all aerodynamic effects, motor dynamics or battery behavior, and aim to reproduce the fundamental dynamic effects only.

The continuous-time equations governing the quadrocopter's dynamics (first-principles model) are described in [19], and the dynamic behavior of $\Omega(t) = (\omega_x(t), \omega_y(t), \omega_z(t))$ is modeled as a first-order system. Given Ω and $\dot{\Omega}$ at each time step, the motor forces (f_a, f_b, f_c, f_d) are obtained from the rotational quadrocopter dynamics by solving a linear system of equations, see e.g. [18]. Parameters such as the vehicle mass are found in [2].

V. RESULTS

This section presents experimental results of the proposed learning scheme applied to quadrocopters. We focus on a particular S-shaped trajectory, for which we show that our learning algorithm significantly improves the tracking performance. While demonstrating the key characteristics of the approach for a single trajectory, the learning scheme has proven to be equally effective for arbitrary 3D trajectories, see video at `http://tiny.cc/SlalomLearning` where various slalom trajectories are learned.

A. Performance of Trajectory-Following Controller (TFC)

We first use the standard TFC described in Sec. IV-A to track the desired S-shaped trajectory (see dashed line in Fig. 4). The resulting tracking performance of the TFC is depicted in Fig. 4 for both control modes, (C1) and (C2). The input to the TFC is the desired trajectory itself (and corresponding derivatives). The TFC's tracking performance is unsatisfactory for both modes. When repeatedly executing the trajectory, we observe a large repetitive tracking error

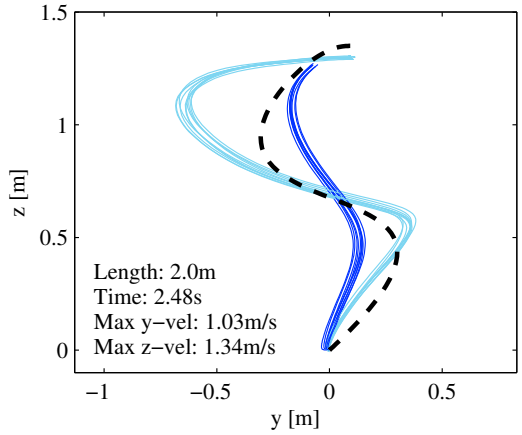

Fig. 4. Tracking an S-shaped trajectory with the trajectory-following controller (TFC). The quadrocopter position in the yz-plane is depicted for two different control modes: (C1) the TFC uses the desired position and yaw angle as inputs (dark blue), and (C2) the TFC uses additional velocity and acceleration feed-forward terms as inputs (light blue). The dashed black line shows the desired trajectory. A repeated operation shows that a large proportion of the tracking error is repetitive and only small non-repetitive effects are visible.

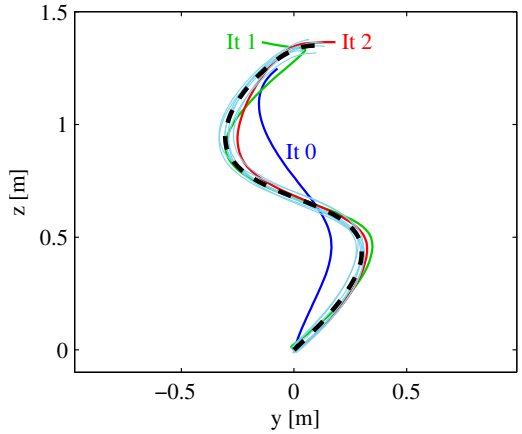

Fig. 5. Learning an S-shaped trajectory. The quadrocopter position in the yz-plane is depicted for different iterations. The dashed black line shows the desired trajectory. The trajectories of iterations 0–2 are drawn in different colors, iterations 3–9 are shown in light blue color.

component and small variations between subsequent executions due to non-repetitive disturbances. Consequently, the overall system behavior, including quadrocopter and TFC, is highly repetitive: repeated executions with the same input result in almost identical output trajectories. Iterative learning schemes are well-suited for such systems, since they are able to learn how to compensate for repetitive error components and thus are able to significantly improve the tracking accuracy. Note that we chose a particularly aggressive S-shaped motion (see velocity values in Fig. 4) to highlight the limitations of pure feedback control.

B. Tracking Performance Using Iterative Learning

Fig. 5 shows the output trajectories when applying the proposed iterative learning algorithm. After two learning steps, the quadrocopter follows the desired trajectory closely.

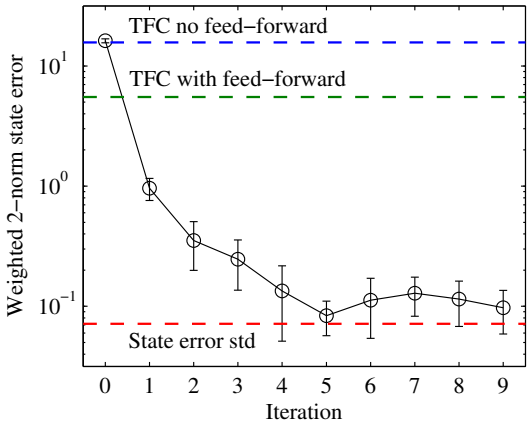

Fig. 6. Error convergence for the S-shaped trajectory. The error is computed according to (16). Ten independent learning experiments were performed. The circles and bars show the average error and standard deviation, respectively. The dashed blue and green lines show the average tracking error of the trajectory-following controller (pure feedback control) with and without feed-forward terms. The dashed red line represents the standard deviation of the tracking error when applying the same input repeatedly. It can be viewed as a measure of the noise level in the experimental setup.

The tracking error convergence is depicted in Fig. 6, where we consider the weighted error

$$e_{w,j} := \left\| S\left(\check{\mathrm{y}} + \mathrm{y}_j\right) \right\|_2 \tag{16}$$

as a performance measure reflecting the learning objective in (11). Starting from an initial performance determined by the TFC performance, the learning scheme successfully compensates for repetitive errors along the trajectory and, in 5 to 6 iterations, reduces the tracking error (16) to values in the range of the stochastic (i.e. non-repetitive) noise level. The dashed red line in Fig. 6 depicts the standard deviation of the tracking error when applying the same input to the system and observing the variations of the performed trajectories. It can be viewed as a measure of the noise level in the system and represents a lower bound of the achievable tracking accuracy. For an intuitive interpretation, the ***average*** position tracking error along the trajectory after successful learning is in the range of 2–3 cm. This is also the accuracy that we achieve when hovering the vehicle at a given point.

As discussed in Sec. II, the learning scheme iteratively updates the reference input signal sent to the system. Fig. 7 depicts the y-position input signals of the initial trial and of iteration 7–9. The input trajectories converge over iterations until their variability is in the range of the stochastic noise level. The final input trajectories, together with the estimated disturbance vector, comprise the knowledge that we gain from the iterative learning process. Fig. 7 shows that the learned reference signal commands the left/right motion sooner (compensating for the delay in the system) and with larger amplitudes (counteracting the system's inherent attenuation).

VI. COMPUTATIONAL COMPLEXITY

Identification Routine. The number of simulations required to identify the mappings (4) is linear in N and n_u.

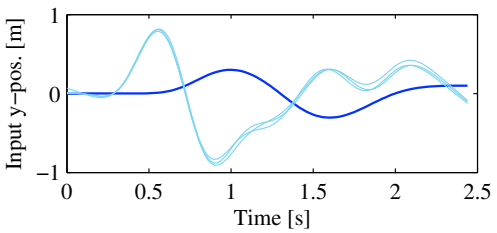

Fig. 7. Learned y-position inputs for the S-shaped trajectory. The initial input corresponds to the desired output trajectory and is drawn in dark blue. The converged learning inputs (iterations 7–9) are shown in light blue color.

According to Algorithm 1, we need exactly $2Nn_{\mathrm{u}}$ simulations. Obviously, the computation time of a single simulation depends on the complexity and implementation of (1). The identification of the trajectory of Sec. V takes 93 s on a standard desktop PC (Windows 7, 64-bit; quad-core @ 2.8 GHz, 4 GB RAM).

Learning Algorithm. A detailed complexity analysis of the learning algorithm can be found in [2]. The biggest computational cost is associated with the solution of (11), which takes 30 s in the example presented.

VII. ADVANTAGES & LIMITATIONS

The experimental results presented in the previous section showed that the learning algorithm significantly improves the system's tracking performance. We achieve a high tracking accuracy due to the fact that the learning scheme embraces the quadrocopter including underlying feedback, cf. Fig. 3. This setup results in a highly repetitive system, where the feedback controller compensates for most of the non-repetitive noise. The remaining non-repetitive noise acting on the overall closed-loop system defines a lower bound on the achievable tracking accuracy. Compared to our previous work [2], this bound has been significantly reduced with this setup.

The tracking error convergence speed depends on the prediction quality of the mapping (6), which is based on the identified system dynamics, cf. Sec. III. However, experimental results suggest that the learning algorithm is very robust to inaccurate mappings. Moreover, after a few executions of the learning step, the system may be *re*-identified around $(\overline{\mathrm{u}}_j, \overline{\mathrm{x}}_j, \overline{\mathrm{y}}_j)$ for some $j > 0$ in order to obtain a more accurate mapping (4). Further, an inaccurate constraint mapping $\mathrm{c} = L\,\mathrm{u}$, cf. (4), may corrupt the learning by either allowing infeasible values or being too restrictive. This can be overcome by estimating an additive correction vector $\mathrm{d_c}$ similar to d in (6); that is, $\mathrm{c} = L\,\mathrm{u} + \mathrm{d_c}$.

VIII. CONCLUSION

This paper provided a conceptually simple and computationally efficient learning framework for trajectory tracking. The approach is a generalization of our previous work. No analytical system model is required; instead, the algorithm is applicable to any dynamic system, for which a numerical dynamics simulation is available. Because of the acausal learning action that corrects for repetitive disturbances *before* they occur, the final tracking performance of the proposed learning scheme outperforms pure feedback control. By basing the learning approach on a coarse dynamics model of the system, we achieve fast convergence, usually in around 5 iterations. The novelty of this paper is that the dynamics model was derived from a numerical simulation, and that this identification routine, combined with the learning algorithm, were experimentally evaluated on quadrotor vehicles guided by an underlying trajectory-following controller.

REFERENCES

[1] A. P. Schoellig and R. D'Andrea, "Optimization-based iterative learning control for trajectory tracking," in *Proceedings of the European Control Conference (ECC)*, 2009, pp. 1505–1510.

[2] A. P. Schoellig, F. L. Mueller, and R. D'Andrea, "Optimization-based iterative learning for precise quadrocopter trajectory tracking," *Autonomous Robots*, vol. 33, pp. 103–127, 2012.

[3] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the American Control Conference (ACC)*, vol. 1, 2005, pp. 300–306.

[4] M. Kawato, "Feedback-error-learning neural network for supervised motor learning," *Advanced Neural Computers*, vol. 6, no. 3, pp. 365–372, 1990.

[5] "The Flying Machine Arena," www.flyingmachinearena.org, last accessed March 07, 2012.

[6] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering operation of robots by learning," *Journal of Robotic Systems*, vol. 1, no. 2, pp. 123–140, 1984.

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

[8] H.-S. Ahn, K. L. Moore, and Y. Chen, *Iterative Learning Control: Robustness and Monotonic Convergence for Interval Systems (Communications and Control Engineering)*, 1st ed. Springer, 2007.

[9] I. Chin, S. J. Qin, K. S. Lee, and M. Cho, "A two-stage iterative learning control technique combined with real-time feedback for independent disturbance rejection," *Automatica*, vol. 40, no. 11, pp. 1913–1922, 2004.

[10] M. Cho, Y. Lee, S. Joo, and K. S. Lee, "Semi-empirical model-based multivariable iterative learning control of an RTP system," *IEEE Transactions on Semiconductor Manufacturing*, vol. 18, no. 3, pp. 430–439, 2005.

[11] K. Barton, S. Mishra, and E. Xargay, "Robust iterative learning control: L1 adaptive feedback control in an ILC framework," in *Proceedings of the American Control Conference (ACC)*, 2011, pp. 3663–3668.

[12] O. Purwin and R. D'Andrea, "Performing aggressive maneuvers using iterative learning control," in *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 1731–1736.

[13] B. Bamieh, J. B. Pearson, B. A. Francis, and A. Tannenbaum, "A lifting technique for linear periodic systems with applications to sampled-data control," *Systems & Control Letters*, vol. 17, no. 2, pp. 79–88, 1991.

[14] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[15] "IBM ILOG CPLEX Optimizer," http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, last accessed February 08, 2012.

[16] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Systems Magazine*, vol. 28, no. 2, pp. 51–64, 2008.

[17] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The GRASP multiple micro UAV testbed," *IEEE Robotics and Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.

[18] S. Lupashin and R. D'Andrea, "Adaptive fast open-loop maneuvers for quadrocopters," *Autonomous Robots*, vol. 33, pp. 89–102, 2012.

[19] A. P. Schoellig, C. Wiltsche, and R. D'Andrea, "Feed-forward parameter identification for precise periodic quadrocopter motions," in *Proceedings of the American Control Conference (ACC)*, 2012, pp. 4313–4318.