Optimization-based iterative learning for precise quadrocopter trajectory tracking

Auton Robot (2012) 33:103–127
DOI 10.1007/s10514-012-9283-2

Optimization-based iterative learning for precise quadrocopter trajectory tracking

Angela P. Schoellig · Fabian L. Mueller · Raffaello D'Andrea

Received: 1 August 2011 / Accepted: 10 February 2012 / Published online: 21 April 2012

Abstract Current control systems regulate the behavior of dynamic systems by reacting to noise and unexpected disturbances as they occur. To improve the performance of such control systems, experience from iterative executions can be used to anticipate recurring disturbances and proactively compensate for them. This paper presents an algorithm that exploits data from previous repetitions in order to learn to precisely follow a predefined trajectory. We adapt the feed-forward input signal to the system with the goal of achieving high tracking performance—even under the presence of model errors and other recurring disturbances. The approach is based on a dynamics model that captures the essential features of the system and that explicitly takes system input and state constraints into account. We combine traditional optimal filtering methods with state-of-the-art optimization techniques in order to obtain an effective and computationally efficient learning strategy that updates the feed-forward input signal according to a customizable learning objective. It is possible to define a termination condition that stops an execution early if the deviation from the nominal trajectory exceeds a given bound. This allows for a safe learning that gradually extends the time horizon of the trajectory. We developed a framework for generating arbitrary flight trajectories and for applying the algorithm to highly maneuverable autonomous quadrotor vehicles in the ETH Flying Machine Arena testbed. Experimental results are discussed for selected trajectories and different learning algorithm parameters.

Keywords Motion planning and control · Aerial robotics · Quadrocopter control · Trajectory tracking · Learning · Input and state constraints · Kalman filtering · Convex optimization

Electronic supplementary material The online version of this article (doi:10.1007/s10514-012-9283-2) contains supplementary material, which is available to authorized users.

A.P. Schoellig (✉) · F.L. Mueller · R. D'Andrea
Institute for Dynamic Systems and Control (IDSC), ETH Zurich, Zurich, Switzerland
e-mail: aschoellig@ethz.ch

F.L. Mueller
e-mail: famuelle@ethz.ch

R. D'Andrea
e-mail: rdandrea@ethz.ch

1 Introduction

1.1 Goal & motivation

Over the last century, controls of dynamic systems have expanded from mechanical and analog-electronic controllers for limited subproblems (such as course stabilization of ships) to digital control of fully autonomous systems (such as unmanned aerial vehicles). This trend has been enabled by technical advancements in sensors, actuators, and digital computing components, as well as by significant developments in the theoretical foundations of control. Like their early counterparts, current control systems usually regulate the behavior of dynamic systems by reacting to noise and unexpected disturbances in the measured system output. 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 in regimes where feedback is not able to react in time and the dynamic behavior is difficult to identify or understand. To achieve high performance in

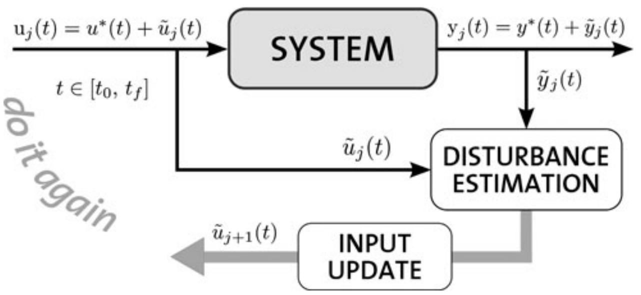

Fig. 1 The general iterative learning framework considered in this paper: A complete trial $u_j(t)$, $t \in [t_0, t_f]$ is performed. Based on the output error $\tilde{y}_j(t)$, a new input $u_{j+1}(t)$ is calculated and applied during the next trial

such cases, we propose data-based control approaches that are able to store and interpret information from past executions, and infer the correct actions or control laws for future experiments.

The objective of this paper is to explore the field of data-based high performance control by developing algorithms that enable systems to precisely track predefined trajectories. We propose an iterative learning scheme with the goal of achieving high tracking performance—even under the presence of model errors, parameter uncertainties, and other recurring disturbances. Figure 1 depicts the general idea of the algorithm. Data is collected through a repeated execution of the same task and the performance is improved from trial to trial by identifying recurring disturbances and adapting the feed-forward input signal accordingly. We leverage a model of the system's key dynamics to increase the efficiency of the learning and the speed of convergence. Thus, our approach applies to any underlying dynamic system for which a nominal model is available. Since the correcting action is executed only after a complete run of the trajectory, the approach is not restricted by slow feedback rates or large system latencies. Furthermore, it is not limited to a causal action, which reacts only to disturbances after they occurred. Instead, recurring disturbances (mainly due to modeling errors) are anticipated and proactively compensated for before they occur. This approach is thus suitable for performing aggressive trajectories.

We apply the algorithm to quadrotor vehicles in the ETH Flying Machine Arena. Quadrotor vehicles offer exceptional agility in the rotational and translational degrees of freedom due to the large torques generated by the off-center mounted propellers and the high thrust-to-weight ratio. When operating these vehicles at high speeds, complex dynamic effects such as aerodynamics, battery behavior, and motor dynamics have a significant impact on the vehicle behavior. These effects are difficult to model but can be compensated for by an iterative execution.

Due to recent technological advances in aerial robotics, interest in using micro aerial vehicles for industrial applications, including exploration and surveillance, inspection and monitoring, and transportation and entertainment has grown. As such, precise trajectory tracking will become relevant for operations where the tracking performance of the system determines the quality of the experimental result. Iterative learning will be applicable if repetition is inherent to the required task. Examples include inspection of civil infrastructure (such as bridges, highways and dams), environmental monitoring (of forest, rivers, lakes, etc.) or filming a scene with a camera mounted on a robot. Common to all these examples is that 'measurements' must be taken along pre-computed paths.

We have developed a framework for the generation and iterative learning of flight trajectories for quadrocopters. Feasible flight trajectories are generated based on user input that defines the shape of the desired flight path in the vertical plane. We restrict ourselves to two-dimensional trajectories for the sake of simplicity (though all derivations generalize to 3D trajectories). Trajectory feasibility is considered with respect to the corresponding first-principles model of the vehicle and sensor/actuator constraints. Based on the same model, the proposed learning scheme is derived, which iteratively improves the trajectory tracking performance by adapting the feed-forward input signal. This adaptation does not change the underlying dynamics of the vehicle, and thus has an advantage over stiff and switching controllers, which may cause non-smooth motions and undesirable transients (especially in cases where the underlying model of the system is inaccurate).

1.2 Related work

Research in aggressive flying and trajectory tracking of autonomous quadrocopters has made considerable progress over the last decade, and strategies for both generating reference trajectories and for designing effective and robust control algorithms have improved. Since quadrotor vehicles are inherently unstable nonlinear systems, and because they exhibit exceedingly complex behavior at high speeds, one approach is to extend classical control methods with sophisticated adaptation and learning schemes designed to cope with the complicated system dynamics, unavoidable model uncertainties and external disturbances.

Examples of classical control approaches used to track trajectories with quadrocopters are PID schemes (Zhou et al. 2010a), backstepping control techniques (Bouabdallah and Siegwart 2005; Mokhtari and Benallegue 2004; Zuo 2010; Madani and Benallegue 2006; Lee et al. 2009; Raffo et al. 2008) and feedback linearization (Zhou et al. 2010b; Al-Hiddabi 2009). Other common strategies are trajectory linearization control (Zhu and Huo 2010), constrained finite-time optimal control (Alexis et al. 2010), LQ optimal solutions (Bauer et al. 2009) or Model Predictive Control (Castillo et al. 2007). Such control schemes fall in the area of causal controllers. Applying one of these approaches and performing the same trajectory over and over again, results in the same tracking error in each trial (on average). These schemes are not designed to exploit past experience in order to improve future performances.

Other research on the control of flying vehicles has focused on learning schemes. In Waslander et al. (2005), where altitude control of quadrocopters is studied, integral sliding mode control and reinforcement learning techniques are compared. Neural networks and output feedback are used in Dierks and Jagannathan (2010) to learn the complete dynamics of the vehicle online. Work in Nicol et al. (2011) and Diao et al. (2011) utilizes adaptive control to achieve both, adaption to unknown payloads and robustness to disturbances. While most of these approaches are concerned with near-hover operations, the goal of our approach is to track fast trajectories. As shown in Sect. 4, we generate reference trajectories that minimize execution time.

Other learning strategies have been developed for fast quadrocopter aerobatics. A method to learn high-speed quadrocopter multi-flips was introduced in Lupashin et al. (2010), which uses a policy gradient method to iteratively learn parameterized flip primitives. Similar approaches are used in Mellinger et al. (2010), Lupashin and D'Andrea (2011), Ritz et al. (2011) to achieve various high-speed, high-performance maneuvers. When comparing those approaches to the approach presented in this paper, the main difference is the level of specificity of the desired reference trajectory. While we aim to follow a continuous trajectory, work in Lupashin et al. (2010), Mellinger et al. (2010), Lupashin and D'Andrea (2011), Ritz et al. (2011) compares the actual and desired states only at specific key frames.

The approach presented in this paper can be characterized as an iterative learning control (ILC) technique and is based on our previous work in Schoellig and D'Andrea (2009). ILC became a popular research topic beginning with Arimoto et al. (1984), and has since proven to be a very powerful method for high performance reference tracking (a recent overview of ILC with an extensive bibliography is available in Bristow et al. 2006 and Ahn et al. 2007). Yet methods from optimal control theory have only recently been applied to the design of ILC laws. Based on a so-called 'lifted' domain representation, cf. Phan and Longman (1988), Moore (1998), Amann et al. (1996), LQG-type solutions have been proposed by Lee et al. (2001), Cho et al. (2005), Tousain et al. (2001), Rice and Verhaegen (2010), Ahn et al. (2007) for estimating the tracking error and minimizing a quadratic cost function. Work in Bristow et al. (2006), Chin et al. (2004), Cho et al. (2005), Barton et al. (2011) has shown that ILC can be applied to systems with underlying feedback loops. The real-time feedback component is intended to reject non-repetitive noise while the ILC adjusts to the repetitive disturbance.

In practice ILC has been applied to repetitive tasks performed by stationary systems, such as wafer stages, chemical reactors, and industrial robots. Applications to autonomous vehicles are more rare. There is one example where ILC was applied to quadrocopters: Purwin and D'Andrea (2009) introduce a least-squares based learning rule to improve on horizontal point-to-point motions. The work presented in this paper can be viewed as an extension of the results in Purwin and D'Andrea (2009), and addresses the issues referred to as 'open questions' in Purwin and D'Andrea (2009). We consider a larger class of motions—namely, arbitrary trajectories in the vertical plane—and a generalized two-step learning framework, which is discussed in more detail below. Input and state constraints of the system are explicitly incorporated in our learning rule.

1.3 Contribution to the field of feed-forward based trajectory learning

The contribution of this paper to the field of feed-forward based trajectory learning is twofold:

First, we develop an algorithm that structures the trajectory learning problem around a disturbance estimation and input update step, see Fig. 1. Both steps rely on a nominal model of the underlying system. Estimation and input update are clearly separated, which allows for a flexible combination of different approaches for both steps. More important from a practical point of view, the clear separation allows for an intuitive tuning of the overall learning scheme. In the first step, we design a time-varying Kalman filter, which estimates the model error along the trajectory. The estimated error serves as the input to the following control step. The Kalman filter explicitly takes noise characteristics into account, which can be adjusted to improve the convergence of the estimation and, thus, of the overall learning. In the second step, the control objective is formulated as a convex optimization problem (Boyd and Vandenberghe 2004). Here, in contrast to least-squares approaches or LQG design (Lee et al. 2001; Cho et al. 2005; Tousain et al. 2001; Rice and Verhaegen 2010; Ahn et al. 2007), input and state constraints can be explicitly incorporated. Different (nonlinear) performance objectives can be defined by choosing appropriate vector norms and adequate scaling and weighting of the error vector. Moreover, derivatives of the input can be included into the objective function to reduce jittering in the control inputs and, consequently, improve the robustness of the learning. The definition of a termination condition justifying the linearization of the system dynamics is unique in the area of ILC and results in a learning process that better meets the need for safe and efficient operation.

Second, the derived learning scheme is thoroughly applied to quadrocopters to achieve fast and accurate trajectory tracking. The paper presents an entire unified process, including the generation of feasible reference trajectories, the application to real quadrotor vehicles, and a detailed experimental study characterizing both the influence of different learning parameter settings as well as features of the experimental setup and the quadrotor vehicles.

The theoretic approach, as well as the application of the algorithm to real quadrotor vehicles, makes this work an original contribution to the field.

1.4 Outline

The paper is organized as follows:

In Sect. 2, we present the iterative learning algorithm in its general form. The learning scheme is introduced as a two-step process of first estimating the unknown repetitive disturbance (Sect. 2.2) and later compensating for it (Sect. 2.3). The approach is based on a dynamic model of the system (Sect. 2.1) and includes the unique feature of gradually increasing the trial horizon (Sect. 2.4).

In the second part of the paper, Sects. 3–7, we apply the algorithm to quadrotor vehicles, and develop a complete framework tailored towards generating and learning arbitrary flight trajectories in the vertical plane. A model of the quadrocopter dynamics and constraints is derived first, in Sect. 3, and a method for generating feasible reference trajectories is presented in Sect. 4. The experimental setup and implementation details are illustrated in Sect. 5. Finally, Sect. 6 presents the quadrocopter's learning behavior in actual experiments.[1] We conclude with a discussion on the limitations of the proposed approach in Sect. 7 and summarize the presented results in Sect. 8.

2 The learning algorithm

The basic idea of the proposed learning scheme is to use iterative experiments to teach a dynamic system how to precisely follow a trajectory. By exploiting the experience gained from previous trials, the system learns to anticipate recurring disturbances (that are mainly due to modeling errors) and to compensate for them in a non-causal way. We execute a learning update after each trial by combining *a priori* knowledge about the system's dominating dynamics with real measurements from experiments.

The basic procedure is depicted in Fig. 1 and is described as follows: We assume that we can derive a model that captures the key dynamics of the underlying system. This model is used to calculate the nominal input and state trajectories. Moreover, by linearizing the system about the nominal trajectory and discretizing the resulting equations, we can derive a static map that describes the system dynamics during one trial (Sect. 2.1). The learning algorithm builds upon this lifted model when interpreting the data of one trial and updating the feed-forward input signal for the next trial. These two steps are clearly separated. We use a Kalman filter to interpret the measurement of the last trial and to incorporate the measurement into the current estimate of the disturbance (Sect. 2.2). The input update step takes the current disturbance estimate and returns a more adequate input for the next trial by solving a constrained optimization problem (Sect. 2.3). The input serves as a feed-forward reference signal to the underlying system. After each iteration the system is reset to the initial state. Safe and gradual learning is ensured by including a predefined termination condition that stops a trial whenever the actual trajectory diverges from the desired one by an unacceptable amount (Sect. 2.4).

Key concepts including the system representation, disturbance estimation, input update, and extending horizon learning are presented first. The main steps of the algorithm from an application perspective are summarized in Sect. 2.5, where we also highlight important prerequisites of the approach. Finally, in Sect. 2.6 we analyze the computational complexity of the approach.

2.1 Model of dynamics and lifted-domain representation

We assume that we can derive a model that captures the key dynamics of the physical system under consideration. In the general case, the system dynamics are modeled by a set of time-varying nonlinear differential equations,

$$\begin{aligned} \dot{\mathrm{x}}(t) &= \mathrm{f}\big(\mathrm{x}(t), \mathrm{u}(t), t\big), \\ \mathrm{y}(t) &= \mathrm{g}\big(\mathrm{x}(t), t\big), \end{aligned} \tag{1}$$

where $\mathrm{u}(t) \in \mathbb{R}^{n_\mathrm{u}}$ denotes the system input, $\mathrm{x}(t) \in \mathbb{R}^{n_\mathrm{x}}$ the system state, and $\mathrm{y}(t) \in \mathbb{R}^{n_\mathrm{y}}$ the output. The vector fields f and g are assumed to be continuously differentiable in x and u. Constraints on the state $\mathrm{x}(t)$, the input $\mathrm{u}(t)$, and respective time derivatives are represented by

$$Z\mathrm{q}(t) \le \mathrm{q}_{\max}, \tag{2}$$

where

$$\mathrm{q}(t) = \left[\mathrm{x}(t),\ \mathrm{u}(t),\ \dot{\mathrm{x}}(t),\ \dot{\mathrm{u}}(t),\ \ldots,\ \frac{d^m}{dt^m}\mathrm{x}(t),\ \frac{d^m}{dt^m}\mathrm{u}(t)\right] \tag{3}$$

and $\mathrm{q}_{\max} \in \mathbb{R}^{n_\mathrm{q}}$. The inequality is defined component-wise where n_q is the total number of constraints, and Z is a constant matrix of appropriate dimensions. Equation (2) allows the incorporation of constraints on any linear combination of $\mathrm{x}(t)$, $\mathrm{u}(t)$, and their time derivatives, for example,

$$\mathrm{x}(t) \le \mathrm{q}_{x,\max} \quad \text{and} \quad a\mathrm{u}(t) + b\dot{\mathrm{u}}(t) \le \mathrm{q}_{u,\max}, \quad a, b \in \mathbb{R}. \tag{4}$$

[1] The accompanying video is found at http://tiny.cc/QuadroLearnsTrajectory.

The goal of our learning algorithm is to track an *a priori* determined output trajectory $y^*(t)$ over a finite-time interval $t \in \mathcal{T} = [t_0, t_f]$, $t_f < \infty$. We assume that the desired trajectory $y^*(t)$, $t \in \mathcal{T}$, is feasible with respect to the nominal model (1), (2). That is, there exists a triple

$$\big(u^*(t), x^*(t), y^*(t)\big), \quad t \in \mathcal{T}, \tag{5}$$

satisfying (1) and (2). For some applications, the desired output trajectory $y^*(t)$ may be known ahead of time. However, it may also be the result of an optimization problem as shown in Sect. 4.

Below, we derive a system representation of (1)–(5) that facilitates the derivation and implementation of the learning algorithm. First we assume that the motion of the system stays close to the generated reference trajectory (5) during the learning process. (Note that this can be enforced by the extending horizon feature introduced in Sect. 2.4.) We linearize the dynamics around the reference trajectory. Considering only small deviations $(\tilde{u}(t), \tilde{x}(t), \tilde{y}(t))$ from the desired trajectory (5),

$$\begin{aligned} &\tilde{u}(t) = \mathrm{u}(t) - u^*(t), \qquad \tilde{x}(t) = \mathrm{x}(t) - x^*(t), \\ &\tilde{y}(t) = \mathrm{y}(t) - y^*(t), \end{aligned} \tag{6}$$

the system's behavior (1) can be approximated by a first-order Taylor series expansion about the reference trajectory (5) (cf. Lee et al. 2001) resulting in the following linear, time-varying system

$$\begin{aligned} &\dot{\tilde{x}}(t) = A(t)\tilde{x}(t) + B(t)\tilde{u}(t), \\ &\tilde{y}(t) = C(t)\tilde{x}(t), \quad t \in \mathcal{T}, \end{aligned} \tag{7}$$

where the time-dependent matrices $A(t)$, $B(t)$, $C(t)$ are the corresponding Jacobian matrices of the nonlinear functions f and g with respect to x and u. The input-output relationship as given by (7) is fundamental to the model-based learning scheme proposed subsequently. On the real system, however, inputs are sent at discrete times, and measurements are available only at fixed time intervals. To capture this fact, we derive a discrete-time representation of the plant dynamics (7), cf. Ahn et al. (2007), Bristow et al. (2006), Chen and Wen (1999), and references therein. Converting (7) to a discrete-time system results in the following linear, time-varying difference equations,

$$\begin{aligned} &\tilde{x}(k+1) = A_D(k)\tilde{x}(k) + B_D(k)\tilde{u}(k), \\ &\tilde{y}(k+1) = C_D(k+1)\tilde{x}(k+1), \end{aligned} \tag{8}$$

where $k \in \mathcal{K} = \{0, 1, \dots, N-1\}$, $N < \infty$, denotes the discrete-time index and N is the trial length in discrete-time steps, $N = t_f/\Delta t$ with Δt being the sampling time and t_f assumed to be a multiple of Δt. That is, the desired trajectory (5) is represented by an N-sample sequence

$$\big(u^*(k), x^*(k+1), y^*(k+1)\big), \quad k \in \mathcal{K}, \tag{9}$$

with given initial state $x^*(0)$.

Other associated signals, e.g. (6), are discretized analogously. The constraints (2) are similarly transformed,

$$Z\tilde{q}(t) \le \mathrm{q}_{\max} - Zq^*(t) := \mathrm{q}_{\max}(t) \tag{10}$$

where $\tilde{q}(t)$ is the deviation of $\mathrm{q}(t)$ from the corresponding nominal values $q^*(t)$ defined analogously to (6). Discretizing the above equation results in

$$Z\tilde{q}(k) \le q_{\max}(k), \tag{11}$$

where the derivative components in $\tilde{q}(k)$ are replaced by a discrete approximation. For example, $\Delta\tilde{u}(k) = (\tilde{u}(k) - \tilde{u}(k-1))/\Delta t$ may be used as an approximation for the input derivative. The values of the vector $q_{\max}(k) \in \mathbb{R}^{n_\mathrm{q}}$ depend on the discretization method.

Introducing the lifted vector representation, cf. Bamieh et al. (1991),

$$\begin{aligned} &u = \big[\tilde{u}(0), \tilde{u}(1), \dots, \tilde{u}(N-1)\big]^T \in \mathbb{R}^{Nn_\mathrm{u}}, \\ &x = \big[\tilde{x}(1), \dots, \tilde{x}(N)\big]^T \in \mathbb{R}^{Nn_\mathrm{x}}, \\ &y = \big[\tilde{y}(1), \dots, \tilde{y}(N)\big]^T \in \mathbb{R}^{Nn_\mathrm{y}}, \end{aligned} \tag{12}$$

the dynamics (8) of a complete trial are captured by a static mapping

$$\begin{aligned} &x = Fu + d^0, \\ &y = Gx, \end{aligned} \tag{13}$$

where the lifted matrix $F \in \mathbb{R}^{Nn_\mathrm{x} \times Nn_\mathrm{u}}$ is composed of the matrices $F_{(l,m)} \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{u}}$, $1 \le l, m \le N$,

$$F = \begin{bmatrix} F_{(1,1)} & \cdots & F_{(1,N)} \\ \vdots & \ddots & \vdots \\ F_{(N,1)} & \cdots & F_{(N,N)} \end{bmatrix}, \tag{14}$$

with

$$F_{(l,m)} = \begin{cases} A_D(l-1) \cdots A_D(m) B_D(m-1) & \text{if } m < l, \\ B_D(m-1) & \text{if } m = l, \\ 0 & \text{if } m > l. \end{cases}$$

The matrix G is block-diagonal and analogously defined by

$$G_{(l,m)} = \begin{cases} C_D(l) & \text{if } l = m, \\ 0 & \text{otherwise,} \end{cases}$$

where $G_{(l,m)} \in \mathbb{R}^{n_y \times n_x}$. Vector d^0 contains the free response of the system (8) to the initial deviation $\tilde{x}(0) = \tilde{x}_0 \in \mathbb{R}^{n_x}$,

$$d^0 = \left[\left(A_D(0)\tilde{x}_0\right)^T, \left(A_D(1)A_D(0)\tilde{x}_0\right)^T, \ldots, \left(\prod_{i=0}^{N-1} A_D(i)\,\tilde{x}_0\right)^T \right]^T.$$

This lifting technique is well-suited for the analysis and synthesis of iterative learning schemes, where the system is assumed to operate in a repetitive mode, cf. Phan and Longman (1988), Tousain et al. (2001), Moore (1998), Amann et al. (1996). The static linear mapping (13) captures the complete time-domain dynamics of a single trial by mapping the finite input time series $\tilde{u}(k)$, $k \in \mathcal{K}$, onto the corresponding output time series $\tilde{y}(k+1)$, $k \in \mathcal{K}$. The goal of the iterative learning scheme is to use data gathered during previous executions to improve the system's performance from iteration to iteration by updating the feed-forward signal u, cf. (12), after each trial. The dynamics of the learning, i.e., the dynamic behavior of a sequence of consecutive trials, can be described in the lifted domain by introducing a subscript j indicating the jth execution of the desired task, $j \in \{1, 2, \ldots\}$.

The evolution of the system over several iterations is modeled by

$$\begin{aligned} x_j &= F u_j + d_j + N_\xi \xi_j, \\ y_j &= G x_j + N_\upsilon \upsilon_j, \end{aligned} \tag{15}$$

with

$$d_j = d_{j-1} + \omega_{j-1}. \tag{16}$$

Here, j denotes the jth trial. The signals ξ_j and υ_j account for process and measurement noise, respectively. These noise signals vary from iteration to iteration and are assumed to be trial-uncorrelated sequences of zero-mean Gaussian white noise. The vector d_j can be interpreted as a repetitive disturbance component that is subject only to slight changes from iteration to iteration, cf. (16) with ω_j being another trial-uncorrelated sequence of zero-mean Gaussian white noise. The vector d_j captures model errors along the trajectory, including repeating disturbances (Norrloef and Gunnarsson 2002), and repeated nonzero initial conditions (Longman 2000), which were previously represented by d^0, cf. (13). The zero-mean noise component of the initial condition is part of the random variable ξ_j.

In the model (15)–(16), the state deviation x_j from the reference trajectory x^*, $x^* = [x^*(1), \ldots, x^*(N)]^T$, is affected by two different noise sources: a trial-uncorrelated zero-mean component ξ_j, and a 'random walk' component d_j. This versatile noise model includes the stochasticity of the process noise ξ_j and the repetitive nature of the modeling errors d_j, which can vary between trials due to the influence of ω_j, see also Rice and Verhaegen (2010), Lee et al. (2001), Chin et al. (2004). In particular, the vector d_j captures all non zero-mean noise effects along the desired trajectory x^*. It may also be interpreted as a vector representation of all unmodeled dynamics along the desired trajectory x^*. As a result, d_j may depend on the applied input $\mathrm{u}(t) = u^*(t) + \tilde{u}(t)$, $t \in \mathcal{T}$. The ultimate goal of the subsequent derivations is to estimate and optimally compensate for the disturbance d_j by updating the input trajectory appropriately.

To complete the lifted representation (15)–(16), the constraints (11) are transformed appropriately. Note that all entries in $\tilde{q}(k)$ can be expressed by linear combinations of the vectors x and u defined in (12). Introducing $q = [x, u]^T$ and stacking the bounds $q_{\max}(k)$ in a vector as

$$q_{\max} = \left[q_{\max}(0), q_{\max}(1), \ldots, q_{\max}(N)\right]^T \in \mathbb{R}^{(N+1)n_q}, \tag{17}$$

constraints (11) read as

$$Lq \leq q_{\max}, \tag{18}$$

where L is a constant matrix of appropriate dimensions.

Subsequently, the representation of the model dynamics in the lifted domain by (15), (16), and (18) allows for the derivation and the execution of operations in the trial-time domain.

2.2 Disturbance estimation

We consider our learning algorithm to be a two-step update law, cf. Fig. 1. First, we estimate the modeling error d_j along the desired trajectory using optimal filtering techniques (Anderson and Moore 2005). Then, in order to optimally compensate for the estimated vector $\widehat{d}_j$, we provide a new feed-forward input $u_{j+1} \in \mathbb{R}^{N n_u}$.

We propose an iteration-domain Kalman filter that retains all available information from previous trials (namely the output signals $y_1, y_2, \ldots, y_j$) in order to estimate the current error d_j. Combining (15) and (16), we obtain a discrete-time system that fits into the standard Kalman filter approach, cf. Chui and Chen (1998):

$$\begin{aligned} d_j &= d_{j-1} + \omega_{j-1}, \\ y_j &= G d_j + G F u_j + \mu_j, \end{aligned} \tag{19}$$

where, consistent with the previous definitions, the noise term μ_j, $\mu_j = G N_\xi \xi_j + N_\upsilon \upsilon_j$, is assumed to be zero-mean Gaussian white noise with covariance M_j: $\mu_j \sim \mathcal{N}(0, M_j)$. The noise characteristics of ω_j are given by $\omega_j \sim \mathcal{N}(0, \Omega_j)$. Both stochastic inputs, ω_j and μ_j, are trial-uncorrelated and assumed to be independent; that is, for $i, j \in \{0, 1, 2, \ldots\}$,

$$E\big[\omega_i \omega_j^T\big] = E\big[\mu_i \mu_j^T\big] = 0 \quad \text{if } i \neq j, \qquad E\big[\omega_i \mu_j^T\big] = 0 \quad \forall i, j. \tag{20}$$

$E[\cdot]$ denotes the expected value.

For the above system (19)–(20), the Kalman filter returns an unbiased disturbance estimate $\widehat{d}_j$ for $j \geq 1$ that minimizes the trace of the error covariance matrix

$$P_j = E\big[(d_j - \widehat{d}_j)(d_j - \widehat{d}_j)^T\big] \tag{21}$$

trial j taking measurements y_m, $1 \leq m \leq j$, into account.

Given initial values for $\widehat{d}_0$ and P_0, the Kalman filter update equations for the specific problem read as:

$$\begin{cases} S_j = P_{j-1} + \Omega_{j-1}, \\ K_j = S_j G^T \big(G S_j G^T + M_j\big)^{-1}, \\ P_j = (I - K_j G) S_j, \end{cases} \tag{22}$$

where $I \in \mathbb{R}^{Nn_x \times Nn_x}$ represents the identity matrix. Based on the optimal Kalman gain K_j, and taking into account the previous estimate $\widehat{d}_{j-1}$ and the actual measurement y_j, the disturbance estimate $\widehat{d}_j$ is calculated by

$$\widehat{d}_j = \widehat{d}_{j-1} + K_j\big(y_j - G\widehat{d}_{j-1} - GFu_j\big). \tag{23}$$

Note that the matrices in (22) and, especially, the Kalman gains K_j (necessary for an appropriate online update of the error estimate $\widehat{d}_j$) can be calculated prior to the experiment as long as we know the initial value P_0.

Design parameters The performance of the estimation can be adjusted by four design parameters:

– The covariance matrix Ω_j, $\omega_j \sim \mathcal{N}(0, \Omega_j)$, indicates the likely change of the disturbance d_j from iteration to iteration. The vector d_j captures the effect of unmodeled dynamics and, hence, may depend on u_j. The input u_j changes significantly during the first iterations of the learning, but converges for an increasing number of trials. This may also be true for d_j. To account for these changes, one possible definition of the covariance Ω_j is

$$\Omega_j = \epsilon_j I, \quad \epsilon_j > 0, \tag{24}$$

where the scalar ϵ_j is chosen to be larger during the first iterations to guarantee a fast initial adaptation of the disturbance estimate $\widehat{d}_j$, and chosen to be smaller as j increases in order to avoid adapting to outliers and non-repetitive noise.

– The covariance matrix M_j, where $\mu_j \sim \mathcal{N}(0, M_j)$, combines the covariance of the process and measurement noise. It is possible to obtain a value for the covariance by carrying given sensor noise characteristics and the known process disturbances of the real system from the original model description (1) through to the lifted domain representation (15). However, modeling M_j as

$$M_j = \eta_j I, \quad \eta_j > 0, \tag{25}$$

is often sufficient. The ratio between ϵ_j and η_j determines how much we trust the measurement vs. the process model, cf. (19). Often this ratio is varied by changing ϵ_j only and keeping η_j constant; that is, $\eta_j = \eta$ for all $j \in \{1, 2, \ldots\}$. Experimental results discussing the choice of η_j and ϵ_j are shown in Sect. 6.6.

– The initial value $\widehat{d}_0$ is another design parameter. Most of the time, $\widehat{d}_0 = 0$ is a reasonable first guess.

– With the starting value $P_0 = E[(d_0 - \widehat{d}_0)(d_0 - \widehat{d}_0)^T]$, the initial error variance is specified. Choosing P_0 to be a diagonal matrix with large positive elements on the diagonal, results in larger changes of $\widehat{d}_j$ at the beginning of the learning.

Note that the entries of d_j and y_j, and consequently of ω_j and μ_j, represent different quantities with different units (e.g. position, velocity, etc.), whose nominal values may differ by orders of magnitude. To account for this, it may be beneficial to introduce scaling matrices S_Ω and S_M, and define

$$\Omega_j = S_\Omega(\epsilon_j I)S_\Omega^T, \qquad M_j = S_M(\eta_j I)S_M^T. \tag{26}$$

To summarize, the advantage of the above approach lies in its explicit incorporation of noise characteristics and its model-based update rule (23), which provides an optimal estimate $\widehat{d}_j$ in the context of the *a priori* tunable parameters (namely the covariances of the disturbances ω_j and μ_j, and the initial values P_0 and $\widehat{d}_0$). The algorithm for the disturbance estimation takes all available information $y_1, y_2, \ldots, y_j$ into account. In addition, the modeling error along the desired trajectory $\widehat{d}_j$ may lead to a better system understanding and may be re-used to update the dynamic model (1) or when learning a different reference trajectory.

Based on the disturbance estimate $\widehat{d}_j$, the feed-forward input signal can be adapted in order to compensate for the estimated disturbance, resulting in an improved performance in the next trial.

2.3 Input update

The learning algorithm is completed by the subsequent learning update. Making use of the information provided by the estimator, cf. Sect. 2.2, we derive a nonlinear model-based update rule, which calculates a new input sequence $u_{j+1} \in \mathbb{R}^{Nn_u}$ in response to the estimated disturbance $\widehat{d}_j$.

The objective of the update step is to find an input u_{j+1}, which optimally compensates for the identified disturbance $\widehat{d}_j$. In the context of (15), this means finding an input u_{j+1}

that minimizes the deviation from the nominal trajectory in the next trial. More precisely, we consider the expected value of x_{j+1} given all past measurements,

$$E\big[x_{j+1}\,\big|\,y_1, y_2, \ldots, y_j\big] = F u_{j+1} + \widehat{d}_j. \tag{27}$$

The constraints (18) are explicitly taken into account when solving for the optimal u_{j+1}. We approximate the future state x_{j+1} in q_{j+1}, cf. (18), by (27), resulting in a constraint inequality that depends only on the decision variable u_{j+1}.

The update rule can be expressed by the following optimization problem:

$$\begin{cases} \min\limits_{u_{j+1}} \big\|S(F u_{j+1} + \widehat{d}_j)\big\|_\ell + \alpha\|D u_{j+1}\|_\ell \\ \text{subject to} \quad L_{\text{opt}} u_{j+1} \le q_{\max}, \end{cases} \tag{28}$$

where $\alpha \ge 0$ weights an additional penalty term, which was included into the objective function as a means of directly penalizing the input. Via the matrix D, the input itself or discrete approximations of its time derivatives can be penalized. This may be beneficial if one wants to enforce smoothness of the optimal input. In Sect. 6.6, an appropriate choice of D and α is discussed in the context of the quadrocopter example. Note that high α values may corrupt or even destroy the learning performance since more emphasis is put on achieving a small (or smooth) input than on minimizing the error along the trajectory.

The matrix $S \in \mathbb{R}^{Nn_x \times Nn_x}$ in (28) allows the original error signal (27) to be scaled, and serves several objectives: equalizing the magnitude of the different physical quantities in the lifted domain, penalizing deviations of certain states more than others, or weighting specific parts of a trajectory (e.g. the first part).

The vector norm $\ell, \ell \in \{1, 2, \infty\}$, of the minimization (28) affects the convergence behavior and the result of the learning algorithm. For a vector $p = (p^{(1)}, p^{(2)}, \ldots, p^{(n_p)}) \in \mathbb{R}^{n_p}$, the one norm ($\ell = 1$), the Euclidean norm ($\ell = 2$), and the maximum norm ($\ell = \infty$), are defined as

$$\|p\|_1 = \sum_{i=1}^{n_p} \big|p^{(i)}\big|, \qquad \|p\|_2 = \sqrt{p^T p},$$

$$\|p\|_\infty = \max_{i \in \{1,2,\ldots,n_p\}} \big|p^{(i)}\big|.$$

The update law defined by (28) can be transformed into a standard convex optimization problem. More precisely, we obtain a linear program for $\ell \in \{1, \infty\}$ and a quadratic program for $\ell = 2$, cf. Boyd and Vandenberghe (2004). Details on the transformation are provided in the Appendix.

Linear and quadratic programs can be solved very efficiently using existing software packages such as the IBM ILOG CPLEX Optimizer (2012) (see also comments in Sect. 2.6). Further, if the optimization problem is feasible (i.e., if there exist u_{j+1} that satisfy the constraints), then there exists a local minimum that is globally optimal. Furthermore, we formulated the update law as a convex optimization problem so that we could incorporate the input and state constraints explicitly. Such constraints are present in any existing real system and have a notable influence on the dynamic behavior of the system. In particular, when learning high performance maneuvers, constraints often represent the limiting factor for further improvement and should, as such, be taken explicitly into account. Relevant constraints of the quadrotor vehicles and their effects during the learning experiments are illustrated in Sect. 3 and following sections.

Design parameters Four different design parameters allow for an adaptation of the optimization problem (28):

– The norm ℓ defines the overall performance objective of the learning algorithm. While $\ell = \infty$ minimizes the maximum deviation from the desired trajectory, the one norm and Euclidean norm minimize the "average error"; that is, they minimize the sum of the deviations along the reference trajectory, where the Euclidean norm weights large errors more than the one norm does. In Sect. 6.6, the learning performance of different norms is experimentally evaluated for the quadrotor tracking problem.
– For an intuitive design, the scaling matrix S may be decomposed into three diagonal matrices,

$$S = T_W S_W S_x, \quad T_W, S_W, S_x \in \mathbb{R}^{Nn_x \times Nn_x}. \tag{29}$$

First the state scaling matrix S_x scales the lifted state vector such that all entries of the scaled vector representation x^s, $x^s = S_x x$, are within the same range of magnitude. The state-weighting matrix S_W puts emphasis on specific states of the original system (1) by penalizing their deviations from the reference trajectory more than the deviations of the other states. The matrices S_x and S_W are usually defined via vectors s_x and s_W of length n_x, which are repeatedly placed along the corresponding matrix diagonal. Finally, the matrix T_W allows us to weight particular parts of the trajectory more than others, i.e. to change the scaling along the trajectory.
– When aiming to penalize the input u directly, the matrix D is chosen as the identity matrix, where the lifted vector u represents the deviation from the nominal input trajectory (12). Another option is to consider the rate of change of u or its curvature by choosing Du such that the kth component is given by

$$(Du)^{(k)} = \frac{\tilde{u}(k+1) - \tilde{u}(k)}{\Delta t} \qquad \text{or} \tag{30}$$

$$(Du)^{(k)} = \frac{\tilde{u}(k+2) - 2\tilde{u}(k+1) + \tilde{u}(k)}{(\Delta t)^2}, \tag{31}$$

representing the discrete approximation of the first or second derivative of u, respectively.

– The scalar α balances the influence of the state deviation component and the input component in the objective function (28). In Sect. 6.6 design criteria for α are discussed in the context of the quadrocopter tracking experiment.

In brief, the above parameters enable us to enforce a specific learning behavior, and thus meet individual performance criteria.

2.4 Extending horizon

Now that we have familiarized the reader with the main ideas of the algorithm in question (namely, the disturbance estimation and input update), we wish to introduce a means of gradually extending its time horizon such that only small deviations from the desired trajectory are guaranteed.

We earlier assumed that each trial u_j, $j \in \{1, 2, \ldots\}$, is performed over the full time horizon $\mathcal{T} = [t_0, t_f]$. However, all derivations in Sects. 2.2 and 2.3 build upon the lifted domain representation, which was the result of a linearization of the system dynamics about the nominal trajectory. The linearization can only be justified if the actual trajectory of the system stays close to the desired one. From a different perspective, it is important that the lifted matrices F and G are valid first-order approximations of the system dynamics as this guarantees a proper interpretation of the measurement (estimation step) and a correct input adaptation (update step). In this section, we introduce a *termination condition* that stops a learning trial whenever the deviation between the real and the desired trajectory grows too large. This guarantees that previous derivations are valid and lead to successful learning.

The general idea of the method is as follows: A new trial is started with an input $u_j \in \mathbb{R}^{Nn_\mathrm{u}}$. During the execution, the actual trajectory may begin to diverge from the reference trajectory; if the deviation exceeds a certain boundary (as specified by the termination condition), the trial is stopped. In this case the learning algorithm, cf. Sects. 2.2 and 2.3, considers only the first part of the execution (until the premature ending at $N_j < N$) and returns an updated input for the first part of the trajectory. The updated input segment is then extended by the last values of the reference input $u^*(k)$, cf. (9), and a new trial $(j+1)$ is executed. In general, the following trial $(j+1)$ performs better during the initial segment of the trajectory and is terminated at a later stage $N_{j+1} \geq N_j$. That is, *the time horizon of the learning algorithm is gradually extended.*

The termination condition is defined on the system's output deviation $\tilde{y}(k) \in \mathbb{R}^{n_\mathrm{y}}$, cf. (6),

$$h\big(\tilde{y}(k)\big) \geq \gamma(k), \tag{32}$$

where $h(\cdot)$ maps the system output at time k to the relevant termination variables, whose critical values are defined by $\gamma(k) \in \mathbb{R}^{n_\gamma}$. If condition (32) is satisfied for some k, the trial j is stopped and $N_j := k$.

Assuming that $N_j \geq N_{j-1}$, the input update is performed for the subproblem, $u_j \in \mathbb{R}^{N_j n_\mathrm{u}}$ and $y_j \in \mathbb{R}^{N_j n_\mathrm{y}}$, such that only the effective length of execution is considered. A current estimate of the modeling error $\widehat{d}_j \in \mathbb{R}^{N_j n_\mathrm{x}}$ is obtained from (22)–(23). In addition to the recent input and output, u_j and y_j, the estimation takes advantage of the previous estimate $\widehat{d}_{j-1} \in \mathbb{R}^{N_{j-1} n_\mathrm{x}}$ and covariance $P_{j-1} \in \mathbb{R}^{N_{j-1}n_\mathrm{x} \times N_{j-1}n_\mathrm{x}}$. However, because of the extended time horizon N_j, these values must be extended for the current estimation using the initial conditions P_0 and $\widehat{d}_0$: with $l, m \in \{1, 2, \ldots, N_j\}$,

$$^{(c)}P_{j-1}^{(l,m)} = \begin{cases} P_{j-1}^{(l,m)} & \text{for } l, m \in \{1, \ldots, N_{j-1}\}, \\ P_0^{(l,m)} & \text{otherwise,} \end{cases} \tag{33}$$

where $P^{(l,m)} \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{x}}$ represents the (l, m)th entry in P and $^{(c)}P$ denotes the adaptation to the current length N_j; similarly,

$$^{(c)}\widehat{d}_{j-1}^{(m)} = \begin{cases} \widehat{d}_{j-1}^{(m)} & \text{for } m \in \{1, \ldots, N_{j-1}\}, \\ \widehat{d}_0^{(m)} & \text{for } m \in \{(N_{j-1}+1), \ldots, N_j\}, \end{cases} \tag{34}$$

where $\widehat{d}^{(m)} \in \mathbb{R}^{n_\mathrm{x}}$.

The matrices G, F, Ω_{j-1}, and M_j are adapted analogously. The resulting estimate $\widehat{d}_j$ from the estimation step is used in (28) to calculate the updated input $^{(f)}u_{j+1} \in \mathbb{R}^{N_j n_\mathrm{u}}$ for the first part (f) of the trajectory. This input is continued by the last entries of the nominal input; that is,

$$u_{j+1}^{(m)} = \begin{cases} ^{(f)}u_{j+1}^{(m)} & \text{for } m \in \{1, \ldots, N_j\}, \\ \mathbf{0} & \text{for } m \in \{N_j + 1, \ldots, N\}, \end{cases} \tag{35}$$

where $u^{(m)}, \mathbf{0} \in \mathbb{R}^{n_\mathrm{u}}$ and $\mathbf{0}$ is a zero vector. The input is applied to the system during the following trial. If the $(j+1)$th trial is stopped earlier again, the input u_{j+1} is cropped accordingly, such that $u_{j+1} \in \mathbb{R}^{N_{j+1}n_\mathrm{u}}$. If $N_{j+1} < N_j$, the algorithm falls back to time horizon N_{j+1} and all variables are cropped correspondingly. Measurement information for times k larger than N_{j+1} (obtained from earlier iterations) is discarded.

This method of extending the horizon not only guarantees the validity of the linearization in Sect. 2.1, but also responds to safety requirements during the learning process by guaranteeing only small deviations from the desired trajectory.

Design parameters This part of the learning algorithm features the following design parameters:

– The termination function $h(\cdot)$ defines critical variables originating from either the nonlinearity of the system equations (1) or from the safety requirements. The choice of $h(\cdot)$ is very specific to the problem under consideration. One example is to put constraints on the states that introduce the nonlinearity to the system.

– The bound $\gamma(k)$ may vary along the trajectory. Its size is crucial for the convergence of the learning. When $\gamma(k)$ is too small, the learning might never reach the final length N, since the system is not able to achieve these small deviations everywhere along the trajectory; if $\gamma(k)$ is too large, it may lead the system into regions where the linearization approximation is not accurate enough.

Even though most papers on ILC are based on a linear system representation, the issue of staying in the region where the linearization is an acceptable approximation, has not been a focus of consideration, see e.g. Rice and Verhaegen (2010). The idea of gradually extending the trial time is novel to our approach.

Examples for reasonable termination conditions are given in Sect. 6.5 for the quadrocopter experiments. Section 6.5 also shows an example where $N_{j+1} < N_j$. Note, however, that the majority of experiments shown in Sect. 6 were performed without a termination condition and still showed learning convergence.

2.5 Summary of the algorithm

The proposed learning algorithm requires a dynamic model of the physical system under consideration (see Fig. 1). This nominal model serves two main purposes: (i) given a desired trajectory, it is used to obtain an initial guess of the feed-forward input, and (ii) it provides the direction for feed-forward corrections in the input update step. Thus, the nominal model needs to approximate the system dynamics (in the proximity of the desired trajectory) to first order. Note that any physical system, including systems with underlying feedback control, can be considered as long as a nominal model is available. Ideally, the desired trajectory is feasible with respect to dynamics, input and state constraints of the physical system.

The learning algorithm consists of several preparation steps that are performed offline prior to experiment, and an iterative correction step executed online after each run of the experiment. We summarize the algorithm as follows:

Prerequisites Derive a dynamics model of the system such that it captures key dynamic effects and relevant input and state constraints.

Offline preparations

(a) Define a desired trajectory and find the corresponding nominal input based on the dynamics model.
(b) Linearize the model about the nominal trajectory, discretize, and build lifted system representation.
(c) Choose design parameters of the estimation and update step. Optional: define a termination condition.
(d) Calculate the Kalman filter gains K_j.
(e) Set $j = 1$ and apply the nominal input, i.e. $u_1 = 0$.

Online refinement through experiments

(a) Run experiment with u_j. Check if termination condition is satisfied.
(b) Stop if the termination condition is satisfied or if the task is completed.
(c) Update the disturbance estimate $\widehat{d}_j$ based on the measurement y_j.
(d) Solve the optimization problem using $\widehat{d}_j$ and obtain the next input u_{j+1}.
(e) Set $j = j + 1$, go to (a).

2.6 Computational complexity

Each of the steps of the learning algorithm highlighted above are of different computational complexity. Below, we provide a brief overview on the complexity of the key steps of the algorithm. We distinguish between offline preparation steps and calculations that are performed online after each iteration.

Recall that the length of the desired trajectory (number of discrete time steps) is denoted by N, where $N = t_f/\Delta t$ increases when extending the duration of the trajectory or when reducing the sampling time.

Offline preparations The offline cost is dominated by the computation of the Kalman filter gains K_j according to (22) for $j = 1, \ldots, J_{\max}$, where $J_{\max}$ denotes the total number of iterations. The computational complexity of calculating the Kalman filter gains is of order $\mathcal{O}(J_{\max} N^3(n_x^3 + n_y^3))$.

Online refinement The online cost comprises two steps: updating the disturbance estimate according to (23) and solving the optimization problem (28). The disturbance estimate update requires $\mathcal{O}(N^2 n_x n_y)$ arithmetic operations. The optimization problem has the form of an inequality-constrained linear or quadratic program, see Sect. 2.3. Problems of this type can be solved by interior-point methods. The complexity is given by the total number of Newton steps required for the solution of the optimization problem multiplied by the cost of one Newton step. Under mild assumptions, Boyd and Vandenberghe (2004) show that the worst-case number of Newton steps is of order $\mathcal{O}(N_c \log N_c)$ (and $\mathcal{O}(\sqrt{N_c})$ for a particular choice of parameters), where N_c is the total number of constraints. The cost of one Newton iteration is a polynomial function of the problem dimensions. For (28), the total number of constraints is of order $\mathcal{O}(N(n_c + n_x + n_u))$ for the 1- and ∞-norm, and of order $\mathcal{O}(Nn_c)$ for the 2-norm (see Appendix), where n_c denotes the number of constrained quantities. For completeness, the number of decision variables in the optimization problem is $\mathcal{O}(Nn_u)$ for the 2- and ∞-norm and $\mathcal{O}(N(2n_u + n_x))$ for the 1-norm. In practice, convex optimization problems are tractable for a large number of decision variables and constraints. As an example, cf. Boyd and Vandenberghe (2004), a linear program with "hundreds of variables and thousands of constraints" can be solved "on a small desktop computer, in a matter of seconds." In Sect. 6.7, we provide specific computations times for the quadrocopter example.

3 Quadrocopter dynamics and constraints

We now apply the iterative learning algorithm to quadrotor vehicles, with the objective of precisely tracking trajectories that are defined in the vertical plane. Considering two-dimensional trajectories only allows for the use of a more concise quadrocopter model (for both, trajectory generation and learning), while the corresponding learning results still highlight the key characteristics of the proposed algorithm, cf. Sect. 6. The subsequent derivations generalize to 3D trajectories and are equally tractable for this class of problems, see Sect. 6.7.

A two-dimensional model of the quadrocopter dynamics is derived from first principles under the assumption that the out-of-plane dynamics, including vehicle yaw, are stabilized separately (Fig. 2). The equations of motion that define the evolution of horizontal position y, the vertical position z and the quadrocopter's roll angle ϕ are

$$\ddot{z} = (f_a + f_b + f_c + f_d)\cos\phi - g, \tag{36}$$

$$\ddot{y} = -(f_a + f_b + f_c + f_d)\sin\phi, \tag{37}$$

$$I_{xx}\ddot{\phi} = ml(f_a - f_c), \tag{38}$$

where m denotes the mass of the vehicle, g represents the gravitational constant, l is the distance from the center of mass of the vehicle to a propeller, I_{xx} is the moment of inertia about the out-of-plane principal axis, and f_a and f_c are the thrust forces produced by the two in-plane rotors normalized by the mass of the vehicle (Fig. 3). The mass-normalized forces of the other two rotors, f_b and f_d, are used to stabilize out-of-plane motion and are nominally set to the average of f_a and f_c,

$$f_b = f_d = (f_a + f_c)/2. \tag{39}$$

In the two-dimensional scenario, the inputs to the quadrocopter are the collective acceleration produced by the four motors,

$$f_{coll} = f_a + f_b + f_c + f_d = 2(f_a + f_c), \tag{40}$$

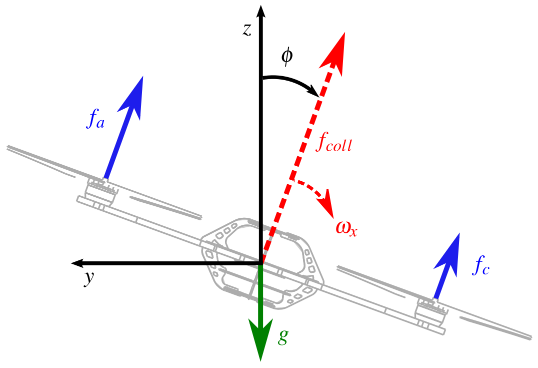

Fig. 2 Schematic drawing of a quadrocopter moving in the vertical yz-plane with relevant coordinates (y, z, and ϕ) and control inputs (f_{coll} and ω_x)

and the roll rate ω_x, cf. Fig. 2. The resulting dynamics are

$$\begin{aligned} \ddot{z} &= f_{coll}\cos\phi - g, \\ \ddot{y} &= -f_{coll}\sin\phi, \\ \dot{\phi} &= \omega_x, \end{aligned} \tag{41}$$

where $\mathrm{x} = (y, \dot{y}, z, \dot{z}, \phi)$ and $\mathrm{u} = (f_{coll}, \omega_x)$ in the framework of (1). Consequently, the feed-forward input corrections of the learning step are applied at the level of thrust and rate. In (41), we assume that the roll rate $\dot{\phi}$ can be controlled directly. In reality, an underlying high-bandwidth controller on board of the vehicle tracks the commanded rates using feedback from gyroscopes, cf. Sect. 5.2. Because the quadrocopters can achieve exceptionally high angular accelerations (typically on the order of several hundred rad/s^2), and thus can respond very quickly to changes in the desired rotational rate, this is a valid approximation for the learning algorithm and trajectory generation. We also assume that the collective thrust can be changed instantaneously. True thrust dynamics are as fast as the rotational dynamics, with propeller spin-up being faster than spin-down. We can—directly or indirectly—measure all five states; that is $\mathrm{y} = \mathrm{x}$ in (1).

The mass-normalized single motor thrusts f_a and f_c are related to the input u by the following equations:

$$f_a = \frac{1}{4} f_{coll} + \frac{I_{xx}}{2ml}\dot{\omega}_x, \qquad f_c = \frac{1}{4} f_{coll} - \frac{I_{xx}}{2ml}\dot{\omega}_x. \tag{42}$$

The first-principles model presented above neglects numerous aerodynamic effects, such as drag, blade flapping (Pounds et al. 2006), or changes in the angle of attack of the propellers (Huang et al. 2009), which may have a significant effect on the dynamic behavior of the quadrocopter, especially at high speeds. These effects are difficult to model and are presumed to be compensated by the iterative learning scheme.

The system is subject to several constraints that result from both limited actuator action and limited range of sensor

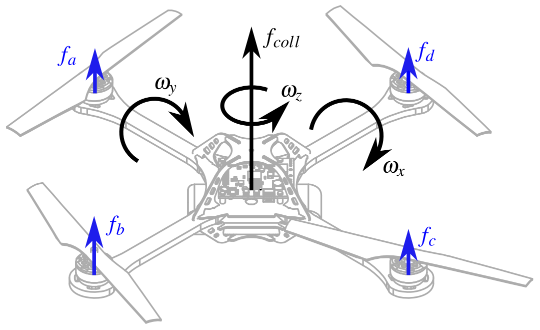

Fig. 3 The control inputs of the quadrocopter are the body rates ω_x, ω_y, and ω_z, and the collective thrust f_{coll}. These inputs are converted by an onboard controller into motor forces $f_i, i \in \{a,b,c,d\}$

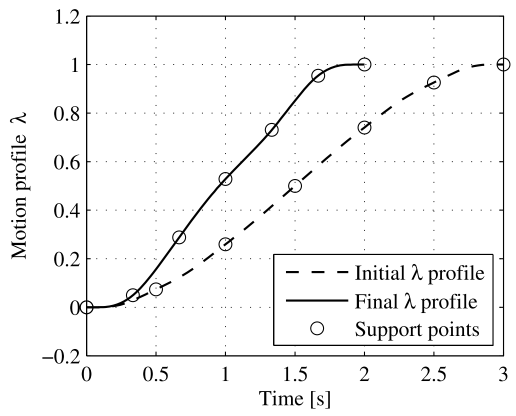

Fig. 4 Initial and optimized motion profiles. The support points are distributed equally over time and their λ values are the optimization variables

measurements. First, the thrust that each motor can provide is limited by

$$f_{\min} \le f_i \le f_{\max}, \quad i \in \{a,b,c,d\}. \tag{43}$$

Second, due to the motor dynamics, the rate of change of the thrust is also limited:

$$|\dot{f}_i| \le \dot{f}_{\max}, \quad i \in \{a,b,c,d\}. \tag{44}$$

Equation (43) imposes a constraint on the maximum feasible angular acceleration via (38),

$$|\ddot{\phi}| \le \ddot{\phi}_{\max}. \tag{45}$$

Furthermore, the onboard rate gyroscopes have a limited measurement range. Taking into account an additional safety margin for the onboard control, a sufficiently conservative constraint on the angular velocity is derived as:

$$|\dot{\phi}| \le \dot{\phi}_{\max}. \tag{46}$$

Note that the above constraints implicitly constrain the inputs (f_{coll}, ω_x) via (42). Later, in the input update of the learning algorithm, we consider upper and lower bounds on the input, its derivatives, and on f_a and f_c. Note that this constraint definition contains some redundancy. In practice, computation time (cf. Sect. 6.7) can be reduced by constraining only f_a, f_c and ω_x. Derivative constraints are usually satisfied because of the time discretization of the input signal.

4 Trajectory generation for quadrocopters

The goal of this section is to plan feasible quadrocopter trajectories (with respect to the constraints introduced in Sect. 3) that track arbitrary user-defined shapes in the vertical plane.

As described in Sect. 2, a feasible state trajectory with its corresponding nominal input is the starting point, and hence a prerequisite of the proposed learning scheme. In this section, we describe an approach for generating feasible state trajectories starting with minimal information on the geometry of the desired state evolution. Once the state trajectory is known as a function of time, the corresponding input is computed from the quadrocopter model in Sect. 3.

This approach is similar to Hoffmann et al. (2008), Cowling et al. (2007), Bouktir et al. (2008), where the trajectory generation problem is split into two parts. First, the trajectory's geometry is defined using a set of basis functions such as splines or polynomials. The trajectory's geometry or shape (hereafter referred to as 'path') does not contain any time information. In the second step, a motion profile is assigned to the path. This profile is chosen such that the resulting time-parameterized trajectory satisfies the feasibility constraints.

In the following, both the path and its motion profile are defined by splines (cf. Curve Fitting Toolbox Splines and MATLAB Splines 2012; Zhang et al. 2010; Lin and Lian 2005; Piazzi and Guarino Lo Bianco 2000), and the trajectory generation is posed as a constrained optimization problem with the objective of minimizing the trajectory's end time.

The trajectory generation algorithm for two-dimensional quadrocopter maneuvers comprises the following steps:

(a) Define the shape of the trajectory, referred to as 'path', by specifying N_p points in the yz-plane,

$$\{p^{(1)}, \dots, p^{(N_p)}\} \quad \text{with } p^{(i)} \in \mathbb{R}^2, i \in \{1,2,\dots,N_p\}. \tag{47}$$

(b) Each point $p^{(i)}$ is assigned a chord-length parameter $\lambda^{(i)}$,

$$\lambda^{(i)} = \frac{i-1}{N_p - 1}, \quad i \in \{1,2,\dots,N_p\}, \tag{48}$$

resulting in an ordered sequence $(\lambda^{(i)}, p^{(i)})$, which defines the shape of a spline $\mathcal{P}$. In other words, a continuous path in the yz-plane is obtained as a mapping from λ, $\lambda \in [0, 1]$, to points in $\mathbb{R}^2$:

$$\mathcal{P}: [0,1] \to \mathbb{R}^2, \tag{49}$$

where $\mathcal{P}(\lambda^{(i)}) = p^{(i)}, i \in \{1, \ldots, N_p\}$. That is, the spline $\mathcal{P}$ passes through the previously defined points (47). The motion profile along the path is defined by $\lambda(t)$, a monotonically increasing function of time:

$$\lambda : [0, t_f] \to [0, 1], \tag{50}$$

where t_f represents the end time of the trajectory. The function $\lambda(t)$ is itself a spline defined by N_λ support points $(t^{(k)}, \sigma^{(k)})$, such that

$$\lambda\big(t^{(k)}\big) = \sigma^{(k)}, \quad k \in \{1, \ldots, N_\lambda\}. \tag{51}$$

The first and the last time point are fixed,

$$\begin{aligned} &\big(t^{(1)}, \sigma^{(1)}\big) = (0, 0), \\ &\big(t^{(N_\lambda)}, \sigma^{(N_\lambda)}\big) = (t_f, 1), \end{aligned} \tag{52}$$

and the time instances $t^{(k)}, k \in \{1, \ldots, N_\lambda\}$, are equally distributed over $[0, t_f]$,

$$t^{(k)} = \frac{k-1}{N_\lambda - 1} t_f. \tag{53}$$

The $(N_\lambda - 2)$ interior support points,

$$\Sigma := \big\{\sigma^{(i)} | i = 2, \ldots, N_\lambda - 1\big\}, \tag{54}$$

define the curvature of the λ profile and act as the decision variables in the optimization problem described in the next step, see Fig. 4. We can write $\lambda(t) = \lambda(t, t_f, \Sigma)$ to make the dependency on the parameters t_f and Σ explicit.

(c) We find a feasible motion profile by solving the constrained minimization problem:

$$\begin{cases} \min\limits_{\{t_f, \Sigma\}} t_f \\ \text{subject to} \quad \dfrac{\partial}{\partial t}\lambda(t, t_f, \Sigma) \geq 0, \quad t \in [0, t_f], \\ \qquad\qquad\quad \mathcal{P}(\lambda) \text{ feasible.} \end{cases} \tag{55}$$

The objective is to find interior support points Σ, see (54), such that the corresponding motion profile increases monotonically and yields feasible state trajectories. Feasibility for the special case of the quadrocopter is discussed at the end of this section. An important advantage of this approach is that the number of decision variables in the optimization problem is relatively small. It is equal to the number of interior support points of the motion profile λ plus the final time t_f, i.e. in total, $N_\lambda - 1$. Moreover, the number of decision variables is independent of the number of path points N_p. Generally, the constraints in (55) are non-convex. Thus, the optimization problem lacks any useful characteristics that would guarantee global optimality. Consequently, a solution of (55) is only locally optimal and depends on the initial values of the decision variables $\{t_f, \Sigma\}$.

(d) The solution of (55) yields a set of locally optimal interior support points Σ^* and a locally optimal end time t_f^*. These values uniquely define the motion profile $\lambda(t) = \lambda(t, t_f^*, \Sigma^*)$, which, in turn, determines the state trajectories $\mathcal{P}(t) = (y(t), z(t)), t \in [0, t_f^*]$, cf. Fig. 4. The nominal inputs are computed via an inversion of the system dynamics (41). Note that time derivatives of y, z can be computed analytically since the nominal state trajectories are given by splines; that is, by piece-wise polynomial functions.

In the remainder of this section, we derive constraints that guarantee the feasibility of the trajectories $\mathcal{P}(\lambda)$. First, constraints result from the assumption that the quadrocopter starts and ends in hover; that is, we derive conditions that ensure the continuity of the states and inputs at the start and end of the trajectory. We then consider constraints on the input and its first derivative.

(a) *Continuity of the states at the start and end point.* Hovering is characterized by

$$\dot{y} = \dot{z} = 0, \qquad \ddot{y} = \ddot{z} = 0, \qquad \phi = \dot{\phi} = \ddot{\phi} = 0. \tag{56}$$

Conditions (56) are satisfied if

$$\left.\frac{\partial^i}{\partial t^i}\lambda(t)\right|_{t \in \{0, t_f\}} = 0, \quad i \in \{1, 2, 3, 4\}. \tag{57}$$

(b) *Continuity of the inputs at the start and end point.* The condition (57) implicitly guarantees the continuity of the inputs at the start and end of a trajectory; that is, if (57) holds so do the following input conditions:

$$f_{coll}(t)\big|_{t \in \{0, t_f\}} = g, \qquad \omega_x(t)\big|_{t \in \{0, t_f\}} = 0, \tag{58}$$

where g denotes the gravitational constant.

(c) *Input constraints.* Input constraints are taken into account directly in the form (43)–(46).

The numeric values of the parameters used in the trajectory generation process are summarized in Table 1. In order to leave room for learning, we have chosen restrictive constraint bounds for the trajectory generation.

Considering (43)–(46) again, we observe that single thrust input trajectories $f_{a,c}(t)$ must be continuous, whereas their first derivatives $\partial/\partial t\, f_{a,c}(t)$ are allowed to have steps.

Table 1 Quadrocopter parameters used for the trajectory generation and learning

	Trajectory generation	Learning
l	0.17 m	0.17 m
m	0.468 kg	0.468 kg
I_{xx}	0.0023 kg m^2	0.0023 kg m^2
$f_{\max}$	4.5 m/s^2	5.5 m/s^2
$f_{\min}$	0.4 m/s^2	0.25 m/s^2
$\dot{f}_{\max}$	27 m/s^3	51 m/s^3
$\dot{\phi}_{\max}$	22 rad/s	25 rad/s
$\ddot{\phi}_{\max}$	150 rad/s^2	200 rad/s^2

According to (42), $f_{a,c}(t)$ are functions of $\partial/\partial t\,\omega_x(t)$, which in turn depends on the fourth derivative of the desired trajectory, $\partial^4/\partial t^4 y(t)$ and $\partial^4/\partial t^4 z(t)$. This statement is supported by (41) and is especially relevant in the inversion step of the previous algorithm. Thus, in order for $f_{a,c}(t)$ to be continuous, the fourth derivatives of $y(t)$ and $z(t)$ (with respect to time) must be continuous. This is guaranteed by describing the $\{y, z\}$-trajectories and the λ-profile by splines that are at least quintic, i.e. of order 5. However, we chose the λ-spline to be of order 9; this leaves us with 8 free parameters needed to satisfy the 8 constraints given by (57).

We implemented the algorithm in MATLAB, and solved the optimization problem with MATLAB's `fmincon` routine (The Mathworks Optimization Toolbox 2012).

5 Experimental setup

5.1 The testbed

We have demonstrated the algorithm in question on custom quadrocopters operated in the ETH Flying Machine Arena (FMA), a dedicated testbed for motion control research. The setup is similar to How et al. (2008), Michael et al. (2010): The space is equipped with an 8-camera motion capture system that, for any properly marked vehicle, provides millimeter-accurate position information and degree-precise attitude data at 200 Hz. The localization data is sent to a PC, which runs the control algorithms (including the iterative learning algorithm), and which in turn sends commands to the quadrocopters. The flying vehicles are based on the Ascending Technologies Hummingbird platform described in Gurdan et al. (2007), with custom wireless communication and central onboard electronics. More details about the test environment can be found in Lupashin et al. (2010) and on the FMA webpage.[2]

[2]www.FlyingMachineArena.org.

5.2 Quadrocopter control

Each vehicle accepts four inputs: three angular rate commands $(\omega_x, \omega_y, \omega_z)$, see Fig. 3, and a mass-normalized collective thrust command f_{coll}. These inputs are usually provided by off-board controllers.

For the experiments, we use two different modes for controlling the vehicle. The first mode, referred to below as (C1), is used to stabilize the quadrocopter at the start and end position of the trajectory. It is also used to return the quadrocopter to its initial position before starting a new trial. In this mode, the off-board controller takes desired vehicle positions as an input and closes the loop based on the camera information. The off-board controller calculates all four commands and sends them to the vehicle.

To fly and learn the desired trajectory, we use a different control mode called (C2). In this mode, the iterative learning scheme provides the collective thrust command f_{coll} and the roll rate command ω_x. The inputs ω_y and ω_z are used to stabilize the vehicle in the vertical plane. These two inputs are computed by a separate off-board feedback controller that uses position and attitude information of the vehicle.

An onboard controller does high-rate feedback control on the angular rates $(\omega_x, \omega_y, \omega_z)$ using rate gyro information. No feedback is done on the thrust command.

In summary, for (C2) the out-of-plane dynamics are stabilized using camera information; the in-plane dynamics are driven by the feed-forward input signals of the iterative learning scheme, f_{coll} and ω_x. The onboard controller closes the loop on the roll rate input ω_x to guarantee that the commanded value is actually achieved.

5.3 Implementation of learning algorithm

In order to test the proposed learning scheme on the real vehicles, we developed a software framework that manages the learning process, and allows for efficient and reliable operation of the quadrocopters. The program manages the operations of flying the quadrocopter to a defined initial position, triggering the learning trajectory, and stabilizing the vehicle at the end of the trajectory. At the core of this setup lies the learning algorithm of Sect. 2. This is implemented in C++ using *boost uBLAS* libraries (Boost—Basic Linear Algebra Library 2012) for matrix operations and the CPLEX optimizer (IBM ILOG CPLEX Optimizer 2012) to solve the convex optimization problem.

We designed the overall program as a state machine consisting of two main states, the *WAIT* and *RUN* mode, and a transition state called *AUTOSTART*, cf. Fig. 5. The general procedure is as follows: first, the desired trajectory and the corresponding nominal input are loaded from a *mat* file (generated by the algorithm presented in Sect. 4) and the settings and parameter values of the learning algorithm are

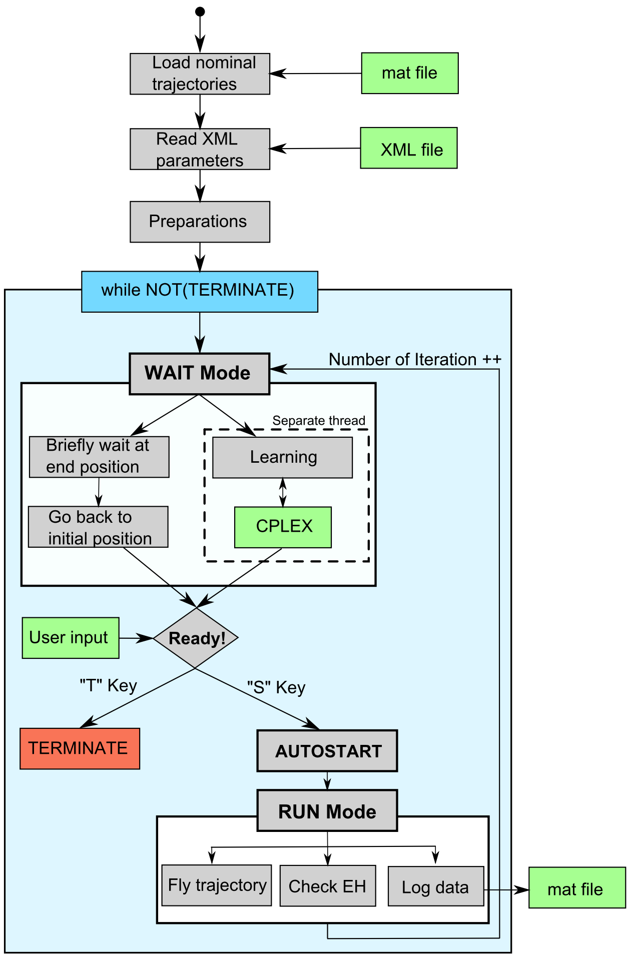

Fig. 5 Flow diagram of the implemented learning procedure

read from an *XML* file. Second, based on the nominal model (Sect. 3), the lifted-domain representation and the Kalman gains are computed. After these preliminary steps, the system enters the *WAIT* mode, where the quadrocopter hovers at a given initial point with (C1), see Sect. 5.2. As soon as the user decides to start the experiment, the *AUTOSTART* mode is activated. The system switches to a more aggressive controller of type (C1) for more precise hover performance. The goal is to achieve accurate initial conditions for the learning trajectory. As soon as the quadrocopter satisfies a set of predefined start conditions, a new iteration is triggered. In the experiments presented herein, the start conditions are defined on the translational velocity of the quadrocopter, its attitude, and the rate of change of the attitude:

$$\begin{aligned}&|\dot{x}|, |\dot{y}|, |\dot{z}| < 0.01 \text{ m/s},\\ &|\psi|, |\theta|, |\phi| < 0.05 \text{ rad}, \qquad (59)\\ &|\dot{\psi}|, |\dot{\theta}|, |\dot{\phi}| < 0.2 \text{ rad/s},\end{aligned}$$

where ψ, θ, ϕ are the yaw, pitch and roll angles that entirely define the vehicle attitude (in z–y–x Euler angle notation). The position is not considered because the quadrocopter motion is invariant with respect to the initial position. The actual initial position is simply subtracted from all following position measurements. Once the start conditions (59) are satisfied, the *RUN* mode is activated and the desired trajectory is performed with (C2) using the most recent feed-forward inputs of the learning algorithm, see Sect. 5.2. A trajectory is either fully completed or is terminated prematurely if the termination condition of the extending horizon condition is activated and satisfied. After an iteration, the system enters the *WAIT* mode, which performs two tasks simultaneously: returning the quadrocopter to the initial position, and executing the online update step of the learning algorithm (which computes a new input trajectory to be applied in the next iteration).

The program allows the execution of an arbitrary number of iterations and stores all log data.

6 Results

This section shows the experimental results of the proposed learning scheme applied to quadrotor vehicles. We consider two different trajectories: a diagonal trajectory and an S-shaped trajectory. Both were generated by the method proposed in Sect. 4. With a set of default learning parameters introduced in Sect. 6.1, the trajectories are learned after five to six iterations and tracked with an accuracy as high as the stochastic noise level of the system allows (Sect. 6.3 and Sect. 6.4). In Sect. 6.5, we apply the extending horizon method to the diagonal trajectory using two different termination conditions. The influence of different learning parameters is shown in Sect. 6.6, where we evaluate the learning performance for different objective function norms, as well as for different input penalty terms and varying noise model parameters. In order to provide insight into the computational cost associated with the approach, we specify the computation times of the different algorithmic steps in Sect. 6.7.

A video of the experiments presented herein is available online,[3] and as an electronic appendix to this article.

6.1 Default learning parameters

As highlighted in Sects. 2.2–2.4, the iterative learning scheme includes several design parameters. The default parameter values used in the subsequent experiments are summarized in Table 2.

The applied state scaling factors s_x are empirical values that map the deviations of $\mathrm{x} = (y, \dot{y}, z, \dot{z}, \phi)$ to similar magnitudes. Here, the position was scaled by a factor of 2 and

[3]The accompanying video is found at http://tiny.cc/QuadroLearnsTrajectory.

Table 2 Default learning parameters for the quadrocopter experiments (in SI units)

	Value	Description
Δt	0.02	Sampling time in seconds
ϵ_j	$[0.5, 0.3]$	Process noise variance
η	0.05	Measurement noise variance
d_0	$0 \in \mathbb{R}^{Nn_x}$	Initial disturbance estimate
P_0	I	Initial variance of disturbance
ℓ	2	Norm of input update
s_x	$[2, 1, 2, 1, 5]$	State scaling factors
s_W	$[1, 0.1, 1, 0.1, 0]$	State weighting factors
T_W	I	Trajectory weighting matrix
α	0.08	1-norm input penalty factor
	$5e^{-5}$	2-norm input penalty factor
	0.05	∞-norm input penalty factor

the angle by a factor of 5 as compared to the velocities. The state weighting s_W in Table 2 penalizes deviations of certain components in x more or less than others. We aim for a very precise position tracking. Achieving the desired velocities is less important, however, and angular errors are completely neglected in the input update rule (28). It is reasonable to prioritize between position and angular error. When an imprecise model is used in the trajectory generation, it may be impossible to follow the position trajectories while also tracking the nominal angle trajectory. We use the 2-norm as the default objective function norm. In the objective function, we use a penalty on the input's second derivative, according to (31), to obtain smooth inputs. As described in Sect. 2.2, we keep the measurement variance constant ($\eta_j = \eta$). We choose a larger process noise covariance ϵ_j during the first iterations and choose a smaller noise covariance as the number of iterations increases. The default value for the first five iterations is $\epsilon_j = 0.5$, $j \in \{1, 2, 3, 4, 5\}$. For all subsequent iterations, $j > 5$, $\epsilon_j = 0.3$ is used.

The choice of parameters is discussed in detail in Sect. 6.6, where experimental results are shown for various parameter combinations.

6.2 Learning performance measure

In order to quantitatively evaluate and compare the learning performance of different iterations and experiments, we introduce a weighted error function, which reflects the objective function (28) of the learning algorithm (not considering the input penalty term):

$$e_{w,j} = \left\| S_y y_j \right\|_\ell, \quad \ell = \{1, 2, \infty\}, \tag{60}$$

where y_j denotes the lifted-domain output vector (12), whose entries are the *measured* deviations from the reference output trajectory. The matrix S_y weights the output such that it best reflects the defined learning objective (28). For this experiment, all states $\mathrm{x} = (y, \dot{y}, z, \dot{z}, \phi)$ are measured (cf. Sect. 3) and $S_y = S$ is chosen, cf. (28). Note that y_j denotes the lifted output vector, while y is the horizontal position of the quadrocopter. The weighted error is obtained by scaling and weighting the measured output error by the same scaling and weighting matrices that were used in the objective function of the learning routine. In addition, the same norm ℓ as in the input update (28) is used in (60), reflecting the main objective of the learning scheme. In the following, we also refer to the weighted error (60) as 'weighted state error'.

This type of performance measure, however, depends on the norm ℓ. In order to compare the learning performance for different choices of ℓ (Sect. 6.6), we introduce another intuitive performance measure for the quadrocopter experiments. We use the average position error along a trajectory:

$$e_{pos,j} = \frac{1}{N} \sum_{k=1}^{N} \sqrt{\Delta y(k)^2 + \Delta z(k)^2}, \tag{61}$$

where $\Delta y(k) \in \mathbb{R}$ denotes the deviation of the quadrocopter's horizontal position (from the desired trajectory) at the discrete time k and, similarly, $\Delta z(k) \in \mathbb{R}$ is the vertical deviation. Since e_{pos} is independent of the objective function's norm, it is used to compare the learning performance of the algorithm for different objective function norms in Sect. 6.6.

6.3 Experiment 1: Diagonal trajectory

The desired trajectory of the first experiment is a diagonal motion in the yz-plane, see dashed black line in Fig. 6. Challenging for this and all subsequent motions is the coupling of the inputs. Both inputs, f_{coll} and ω_x, have an influence on both quadrocopter coordinates, the horizontal and vertical position. Second, the quadrocopter is required to hover at the beginning and the end of the trajectory. That is, the acceleration and de-acceleration phases at the beginning and end of the trajectory must be learned precisely.

In the first iteration, we apply the nominal input (obtained from the method presented in Sect. 4) to the quadrocopter. As depicted in Fig. 6, the resulting trajectory is far off. Despite the large initial discrepancy, the vehicle learns to track the reference trajectory over the next four iterations. Figure 7 shows the corresponding evolution of the tracking error and Fig. 8 highlights the convergence of the feed-forward input corrections. Although the feed-forward input converges, the corresponding trajectories in Fig. 6 vary around the desired trajectory, resulting in non-zero tracking errors (Fig. 7). These variations reflect an important characteristic of feed-forward based learning approaches. The feed-forward input that is adapted during the learning process (see Fig. 8) is only able to compensate for repetitive

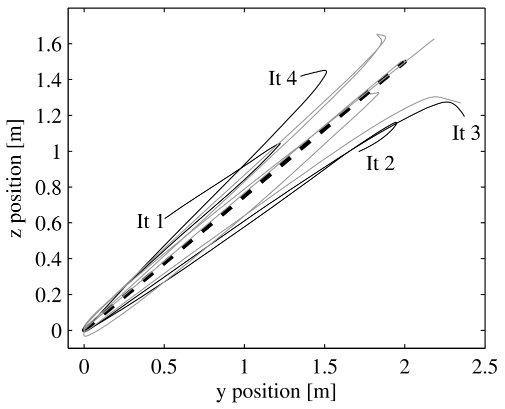

Fig. 6 *Experiment 1*: Learning a diagonal trajectory. The quadrocopter position in the yz-plane is depicted for different iterations. *The dashed black line* shows the desired trajectory. The trajectories of iterations 1–4 are drawn in *black*, iterations 5–10 are shown in *grey color*

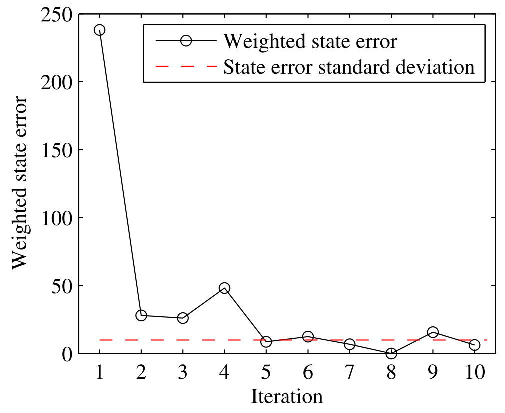

Fig. 7 *Experiment 1*: Error convergence for the diagonal trajectory. The error is computed according to (60). *The dashed line* illustrates the standard deviation of the tracking error when applying the same diagonal-trajectory input to the vehicle and observing the variations in the performed trajectories. It can be viewed as a measure of the noise level in the experimental setup

disturbances while any non-repetitive noise directly affects the tracking performance. More precisely, the tracking accuracy is lower-bounded by the level of stochastic (i.e. non-repetitive) noise that acts on the system output.

In order to measure the non-repetitive noise level of our system, we repeatedly apply the same diagonal-trajectory input and observe the variations in the output. Figure 9 shows the corresponding state trajectories. We calculate the respective tracking errors and compute their standard deviation. This value characterizes the system noise level and serves as a lower bound for the achievable tracking error, illustrated in Fig. 8 by the dashed horizontal line.

The tracking error in Fig. 8 reaches magnitudes that are in the range of the non-repetitive variability of the system. Consequently, the visible variations of the output trajectories (Fig. 6, grey solid lines) are due to non-repetitive noise and

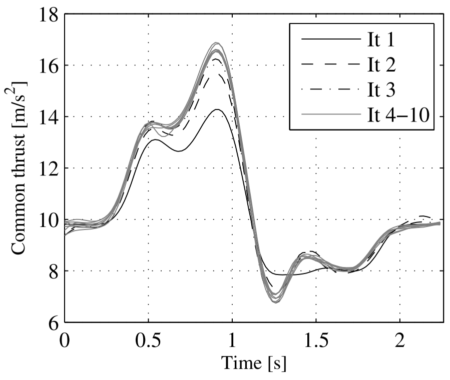

Fig. 8 *Experiment 1*: The thrust input converges for an increasing number of executions of the diagonal trajectory. The roll rate input converges similarly

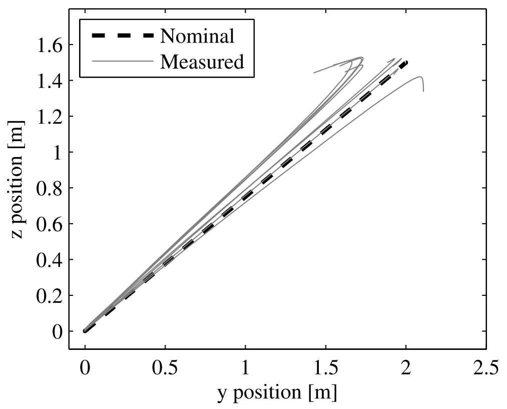

Fig. 9 Quadrocopter state trajectories for the same feed-forward input (applied repeatedly to the vehicle). Note that the system output varies for identical inputs due to non-repetitive noise acting on the system

are comparable in size to Fig. 9. We conclude that the learning algorithm is able to effectively compensate for repetitive disturbances and achieves the best possible tracking performance for the given overall system setup.

Moreover, this experiment proves the robustness of the proposed learning algorithm to modeling errors (reflected by the large initial tracking error). Based on the simplified model of Sect. 3, the algorithm is able to learn the desired trajectory in a few iterations.

Statistical information for the proposed learning scheme (including mean and variance of the errors in Fig. 7) are presented in Sect. 6.6 and derived from performing the same learning experiment several times. In Sect. 7, we discuss possibilities to decrease the system noise level and consequently improve the tracking performance.

6.4 Experiment 2: S-shaped trajectory

The proposed framework enables us to define and learn arbitrary trajectories in the vertical plane. In the second experiment, we consider an S-shaped trajectory, cf. Fig. 10 dashed

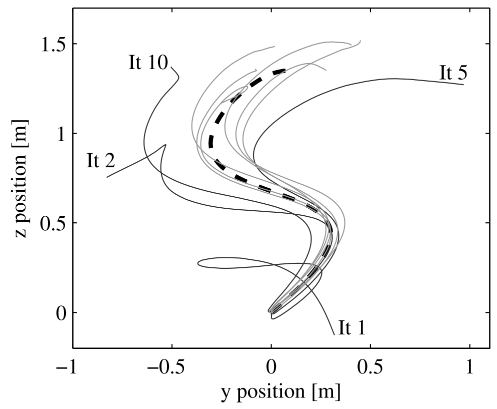

Fig. 10 *Experiment 2*: Learning an S-shaped trajectory. The quadrocopter position in the yz-plane is depicted for different iterations. *The black dashed line* shows the desired trajectory. The trajectories of iterations 1,2,5,10 are drawn in *black*, iterations 3,4,6–9 are shown in *grey color*

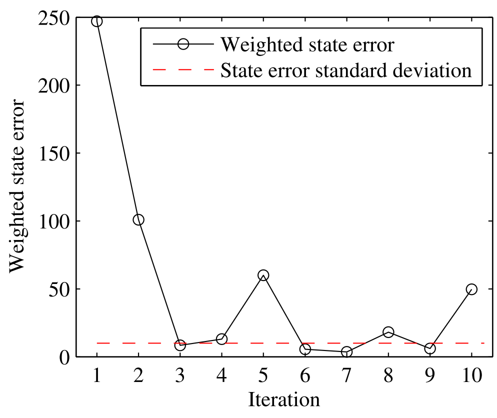

Fig. 11 *Experiment 2*: Error convergence for the S-shaped trajectory. The error is computed according to (60). *The dashed line* illustrates the standard deviation of the tracking error when applying the same S-shaped trajectory input to the vehicle and observing the variations in the performed trajectories. It can be viewed as a measure of the noise level in the experimental setup

line. The experimental results show the same system characteristics as discussed in Sect. 6.3 and are summarized in Figs. 10, 11, 12: Starting from a poor initial performance, the tracking accuracy improves quickly, but is bounded by the system's inherent stochastic variability. Moreover, despite the two outliers at iteration 5 and 10, the feed-forward corrections converge (Fig. 12) to values that best compensates for the occurring repetitive disturbances. The Kalman filter, which provides the disturbance estimate, handles outliers effectively by averaging them out rather than over-adapting. The outliers may be caused by the more challenging motor actuation required for the S-shaped trajectory, which implies larger changes and change rates in the single motor thrusts. In brief, the experiment highlights the algorithm's robustness to outliers caused by non-repetitive noise.

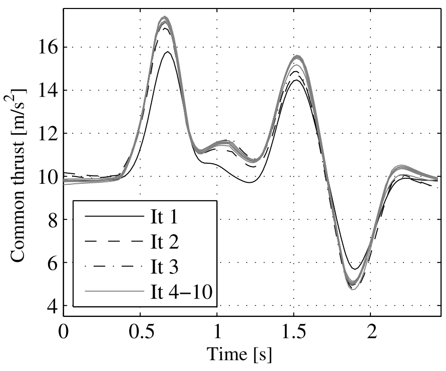

Fig. 12 *Experiment 2*: The input trajectories converge for an increasing number of executions of the S-shaped trajectory. The roll rate input converges similarly

6.5 Experiment 3: Extending horizon

An additional component of the presented learning algorithm is the extending horizon feature. This feature terminates a trial as soon as a predefined termination condition is satisfied, cf. Sect. 2.4, and is a means of guaranteeing that the actual motion stays close to the desired trajectory. The following types of termination conditions (TC) are tested in experiments:

– *TC1*: Terminate the current trial if

$$|\Delta z| \geq 0.1 \text{ m}. \tag{62}$$

– *TC2*: Terminate the current trial if

$$\sqrt{(\Delta y)^2 + (\Delta z)^2} \geq 0.2 \text{ m}. \tag{63}$$

The same diagonal reference trajectory as in Sect. 6.3 is considered again. Figures 13 and 15 show learning results using (62) and (63), respectively. In both examples, the horizon is gradually extended until, in iteration 5, the entire trajectory is flown within the prescribed bound. Figures 14 and 16 show the corresponding error trajectories. Note that in the second experiment (Fig. 15 and Fig. 16), the time horizon of iteration 3 is shorter than the one of iteration 2. In this case, the online learning update is performed only for the first part of the trajectory, until the termination time of iteration 3, and all measurement information for larger times (from earlier iterations) is discarded. For the particular bounds (62) and (63), extending horizon learning takes a similar amount of iterations to converge as learning without a termination condition, cf. Sect. 6.3. If smaller bounds are used, however, the learning may take many more iterations or may not even converge. This is particularly the case if the bounds are smaller than the noise level of the system, cf. Sect. 6.3.

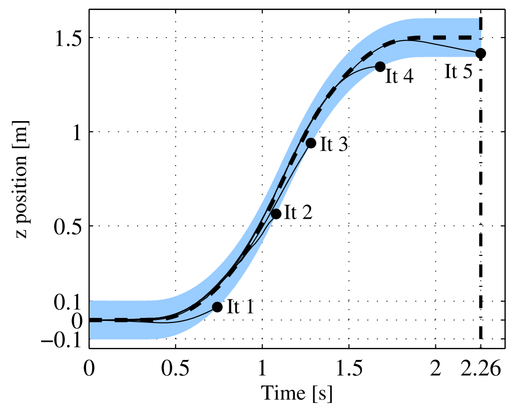

Fig. 13 *Experiment 3*, *TC1*: Learning of the diagonal trajectory with extending horizon and termination condition: $|\Delta z| \geq 0.1$ m. The learning horizon is gradually increased from iteration to iteration. *The black dashed line* depicts the desired trajectory of the vertical quadrocopter position over time, *the shaded area* represents the extending horizon bound, and *the black dots* show the end of an executed trajectory

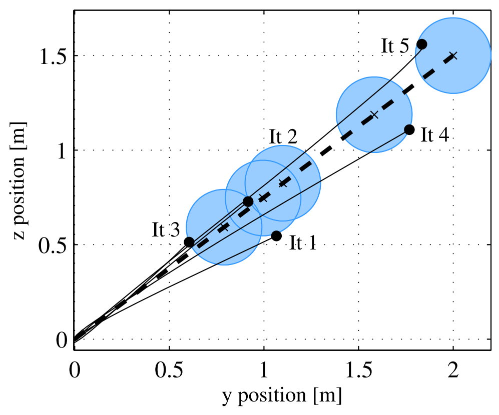

Fig. 15 *Experiment 3*, *TC2*: Learning of the diagonal trajectory with extending horizon and termination condition: $\sqrt{(\Delta y)^2 + (\Delta z)^2} \geq 0.2$ m. *The black dashed line* shows the desired trajectory in the yz-plane, *the shaded disks* represent the extending horizon bound, and *the black dots* show the end of an executed trajectory

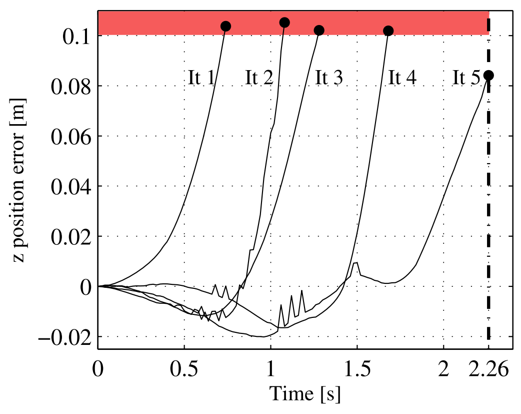

Fig. 14 *Experiment 3*, *TC1*: Learning of the diagonal trajectory with extending horizon and termination condition: $|\Delta z| \geq 0.1$ m. *The solid black lines* illustrate the z-position error and *the black dots* show the end of an iteration. *The shaded area* represents the termination condition (62). *The vertical dashed-dotted line* marks the end of the desired trajectory

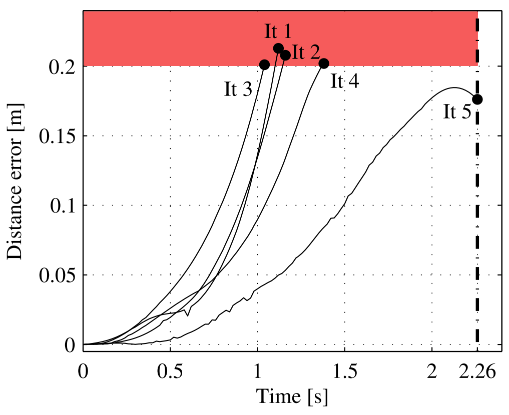

Fig. 16 *Experiment 3*, *TC2*: Learning of the diagonal trajectory with extending horizon and termination condition: $\sqrt{(\Delta y)^2 + (\Delta z)^2} \geq 0.2$ m. *The black solid lines* illustrate the distance error evolution and *the black dots* show the end of an iteration. *The shaded area* represents the termination condition (63). *The vertical dashed-dotted* line marks the end of the desired trajectory

6.6 Influence of different learning parameters

We now discuss the influence of different parameter settings on the learning performance. We use the diagonal trajectory as desired trajectory throughout this section, allowing us to compare the subsequent experiments with previous results in Sects. 6.3 and 6.5. We also give more insight into the typical characteristics of the proposed learning scheme, and attempt to justify the choice of the default parameters (Table 2).

Unless otherwise stated, the parameter values given in Table 2 apply.

Choice of input penalty factor α When solving the optimization problem (28) with $\alpha = 0$, the feed-forward corrections we obtain are highly non-smooth and jitter within the allowed bounds (43)–(46). This effect is observed even in noise-free simulations. One reason may be the discretization that is inherent in the lifted model used in the optimization problem. Figure 17 shows the input trajectories that are obtained in experiments after one execution of the learning step with $\alpha = 0$. Comparing Fig. 8 ($\alpha = 5e^{-5}$) to Fig. 17 illustrates the positive effect of adding a penalty on the second derivative of the input (31). Smooth inputs are particularly beneficial because they increase the repeatability of the experiment, and thus the effectiveness of the learning. The system noise level is usually much higher when driving a system by fast changing inputs.

For the 2-norm optimization problem, the smallest α value still yielding smooth inputs is $\alpha = 5e^{-5}$ (cf. Table 2). For larger values, the learning performance may be cor-

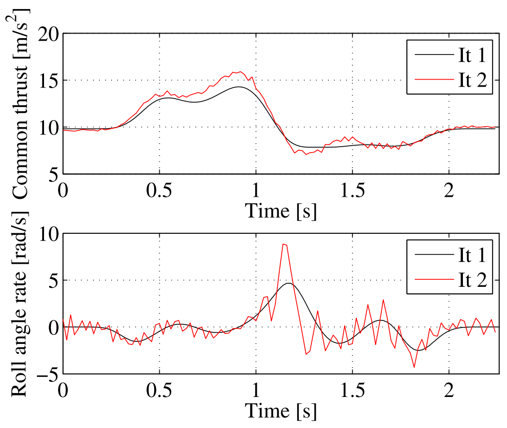

Fig. 17 *Iteration 1:* The nominal input trajectory. *Iteration 2:* Input trajectory obtained from the input update (28) after the first iteration. No input penalty term was active, i.e. $\alpha = 0$ in (28). The jittering in the inputs is significant

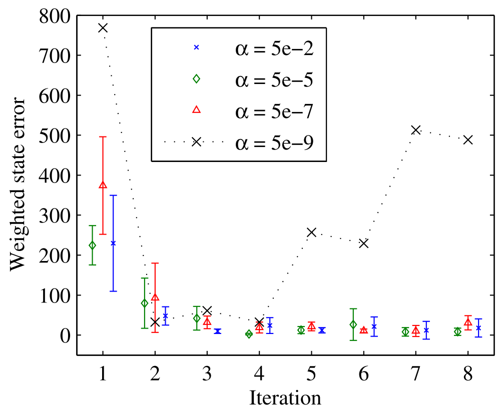

Fig. 18 Learning performance for different input penalty factors α. The error is computed according to (60). The figure shows the average error and its standard deviation obtained from five independent learning experiments. Using very small α values leads to jittering in the input which corrupts the learning performance, see *dotted line*

rupted as more emphasis is put on minimizing the input's second derivative than on learning to track the desired trajectory. Smaller values of α result in non-smooth inputs, which have a fatal effect on the learning performance. Figure 18 shows the statistic error convergence for different values of α.

Choice of input penalty matrix D The matrix D is used to penalize either the input or its approximate first or second derivative, cf. (28) and Sect. 2.3. When performing and learning the diagonal trajectory with the quadrocopter, it is possible to obtain smooth inputs with either of those choices. Interestingly, all three types of input penalty terms result in similar learning performances, as illustrated in Fig. 19. It is, however, reasonable to penalize the input's second derivative since jittering corresponds to large absolute values of the second derivative.

Choice of input update norm For the objective function (28), three different norms $\ell \in \{1, 2, \infty\}$ can be chosen. The diagonal trajectory is learned successfully by all three input update rules as illustrated in Fig. 20. In order to compare the learning performance of different norms, we used the average position error of the quadrocopter defined by (61). All three norms show fast error convergence.

Choice of noise covariances The performance of the disturbance estimation has a large influence on the learning behavior. Lying at the heart of the Kalman equations (22), the noise model (19) is characterized mainly by the process variance ϵ_j and the measurement variance η, cf. (24) and (25). Both are assumed to be constant over iterations, i.e. $\epsilon_j = \epsilon$. The ratio ϵ/η expresses the belief in the Kalman filter measurement model versus the process model (19). Since the Kalman filter uses a very simple process model and the

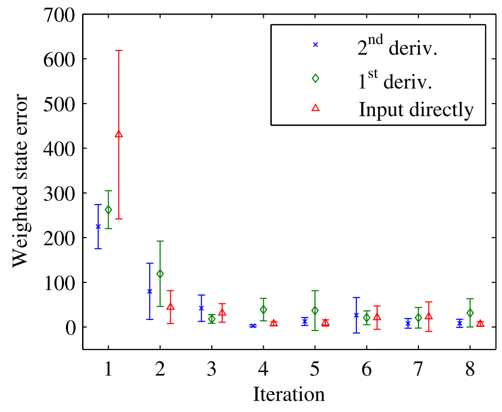

Fig. 19 Learning performance for different choices of input penalty matrices D in (28). The error is computed according to (60). For each iteration, we show the average error and its standard deviation obtained from five independent learning experiments. Penalizing the second derivative of the input results in small average and standard deviation values

motion capture system provides accurate measurement data, it is obvious to choose $\epsilon/\eta \gg 1$. The larger the ratio ϵ/η, the more emphasis is placed on the measurements. Moreover, a large value of ϵ allows the estimated disturbance to change rapidly. Therefore, a large ratio ϵ/η should result in a fast convergence of the disturbance estimate and, as a consequence, of the tracking error. As expected, the larger the ratio ϵ/η, the faster the error converges, cf. Fig. 21.

6.7 Computation times

The objective of this section is to relate the theoretic complexity analysis (Sect. 2.6) to real computation times and develop an intuition for the computational cost of the algorithm. As an example, we use the diagonal trajectory of Sect. 6.3. The trajectory comprises $N = 113$ discrete steps

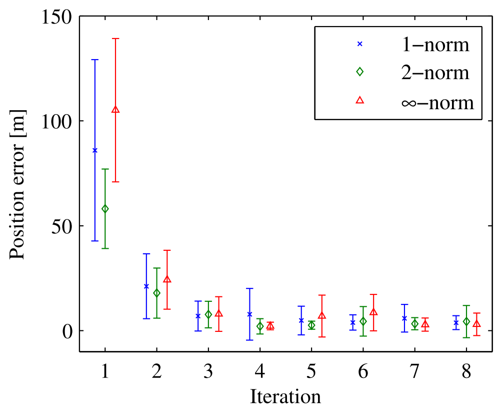

Fig. 20 Learning performance for different objective function norms ℓ. Shown is the position error as defined by (61). For each iteration, we show the average error and its standard deviation obtained from ten independent learning experiments. All norms show reasonable convergence behavior. We observe fast convergence and small standard deviations for the 1- and 2-norm with increasing number of iterations

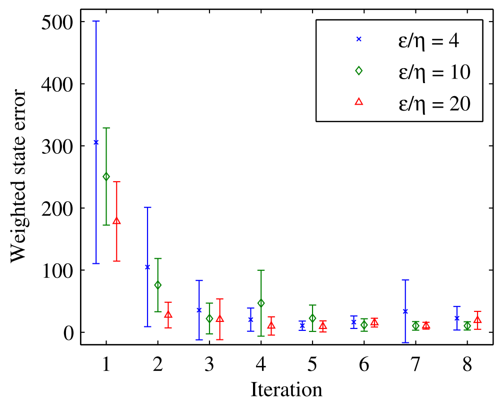

Fig. 21 Learning performance for different noise settings. The values ϵ and η are constant over iterations. The error is computed according to (60). For each iteration, we show the average error and its standard deviation obtained from five independent learning experiments. The larger the ratio ϵ/η, the faster the error convergence and the smaller the standard deviation

($t_f = 2.26$ s and $\Delta t = 0.02$ s). We recall that the number of inputs is $n_u = 2$, the number of states is $n_x = 5$, and the number of constrained quantities is $n_c = 6$, where for each constrained quantity an upper and lower bound is defined. We run the algorithm on a standard desktop PC (Windows 7, 64-bit; quad-core processor with 2.8 GHz; 4 GB RAM) for $J_{\max} = 10$ iterations and solve the optimization problem with CPLEX Version 11.2. As summarized in Table 3, the computation times are in the range of seconds, even for solving the optimization problem with hundreds of decision variables and thousands of constraints.

Table 3 Computation times in seconds for the diagonal trajectory

	1-norm	2-norm	∞-norm
Offline preparation			
Kalman gains	1.5	1.5	1.5
Online refinement			
Estimate update	0.028	0.028	0.028
Input update/Optimization	1.34	0.17	0.87
Number of decision variables	1018	226	228
Number of constraints	2938	1356	2938

7 Advantages & limitations

The experiments in the previous section demonstrated the effectiveness of the proposed learning. The tracking error was reduced to a level defined by the non-repetitive noise in the system, while any repetitive disturbances were compensated for by proper acausal input corrections. The learning algorithm proved to be robust to the choice of learning parameters, cf. Sect. 6.6. The separation of disturbance estimation and input update allowed for an intuitive interpretation and tuning of the learning algorithm parameters.

Limitations of the approach are caused by: (i) the prerequisites that must be fulfilled for applying the proposed learning approach, and (ii) the requested quality of the final tracking performance. Prerequisites of the approach are a model of the system dynamics and a method for generating feasible trajectories given the model and constraints. Refer to Sect. 2.5 for more details.

The tracking performance that can be achieved with the proposed method depends on the level of noise that corrupts the system output. Non-repetitive noise cannot be compensated for by the proposed feed-forward learning strategy and directly affects the tracking performance. A measure of the system noise level is the variance of the system output when repeatedly applying the same input. In order to increase the repeatability of the system and decrease the effect of non-repetitive noise, one may choose to introduce underlying feedback loops, cf. Bristow et al. (2006), Chin et al. (2004), Cho et al. (2005), Barton et al. (2011).

For the quadrocopter example, we defined the system inputs on the level of thrusts and rates, which implied a high level of non-repetitive noise resulting mainly from imperfect initial conditions, unpredictable motor dynamics, and environmental noise like wind gusts during flight. Experiments in the FMA have shown that the measured trajectories of a quadrocopter differ significantly, even if the same input is applied, cf. Fig. 9. The variability of the observed trajectories increases towards the end of the trajectory, which is caused by the fact that the entire trajectory is flown without feedback on position or attitude. Angle errors at the beginning of the trajectory, for instance,

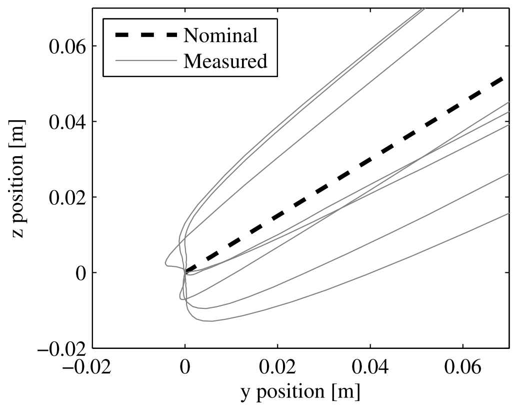

Fig. 22 Measured quadrocopter position at the start of the diagonal trajectory when not using an *AUTOSTART* mode, cf. Sect. 5.3. The entire trajectory is influenced by the initial state

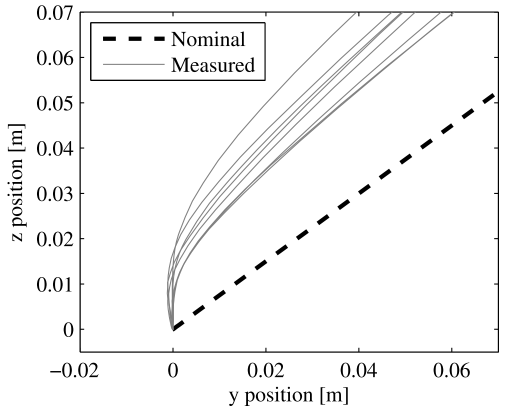

Fig. 23 Measured quadrocopter position at the start of the diagonal trajectory for more restrictive start tolerance values enforced by the *AUTOSTART* mode, cf. Fig. 5. This start procedure increases the repeatability of the system

influence all subsequent angle values and inevitably add up over an iteration. Thus, achieving an accurate initial state is crucial. To this end, we introduced start conditions (see *AUTOSTART* mode in Sect. 5.3), which allow the quadrocopter to start an iteration only if the initial state lies in the bounds (59). These start conditions significantly improve the repeatability of the system, cf. Fig. 22 and Fig. 23, and are essential for the experimental investigations in Sect. 6.

The quadrocopter experiment can be viewed as a proof-of-concept example, where thrust and roll rate serve as inputs for two reasons: (i) for the sake of simplicity (the nominal model is straight-forward to derive), and (ii) to show the limiting case where no global measurements from the cameras are used during a trajectory execution. To obtain a more versatile learning scheme for the quadrocopters and to lower the effect of non-repetitive noise on the system output, a next step may be to close feedback loops on position and attitude. The input and corresponding feed-forward corrections may then be defined on the level of position and attitude. Experiments in the FMA have shown that for this setup, the trajectory variations, when repeatedly applying the same input, lie within one to two centimeters.

In practice, we expect the proposed learning scheme to be used for quadrotor vehicles that have access to (possibly rare) position and/or attitude measurements (for example from GPS), and that use this information for feedback. As mentioned in Sect. 1.1, feed-forward adaptation is especially beneficial if feedback is available with low rate only. In this case, feedback is not able to react fast enough to allow for precise tracking but can still be used as correcting action in a feed-forward based approach. Such a setup may be used for practical applications including the ones stated in Sect. 1.1.

8 Conclusion

This paper presents an optimization-based iterative learning approach for trajectory tracking. Optimality is achieved in both the estimation of the recurring disturbance and the following input update step, which optimally compensates for the disturbance with an updated feed-forward input signal. While the first method is borrowed from classical control theory, the latter originates from mathematical optimization theory and uses a computationally efficient state-of-the art convex optimization solver. Input and state constraints are explicitly taken into account. Depending on the problem under consideration, the overall learning behavior can be controlled by changing the noise characteristics in the estimation step or the optimization objective, or by assigning different weights on different states and different parts of the trajectory.

The approach was successfully applied to quadrotor vehicles. It has been shown that trajectory tracking can be achieved by pure feed-forward adaptation of the input signal. The accuracy of the tracking is limited by the level of non-repetitive noise. In future research, the final tracking accuracy can be increased by incorporating feedback from position and/or attitude measurements (refer to Sect. 7). Furthermore, the approach is equally feasible for trajectories in three dimensions, especially when one considers that none of the calculations must be done in real time. In the 3D case, the number of inputs and constrains doubles and the number of states increases from five to nine.

Acknowledgements This research was funded in part by the Swiss National Science Foundation (SNSF). The authors thank the anonymous reviewers for their thoughtful comments.

Appendix: Input update as convex optimization problem

The update law defined in (28) can be transformed into a convex optimization problem of the form:

$$\min_z \left(\frac{1}{2} z^T V z + v^T z\right) \quad \text{subject to} \quad W z \leq w, \tag{64}$$

where z represents the vector of decision variables and V is a symmetric positive semi-definite matrix. The number of constraints is defined by the length of w. We call (64) a 'linear program' if V is zero and a 'quadratic program' otherwise, cf. Boyd and Vandenberghe (2004).

For simplicity, we consider $\alpha = 0$ in the following derivations. However, similar arguments can be made for arbitrary $\alpha > 0$. In case of norms $\|\cdot\|_\ell, \ell \in \{1, \infty\}$, which are inherently nonlinear, non-quadratic functions, (28) is re-formulated by extending the original vector of decision variables u_{j+1} and adding additional inequality constraints. Thus, in case of the 1-norm, the objective function in (28) is replaced by

$$\min_{u_{j+1}, e} \mathbb{I}^T e \quad \text{subject to} \quad -e \leq S(F u_{j+1} + \widehat{d}_j) \leq e, \tag{65}$$

where $e \in \mathbb{R}^{N n_x}$ and $\mathbb{I}$ represent a vector of ones, $\mathbb{I} = [1, 1, 1, \dots]^T \in \mathbb{R}^{N n_x}$. Similarly, for the maximum norm, the extended equation reads as

$$\min_{u_{j+1}, e} e \quad \text{subject to} \quad -e\mathbb{I} \leq S(F u_{j+1} + \widehat{d}_j) \leq e\mathbb{I} \tag{66}$$

with $e \in \mathbb{R}$. In both cases, the constraints in (28) must still be satisfied. The 2-norm results in the following objective function:

$$\min_{u_{j+1}} (F u_{j+1} + \widehat{d}_j)^T S^T S (F u_{j+1} + \widehat{d}_j). \tag{67}$$

References

Ahn, H.-S., Moore, K. L., & Chen, Y. (2007). *Communications and control engineering. Iterative learning control: robustness and monotonic convergence for interval systems* (1st ed.). Berlin: Springer.

Al-Hiddabi, S. (2009). Quadrotor control using feedback linearization with dynamic extension. In *Proceedings of the international symposium on mechatronics and its applications (ISMA)* (pp. 1–3).

Alexis, K., Nikolakopoulos, G., & Tzes, A. (2010). Constrained-control of a quadrotor helicopter for trajectory tracking under wind-gust disturbances. In *Proceedings of the IEEE Mediterranean electrotechnical conference (MELECON)* (pp. 1411–1416).

Amann, N., Owens, D. H., & Rogers, E. (1996). Iterative learning control using optimal feedback and feedforward actions. *International Journal of Control*, *65*(2), 277–293.

Anderson, B. D. O., & Moore, J. B. (2005). *Dover books on engineering. Optimal filtering*. New York: Dover.

Arimoto, S., Kawamura, S., & Miyazaki, F. (1984). Bettering operation of robots by learning. *Journal of Robotic Systems*, *1*(2), 123–140.

Bamieh, B., Pearson, J. B., Francis, B. A., & Tannenbaum, A. (1991). A lifting technique for linear periodic systems with applications to sampled-data control. *Systems & Control Letters*, *17*(2), 79–88.

Barton, K., Mishra, S., & Xargay, E. (2011). Robust iterative learning control: L1 adaptive feedback control in an ILC framework. In *Proceedings of the American control conference (ACC)* (pp. 3663–3668).

Bauer, P., Kulcsar, B., & Bokor, J. (2009). Discrete time minimax tracking control with state and disturbance estimation ii: Time-varying reference and disturbance signals. In *Proceedings of the Mediterranean conference on control and automation (MED)* (pp. 486–491).

Boost—Basic Linear Algebra Library (2012). Last accessed at 08 Feb 2012. [Online]. Available: http://www.boost.org/doc/libs/1_46_1/libs/numeric/ublas/doc/index.htm.

Bouabdallah, S., & Siegwart, R. (2005). Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (pp. 2247–2252).

Bouktir, Y., Haddad, M., & Chettibi, T. (2008). Trajectory planning for a quadrotor helicopter. In *Proceedings of the Mediterranean conference on control and automation* (pp. 1258–1263).

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge: Cambridge University Press.

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

Castillo, C. L., Moreno, W., & Valavanis, K. P. (2007). Unmanned helicopter waypoint trajectory tracking using model predictive control. In *Proceedings of the Mediterranean conference on control automation* (pp. 1–8).

Chen, Y., & Wen, C. (1999). *Lecture notes in control and information sciences. Iterative learning control: convergence, robustness and applications* (1st ed.). Berlin: Springer.

Chin, I., Qin, S. J., Lee, K. S., & Cho, M. (2004). A two-stage iterative learning control technique combined with real-time feedback for independent disturbance rejection. *Automatica*, *40*(11), 1913–1922.

Cho, M., Lee, Y., Joo, S., & Lee, K. S. (2005). Semi-empirical model-based multivariable iterative learning control of an RTP system. *IEEE Transactions on Semiconductor Manufacturing*, *18*(3), 430–439.

Chui, C. K., & Chen, G. (1998). *Springer series in information sciences. Kalman filtering: with real-time applications*. Berlin: Springer.

Cowling, I., Yakimenko, O., Whidborne, J., & Cooke, A. (2007). A prototype of an autonomous controller for a quadrotor UAV. In *Proceedings of the European control conference (ECC)* (pp. 1–8).

Curve Fitting Toolbox Splines and MATLAB Splines (2012). Last accessed 08 Feb 2012. [Online]. Available: http://www.mathworks.com/help/toolbox/curvefit/f2-10452.html.

Diao, C., Xian, B., Yin, Q., Zeng, W., Li, H., & Yang, Y. (2011). A nonlinear adaptive control approach for quadrotor UAVs. In *Proceedings of the Asian control conference (ASCC)* (pp. 223–228).

Dierks, T., & Jagannathan, S. (2010). Output feedback control of a quadrotor UAV using neural networks. *IEEE Transactions on Neural Networks*, *21*(1), 50–66.

Gurdan, D., Stumpf, J., Achtelik, M., Doth, K.-M., Hirzinger, G., & Rus, D. (2007). Energy-efficient autonomous four-rotor flying robot controlled at 1 kHz. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (pp. 361–366).

Hoffmann, G., Waslander, S., & Tomlin, C. (2008). Quadrotor helicopter trajectory tracking control. In *Proceedings of the AIAA guidance, navigation and control conference and exhibit*, Honolulu, Hawaii.

How, J. P., Bethke, B., Frank, A., Dale, D., & Vian, J. (2008). Real-time indoor autonomous vehicle test environment. *IEEE Control Systems Magazine*, *28*(2), 51–64.

Huang, H., Hoffmann, G., Waslander, S., & Tomlin, C. (2009). Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (pp. 3277–3282).

IBM ILOG CPLEX Optimizer (2012). Last accessed 08 Feb 2012. [Online]. Available: http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

Lee, K. S., Lee, J., Chin, I., Choi, J., & Lee, J. H. (2001). Control of wafer temperature uniformity in rapid thermal processing using an optimal iterative learning control technique. *Industrial & Engineering Chemistry Research*, *40*(7), 1661–1672.

Lee, D., Burg, T., Dawson, D., Shu, D., Xian, B., & Tatlicioglu, E. (2009). Robust tracking control of an underactuated quadrotor aerial-robot based on a parametric uncertain model. In *Proceedings of the IEEE international conference on systems, man and cybernetics (SMC)* (pp. 3187–3192).

Lin, S.-H., & Lian, F.-L. (2005). Study of feasible trajectory generation algorithms for control of planar mobile robots. In *Proceedings of the IEEE international conference on robotics and biomimetics (ROBIO)* (pp. 121–126).

Longman, R. W. (2000). Iterative learning control and repetitive control for engineering practice. *International Journal of Control*, *73*(10), 930–954.

Lupashin, S., & D'Andrea, R. (2011). Adaptive open-loop aerobatic maneuvers for quadrocopters. In *Proceedings of the IFAC World congress* (Vol. 18(1), pp. 2600–2606).

Lupashin, S., Schoellig, A.P., Sherback, M., & D'Andrea, R. (2010). A simple learning strategy for high-speed quadrocopter multi-flips. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (pp. 1642–1648).

Madani, T., & Benallegue, A. (2006). Backstepping control for a quadrotor helicopter. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 3255–3260).

Mellinger, D., Michael, N., & Kumar, V. (2010). Trajectory generation and control for precise aggressive maneuvers with quadrotors. In *Proceedings of the international symposium on experimental robotics*.

Michael, N., Mellinger, D., Lindsey, Q., & Kumar, V. (2010). The GRASP multiple micro UAV testbed. *IEEE Robotics & Automation Magazine*, *17*(3), 56–65.

Mokhtari, A., & Benallegue, A. (2004). Dynamic feedback controller of Euler angles and wind parameters estimation for a quadrotor unmanned aerial vehicle. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (Vol. 3, pp. 2359–2366).

Moore, K. L. (1998). Multi-loop control approach to designing iterative learning controllers. In *Proceedings of the IEEE conference on decision and control (CDC)* (Vol. 1, pp. 666–671).

Nicol, C., Macnab, C. J. B., & Ramirez-Serrano, A. (2011). Robust adaptive control of a quadrotor helicopter. *Mechatronics*, *21*(6), 927–938.

Norrloef, M., & Gunnarsson, S. (2002). Time and frequency domain convergence properties in iterative learning control. *International Journal of Control*, *75*(14), 1114–1126.

Phan, M. Q., & Longman, R. W. (1988). A mathematical theory of learning control for linear discrete multivariable systems. In *Proceedings of the AIAA/AAS astrodynamics conference* (pp. 740–746).

Piazzi, A., & Guarino Lo Bianco, C. (2000). Quintic G^2-splines for trajectory planning of autonomous vehicles. In *Proceedings of the IEEE intelligent vehicles symposium* (pp. 198–203).

Pounds, P., Mahony, R., & Corke, P. (2006). Modelling and control of a quad-rotor robot. In *Proceedings of the Australasian conference on robotics and automation*.

Purwin, O., & D'Andrea, R. (2009). Performing aggressive maneuvers using iterative learning control. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (pp. 1731–1736).

Raffo, G., Ortega, M., & Rubio, F. (2008). Backstepping/nonlinear H_∞ control for path tracking of a quadrotor unmanned aerial vehicle. In *Proceedings of the American control conference (ACC)* (pp. 3356–3361).

Rice, J. K., & Verhaegen, M. (2010). A structured matrix approach to efficient calculation of LQG repetitive learning controllers in the lifted setting. *International Journal of Control*, *83*(6), 1265–1276.

Ritz, R., Hehn, M., Lupashin, S., & D'Andrea, R. (2011). Quadrotor performance benchmarking using optimal control. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 5179–5186).

Schoellig, A. P., & D'Andrea, R. (2009). Optimization-based iterative learning control for trajectory tracking. In *Proceedings of the European control conference (ECC)* (pp. 1505–1510).

The Mathworks Optimization Toolbox (2012). Last accessed 08 Feb 2012. [Online]. Available: http://www.mathworks.com/help/toolbox/optim/ug/fmincon.html.

Tousain, R., van der Meche, E., & Bosgra, O. (2001). Design strategy for iterative learning control based on optimal control. In *Proceedings of the IEEE conference on decision and control (CDC)* (Vol. 5, pp. 4463–4468).

Waslander, S. L., Hoffmann, G. M., Jang, J. S., & Tomlin, C. (2005). Multi-agent quadrotor testbed control design: integral sliding mode vs. reinforcement learning. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)*, August 2005 (pp. 3712–3717).

Zhang, C., Wang, N., & Chen, J. (2010). Trajectory generation for aircraft based on differential flatness and spline theory. In *Proceedings of the international conference on information networking and automation (ICINA)* (Vol. 1, pp. V1-110–V1-114).

Zhou, Q.-L., Zhang, Y., Qu, Y.-H., & Rabbath, C.-A. (2010a). Dead reckoning and Kalman filter design for trajectory tracking of a quadrotor UAV. In *Proceedings of the IEEE/ASME international conference on mechatronics and embedded systems and applications (MESA)* (pp. 119–124).

Zhou, Q.-L., Zhang, Y., Rabbath, C.-A., & Theilliol, D. (2010b). Design of feedback linearization control and reconfigurable control allocation with application to a quadrotor UAV. In *Proceedings of the conference on fault-tolerant systems (SysTol)* (pp. 371–376).

Zhu, B., & Huo, W. (2010). Trajectory linearization control for a quadrotor helicopter. In *Proceedings of the IEEE international conference on control and automation (ICCA)* (pp. 34–39).

Zuo, Z. (2010). Trajectory tracking control design with command-filtered compensation for a quadrotor. *IET Control Theory & Applications*, *4*(11), 2343–2355.

Angela P. Schoellig is a Ph.D. candidate in Dynamic Systems and Control at ETH Zurich. She graduated with a M.Sc. in Engineering Science and Mechanics from the Georgia Institute of Technology in 2007 and received her Masters degree in Engineering Cybernetics from the University of Stuttgart in 2008. Her past research experience includes optimal control of hybrid systems and stability analysis of networked systems with communication delays. She has worked at the European Aeronautic Defence and Space Company (Germany), where she developed distributed estimation and control algorithms for satellites. For her graduate studies, Angela received scholarships from the German National Academic Foundation, the Cusanuswerk, and the German Academic Exchange Service. She was finalist of the 2008 IEEE Fellowship in Robotics and Automation, which supports prospective leaders in this field.

Fabian L. Mueller is a graduate student at ETH Zurich in the Master program Robotics, Systems and Control. He got his Bachelor degree in Mechanical Engineering in 2009 from ETH Zurich. During his studies, he has worked on learning algorithms, bio-inspired robotic legs and the kinematics of human hands. His main interest are learning schemes and optimization techniques.

Raffaello D'Andrea received a B.Sc. degree in Engineering Science from the University of Toronto in 1991, and M.S. and Ph.D. degrees in Electrical Engineering from the California Institute of Technology in 1992 and 1997, respectively. He held positions as assistant professor, and later, associate professor at Cornell University from 1997 to 2007. He is currently a full professor of dynamic systems and control at ETH Zurich, and technical co-founder and chief technical advisor at Kiva Systems. A creator of dynamic sculpture, his work has appeared at various international venues, including the National Gallery of Canada, the Venice Biennale, Ars Electronica, and ideaCity.