The International Journal of Advanced Manufacturing Technology
https://doi.org/10.1007/s00170-022-10789-w

ORIGINAL ARTICLE

Vibration compensation of delta 3D printer with position-varying dynamics using filtered B-splines

Nosakhare Edoimioya[1] · Cheng-Hao Chou[1] · Chinedum E. Okwudire[1]

Received: 14 September 2022 / Accepted: 28 December 2022

Abstract
The delta robot can reach higher speeds than traditional serial-axis machines used in fused filament fabrication 3D printing. However, like serial machines, delta printers suffer from undesirable vibration at high speeds which degrades the quality of fabricated parts. This undesirable vibration has been suppressed in serial printers using linear model-inversion feedforward control methods like the filtered B-splines (FBS) approach. However, techniques like the FBS approach are computationally challenging to implement on delta 3D printers because of their coupled, position-dependent dynamics. In this paper, we propose a methodology to address the computational bottlenecks by (1) parameterizing the position-dependent portions of the dynamics offline to enable efficient computation of the model online, (2) computing models at sampled points (instead of every point) along the given trajectory, and (3) employing QR factorization to reduce the number of floating-point arithmetic operations associated with matrix inversion. In simulations, we report a computation time reduction of up to 23× using the proposed method when compared to using the computationally expensive exact LPV model—all while maintaining high tracking accuracy. Accordingly, we demonstrate significant quality improvements on parts printed at various positions on a commercial delta 3D printer using our proposed controller compared to a baseline alternative, which uses an LTI model from one position. Acceleration measurements during printing show that the improvement in print quality of the proposed controller is due to vibration reductions of up to 39% when compared to the baseline controller.

Keywords Delta robot · Position-varying dynamics · Filtered B-splines · Model-based control · 3D printing

1 Introduction

Delta robots suffer from vibration errors that are a result of structural flexibilities in their kinematic chain [1] and such vibration errors can adversely impact the quality of 3D printed parts. Promisingly, model-based feedforward control techniques have been used to suppress vibration on serial-axis 3D printers [2–5], resulting in up to 2× increase in productivity without sacrificing accuracy [5]. However, the parallel-axis delta 3D printers have not yet benefited from these methods because of the difficulty modeling and controlling the delta's coupled, nonlinear dynamics. Previous work on modeling and controlling delta robots has largely focused on rotary-joint delta robots, which are actuated with servo motors [6–16]. (Most commercial delta 3D printers are prismatic-joint delta robots and typically use stepper motors). Due to the popularity of servo motor delta robots, techniques for its tracking control have been studied extensively in the literature. Although we will not cover all the literature here, we will discuss relevant examples that uncover the challenge of controlling delta 3D printers.

Most of the delta robot control literature can be grouped in one of three categories: (1) PID control, (2) adaptive/sliding-mode control, and (3) learned control. Most of the early methods proposed rely on state measurements to estimate servo errors for accurate compensation. A PD or PID controller is usually a key element of these compensation methods. However, since standalone PD/PID controllers do not consider the dynamic coupling of delta robots, their performance is affected by disturbance inputs from other kinematic chains. To address this issue, Codourey [6] combined a lumped model of the delta manipulator with a PD regulator in a computed torque (CT) control implementation to improve the tracking error

✉ Chinedum E. Okwudire
okwudire@umich.edu

[1] Department of Mechanical Engineering,
University of Michigan-Ann Arbor,
2350 Hayward Street, Ann Arbor, MI, 48104, USA

Published online: 27 January 2023

performance in pick-and-place tasks when compared with a standalone PD regulator. Similarly, Angel and Viola [7] proposed a fractional PID controller combined with a CT controller. However, CT controllers need to have complete knowledge of the robot's dynamics, which can be challenging to obtain efficiently [17], and are sensitive to uncertainties and disturbance inputs. For example, in [6], workspace accelerations, which are necessary to calculate torques, are computed as second derivatives of the direct-geometric model of the robot (i.e., functions of joint positions). These calculations can be problematic when there is noise or other inaccuracies in the measurements. Perhaps, this explains why no experiments implementing the controller on hardware are presented in [7] (only simulations).

These challenges led to the development of other techniques focused on adaptive control [8–14]. These include methods like changing the PD gains online as a function of servo error estimates [8], disturbance rejection in the feedback loop using linear disturbance observers [9, 10], injecting inputs learned by a neural network to compensate errors that the feedback controller does not reject [11, 12], and using synchronization control strategies to reject coupling disturbances in each actuator from the other actuators [13, 14]. Other similar approaches focus on tuning trajectory-dependent controller gains offline to minimize errors along a desired path that is known a priori [15, 16]. These gains provide reasonable tracking performance along the trajectory but require a priori knowledge of the entire trajectory. Some researchers have also employed sliding mode controllers to improve tracking performance since they are relatively insensitive to the disturbance inputs [18–20].

The central theme of the methods discussed above is the dependence on sensor measurements and feedback regulation to compensate for inaccurate models of the delta robot. However, most delta 3D printers cannot benefit from feedback control because they do not have sensors. Excluding expensive sensors keeps the cost of 3D printers down which encourages adoption of the technology. Hence, we must rely on accurate models that can be used in feedforward control schemes.

Recently, machine learning-based methods—mostly using artificial neural networks (ANNs)—have been employed to model and control delta robots [21–25]. ANNs are promising because of their ability to learn nonlinear behaviors and to save time in computationally challenging problems—characteristics that are useful for controlling delta 3D printers. Currently, the most promising direction is using ANNs to generate a model for parameter varying dynamics. For example, Liu and Altintas [21] trained a transfer learning model for a machine tool using two ANNs: the first using abundant data from the simulated dynamics of the machine, and the second with significantly less measurements from the machine. The models were combined to fine-tune the simulation model with the "real-world" model. Despite the success of this technique, it requires the laborious process of tuning hyperparameters during training. Additionally, since these models often lack physical interpretation, designing stable controllers for them can be challenging. In general, using ANNs to design low-level control laws can be problematic because it is difficult to guarantee their stability. Therefore, in practical applications, ANNs are typically used to generate adaptive controller gains [22–25] instead of using their models in model-based (feedback or feedforward) controllers or to directly generate low-level control laws.

Inspired by the feedforward CT controllers in [6] and [7], we recently proposed an efficient and accurate framework to obtain physics-based, linear parameter-varying (LPV) models of prismatic-joint delta robots commonly used in 3D printers [17]. The framework uses receptance coupling (see [26, 27]) to split the full model of the delta robot into sub-models that can be independently identified using empirical measurements and analytical derivations. In [17], we demonstrated that the model captures dynamic variation across different locations in the robot's task space. Accordingly, this model enables a number of feedforward model-based vibration control techniques to be applied to the delta 3D printer. Among those, there is a class of methods known as model-inversion [28] which compensate vibration errors by using the inverse of the system's dynamics to pre-filter motion commands. Unlike other feedforward control methods such as smooth command generation [29, 30] and input shaping [31], model-inversion does not introduce time delays [32] and can theoretically lead to perfect compensation [28]. In practice, perfect compensation is difficult to achieve due to unmodeled errors [2] and the prevalence of nonminimum phase zeros, which can cause oscillatory or unbounded control inputs. Nevertheless, several "approximate" model-inversion controllers have been employed in the literature [28, 33, 34]. Of the available methods (as reviewed in [28]), the filtered basis functions (FBF) approach has been shown to be versatile, compared to others, regarding its applicability to any linear system dynamics [2, 3, 32, 35–37]. The FBF approach expresses motion commands as a linear combination of basis functions, forward filters the basis functions using the system's dynamics, and optimizes the basis functions coefficients to minimize motion errors. A version of FBF commonly used for controlling manufacturing machines is the filtered B-splines (FBS) method [2–5, 35, 37], where B-splines are selected as the basis functions because they are amenable to the lengthy motion trajectories common in manufacturing. FBS is implemented in

real-time by sequentially processing small windows of the full trajectory [3].

FBS has been implemented on serial-axis 3D printers, which are modeled as linear time-invariant (LTI) systems [2–5], as well as a parallel-axis LPV 3D printer [37]. In the LTI (i.e., standard) implementation of FBS, B-splines are filtered and inverted offline to enable fast computation of their coefficients online [3]. For the LPV system in [37], the authors modeled motion errors as linear relationships between the x and y axes (and their LTI models). Furthermore, they approximate the dynamics as decoupled, resulting in independent computation of the B-spline coefficients for each axis. Using these approximations, the B-splines could also be filtered and inverted offline. However, the delta 3D printer has a coupled kinematic chain and its LPV model cannot be decoupled. Hence, its model needs to be updated at each new position, rendering real-time control with FBS computationally challenging: one must compute the new model, use it to filter the B-splines, and invert the filtered B-splines at every point along the trajectory.

Vibration compensation with FBS can theoretically improve the accuracy of delta 3D printers, but using FBS is currently impractical due to the computational challenges. Hence, this paper aims to mitigate those computational challenges through the following contributions:

1. We parameterize expressions of the delta's transfer functions offline, which leads to fast computation of the model online.
2. We select one point per window of the trajectory points to generate a model used to control all points in the window. This choice leads to faster computation and lower memory allocation, but can lead to discontinuities in control inputs. Therefore, we also propose a switching compensation method to preserve continuity in the controller's prediction of output trajectories.
3. We calculate the B-spline coefficients using QR factorization instead of pseudoinversion, leading to faster computations.

The layout of rest of the paper is as follows: Section 2 provides a review of the dynamic model developed in [17]; Section 3 gives an overview of the standard FBS approach and describes the techniques we propose to enable real-time control of delta 3D printers; Section 4 validates our proposed approach through simulations and experiments; and Section 5 concludes the paper, summarizing key insights.

2 Overview of LPV model of delta 3D printer

2.1 Description of the delta 3D printer

As depicted in Fig. 1, the three pairs of forearms on the delta 3D printer are connected to a carriage on one end, and the end-effector on the other end. Each pair of forearms are parallel and allowed to rotate freely about universal joints on both ends. The parallelogram formed by each pair of forearms guarantees that the end-effector and fixed base remain co-planar. The carriages translate vertically on linear guideways to position one end of the forearms, which leads to the other end of the forearms translating the end-effector. Hence, the relative vertical position of each carriage (i.e., the joint space) determines the Cartesian position of the 3-DOF end-effector (i.e., the task space), which holds a nozzle that deposits melted filament onto a stationary bed. For

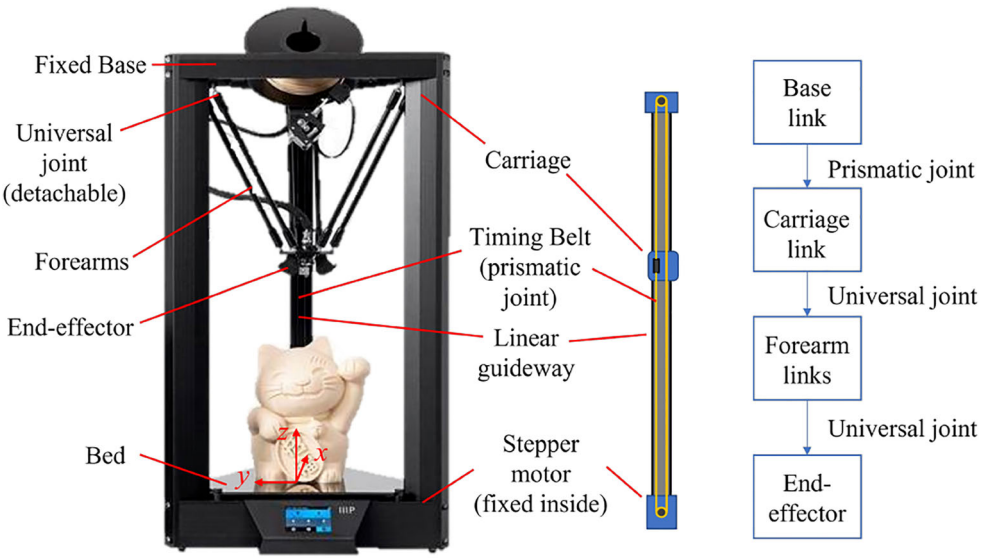

Fig. 1 From left to right: A commercial delta 3D Printer (Monoprice Delta Pro) with labeled components, a schematic of the belt-driven carriage system, and the delta manipulator configuration showing the connections between joints and links. The print volume dimensions are 270 × 270 × 300 mm

actuation, each carriage is mounted to a timing belt, which is, in turn, connected to a base-mounted stepper motor via a motor pulley—forming the prismatic joint.

2.2 Linear parameter-varying model

The carriage output positions of the delta 3D printer, q_i, are a function of two inputs: (a) the commanded position of each carriage q_{d_i} and (b) the forces F_{q_i} imposed on each carriage due to the dynamics of the forearms and end-effector, where $i \in \{A, B, C\}$ denotes the carriages labeled A, B, and C (see Fig. 2). Formally, the carriage output dynamics are given by

$$q_i(s) = G_{q_d}(s) q_{d_i}(s) + G_{Fq}(s) F_{q_i}(\mathbf{X}, s) \tag{1}$$

where s is the Laplace variable, $G_{q_d}(s)$ and $G_{Fq}(s)$ are LTI systems representing the carriage position to position transfer function (TF) and the external force to carriage position TF, respectively, and $\mathbf{X} = [x \quad y \quad z]^T$ is the end-effector's position in the task space coordinates.

The coefficients of $G_{q_d}(s)$ and $G_{Fq}(s)$ can be identified from measurements on the printer. Additionally, an analytical model of $F_{q_i}(\mathbf{X}, s)$ is obtained by considering the inertial dynamics of the end-effector in the task space, which can be transformed into joint space dynamics using the Jacobian transpose matrix. The parameters of $F_{q_i}(\mathbf{X}, s)$ (e.g., inertia, stiffness, etc.) are also identified from measurements and least squares estimation techniques. More details of this process are given in [17], which also describes how the formulation transforms Eq. 1 into the following expression:

$$\mathbf{q}(s) = \mathbf{G}_{q_d}(s)\mathbf{q}_d(s) + \mathbf{G}_{Fq}(s) \begin{bmatrix} \bar{\mathbf{J}}_A^T \mathbf{P}_A \\ \bar{\mathbf{J}}_B^T \mathbf{P}_B \\ \bar{\mathbf{J}}_C^T \mathbf{P}_C \end{bmatrix} \mathbf{W}(s) \bar{\mathbf{J}} \mathbf{q}(s) \tag{2}$$

where $\mathbf{G}_{q_d}(s)$ and $\mathbf{G}_{Fq}(s)$ are 3×3 diagonal matrices that contain $G_{q_d}(s)$ and $G_{Fq}(s)$ as the diagonal entries, respectively, $\mathbf{q} = [q_A \quad q_B \quad q_C]^T$ is the carriage output position vector, $\mathbf{q}_d = [q_{d_A} \quad q_{d_B} \quad q_{d_C}]^T$ is the desired position vector, $\mathbf{P}_i \in \mathbb{R}^{3\times 3}$ is a matrix representing the distribution of task space inertial forces associated with carriage i, $\bar{\mathbf{J}}_i \in \mathbb{R}^{3\times 1}$ is the column vector for carriage i extracted from the linearized Jacobian matrix, denoted by $\bar{\mathbf{J}}$ (see [17] for details),

$$\mathbf{W}(s) = \begin{bmatrix} w_x(s) & 0 & 0 \\ 0 & w_y(s) & 0 \\ 0 & 0 & w_z(s) \end{bmatrix}, \tag{3}$$

and $w_x(s)$, $w_y(s)$, and $w_z(s)$ are the flexible inertial dynamics of the end-effector in the x-, y-, and z-axes directions, respectively. The model in Eq. 2 can be expressed simply as

$$\mathbf{q}(s) = \mathbf{G}(s)\mathbf{q}_d(s) \tag{4}$$

where

$$\mathbf{G}(s) = \left[\mathbf{I} - \mathbf{G}_{Fq}(s) \begin{bmatrix} \bar{\mathbf{J}}_A^T \mathbf{P}_A \\ \bar{\mathbf{J}}_B^T \mathbf{P}_B \\ \bar{\mathbf{J}}_C^T \mathbf{P}_C \end{bmatrix} \mathbf{W}(s) \bar{\mathbf{J}}\right]^{-1} \mathbf{G}_{q_d}(s), \tag{5}$$

yielding a linear parameter-varying (LPV) model of the delta 3D printer that can be used for linear model-inversion feedforward control. Note that since there are no position sensors, we assume the parameters of $\mathbf{G}(s)$ can be computed using the desired configuration instead of the output configuration, e.g., $\mathbf{X}_d$ instead of $\mathbf{X}$, which was a reasonable assumption in prior work [37].

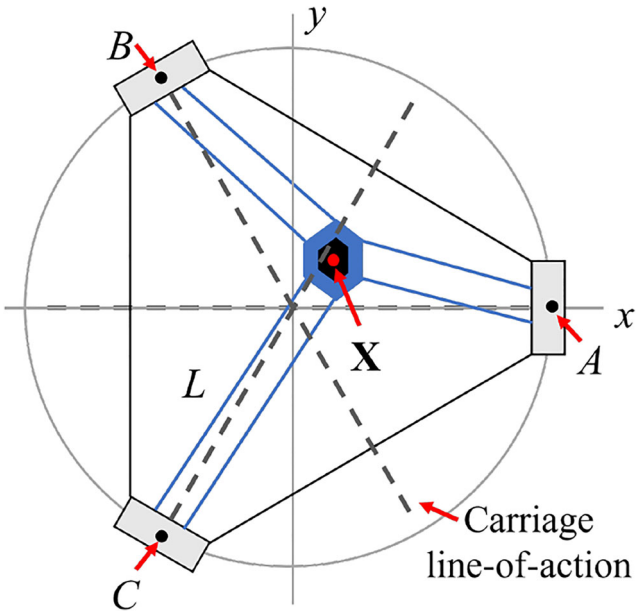

Fig. 2 Overhead view of the delta 3D printer showing the (x, y)-coordinate locations of carriages A, B, and C, the end-effector's position in task space $\mathbf{X}$, and the length of the forearms L. End-effector motion along a carriage's line-of-action results in significant change to the carriage dynamics

3 Feedforward control with filtered B-splines

3.1 Overview of the standard FBS approach

The FBS approach (as presented in [3]) controls the lifted system representation (LSR) of $\mathbf{G}(s)$ with a feedforward controller (see Appendix for details on the LSR). Let $\mathbf{q}_{i_d} = [q_{i_d}(t_0) \quad q_{i_d}(t_1) \quad \cdots \quad q_{i_d}(t_E)]^T$ represent the entire $E + 1$ discrete time steps of the desired trajectory of carriage i, which are processed in sliding windows. Assume that time t_k marks the beginning of the current window and that the unknown modified motion command,

$\mathbf{q}_{i_{dm},\mathrm{C}} = [q_{i_{dm}}(t_k) q_{i_{dm}}(t_{k+1}) \cdots q_{i_{dm}}(t_{k+L_C})]^T$, is parameterized using B-splines such that

$$\begin{bmatrix} q_{i_{dm}}(t_k) \\ q_{i_{dm}}(t_{k+1}) \\ \vdots \\ q_{i_{dm}}(t_{k+L_C}) \end{bmatrix} = \underbrace{\begin{bmatrix} \phi_{m,m}(t_k) & \cdots & \phi_{m+n,m}(t_k) \\ \phi_{m,m}(t_{k+1}) & \cdots & \phi_{m+n,m}(t_{k+1}) \\ \vdots & \ddots & \vdots \\ \phi_{m,m}(t_{k+L_C}) & \cdots & \phi_{m+n,m}(t_{k+L_C}) \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} p_{i,m} \\ \vdots \\ p_{i,m+n} \end{bmatrix}}_{\mathbf{p}_{i,\mathrm{C}}} \tag{6}$$

where the (non-italicized) subscript C denotes the current window, L_C is the number of trajectory points considered for each window, Φ is the open-ended B-spline basis functions matrix of degree m [3], $\phi_{j,m}(t)$ are real-valued basis functions [38], $j = m, m+1, ..., m+n$, $\mathbf{p}_{i,\mathrm{C}}$ is a vector of $n+1$ unknown coefficients (or control points), $t_k = kT_s$ is the current time, and T_s is the sampling time (see [3] for more details). To capture the coupling between carriages, we define $\mathbf{q}_{d,\mathrm{C}} = [\mathbf{q}_{A_d,\mathrm{C}}^T \mathbf{q}_{B_d,\mathrm{C}}^T \mathbf{q}_{C_d,\mathrm{C}}^T]^T$, such that

$$\mathbf{q}_{dm,\mathrm{C}} = \begin{bmatrix} \mathbf{q}_{A_{dm},\mathrm{C}} \\ \mathbf{q}_{B_{dm},\mathrm{C}} \\ \mathbf{q}_{C_{dm},\mathrm{C}} \end{bmatrix} = \underbrace{\begin{bmatrix} \Phi & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \Phi & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \Phi \end{bmatrix}}_{\mathbf{N}_\mathrm{C}} \underbrace{\begin{bmatrix} \mathbf{p}_{A,\mathrm{C}} \\ \mathbf{p}_{B,\mathrm{C}} \\ \mathbf{p}_{C,\mathrm{C}} \end{bmatrix}}_{\mathbf{p}_\mathrm{C}} \tag{7}$$

Our aim is to minimize the tracking error which is defined as

$$\begin{aligned} &\bar{\mathbf{e}} = \mathbf{q}_d - \mathbf{q} = \mathbf{q}_d - \bar{\mathbf{N}}\bar{\mathbf{p}} \Leftrightarrow \\ &\begin{bmatrix} \bar{\mathbf{e}}_\mathrm{P} \\ \bar{\mathbf{e}}_\mathrm{C} \\ \bar{\mathbf{e}}_\mathrm{F} \end{bmatrix} = \begin{bmatrix} \mathbf{q}_{d,\mathrm{P}} \\ \mathbf{q}_{d,\mathrm{C}} \\ \mathbf{q}_{d,\mathrm{F}} \end{bmatrix} - \begin{bmatrix} \bar{\mathbf{N}}_\mathrm{P} & \mathbf{0} & \mathbf{0} \\ \bar{\mathbf{N}}_\mathrm{PC} & \bar{\mathbf{N}}_\mathrm{C} & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{N}}_\mathrm{CF} & \bar{\mathbf{N}}_\mathrm{F} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{p}}_\mathrm{P} \\ \bar{\mathbf{p}}_\mathrm{C} \\ \bar{\mathbf{p}}_\mathrm{F} \end{bmatrix} \end{aligned} \tag{8}$$

where subscripts P and F denote the past and future windows, respectively, and the bar on the matrices and vectors indicates that the impulse response of the transfer function used for filtering the B-splines is truncated [3]. Note that the current output carriage motion is given by

$$\mathbf{q}_\mathrm{C} = \bar{\mathbf{N}}_\mathrm{C}\bar{\mathbf{p}}_\mathrm{C} + \bar{\mathbf{N}}_\mathrm{PC}\bar{\mathbf{p}}_\mathrm{P} \tag{9}$$

Using local least squares, the optimal coefficients of the current window can be computed as

$$\bar{\mathbf{p}}_\mathrm{C} = (\bar{\mathbf{N}}_\mathrm{C}^T\bar{\mathbf{N}}_\mathrm{C})^{-1}\bar{\mathbf{N}}_\mathrm{C}\left(\mathbf{q}_{d,\mathrm{C}} - \bar{\mathbf{N}}_\mathrm{PC}\bar{\mathbf{p}}_\mathrm{P}\right) \tag{10}$$

$$= \bar{\mathbf{N}}_\mathrm{C}^\dagger\left(\mathbf{q}_{d,\mathrm{C}} - \bar{\mathbf{N}}_\mathrm{PC}\bar{\mathbf{p}}_\mathrm{P}\right) \tag{11}$$

where $\bar{\mathbf{p}}_\mathrm{P}$ denotes the coefficients calculated in the previous window. The windows are designed to overlap such that n coefficents are computed in each window but only $n_{\mathrm{up}} < n$ are updated [3].

For an LTI system, $\mathbf{N}_\mathrm{C}$ is pre-filtered and $\bar{\mathbf{N}}_\mathrm{PC}$ and $\bar{\mathbf{N}}_\mathrm{C}^\dagger$ are computed offline and stored for calculating the optimal coefficients in every window using Eq. 11 (see [3] for details on the offline computation). However, for complex LPV systems like the delta, filtering and inverting the (large) B-splines matrix must be done online, which is computationally challenging to do at a fast enough rate for real-time requirements on many hardware processors. The rest of this section proposes techniques to optimize the computation and memory resources required to apply FBS to the delta 3D printer without significantly sacrificing the improved accuracy performance when constrained by the system's computation and memory capabilities.

3.2 Selecting a parameterized model for B-splines filtering

Consider the problem of filtering each column of $\mathbf{N}$ through $\mathbf{G}(s)$, reproduced below:

$$\mathbf{G}(s) = \underbrace{\left[\mathbf{I} - \mathbf{G}_{Fq}(s)\begin{bmatrix} \bar{\mathbf{J}}_A^T\mathbf{P}_A \\ \bar{\mathbf{J}}_B^T\mathbf{P}_B \\ \bar{\mathbf{J}}_C^T\mathbf{P}_C \end{bmatrix}\mathbf{W}(s)\bar{\mathbf{J}}\right]^{-1}}_{\mathbf{G}_J^{-1}(s)} \mathbf{G}_{q_d}(s). \tag{12}$$

Note that $\mathbf{G}_J^{-1}(s) \in \mathbb{R}^{3\times 3}$ depends on the configuration through the Jacobian matrix $\bar{\mathbf{J}}$, while $\mathbf{G}_{q_d}(s)$ is not position dependent. Hence, we can derive symbolic expressions of each transfer function in $\mathbf{G}_J^{-1}(s)$ as functions of position. This derivation leads to symbolic transfer functions of the form

$$G_{J,AA}^{-1}(s) = \frac{b_{AA}(x, y, z, q_A, q_B, q_C)}{a(x, y, z, q_A, q_B, q_C, s)} \tag{13}$$

where $b_{AA}(\cdot)$ and $a(\cdot)$ are the numerator and denominator of the transfer function, respectively, and the subscript "AA" denotes values pertaining to the A-to-A carriage position dynamics. The other 8 transfer functions ($G_{J,BA}^{-1}$, $G_{J,CA}^{-1}$, $G_{J,AB}^{-1}(s)$, and so on) can be expressed similarly with $b_{BA}(\cdot)$, $b_{CA}(\cdot)$, $b_{AB}(\cdot)$, and so on since all transfer functions share the same denominator $a(\cdot)$. These parameterized transfer functions enable fast computations of the coefficients of $\mathbf{G}_J^{-1}(s)$ during real-time control by simply substituting the corresponding values of x, y, z, q_A, q_B, and q_C into the symbolic expressions. Furthermore, we can prefilter $\mathbf{N}$ with $\mathbf{G}_{q_d}(s)$ offline to obtain $\bar{\mathbf{N}}_{q_d}$. Then, for each window of trajectory points processed, we filter $\bar{\mathbf{N}}_{q_d}$ with the transfer functions in Eq. 13 to obtain

$$\begin{bmatrix} \bar{\mathbf{N}}_\mathrm{C} \\ \bar{\mathbf{N}}_\mathrm{CF} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \bar{\mathbf{N}}_{\mathrm{C}_{AA}} \\ \bar{\mathbf{N}}_{\mathrm{CF}_{AA}} \end{bmatrix} & \begin{bmatrix} \bar{\mathbf{N}}_{\mathrm{C}_{BA}} \\ \bar{\mathbf{N}}_{\mathrm{CF}_{BA}} \end{bmatrix} & \begin{bmatrix} \bar{\mathbf{N}}_{\mathrm{C}_{CA}} \\ \bar{\mathbf{N}}_{\mathrm{CF}_{CA}} \end{bmatrix} \\ \begin{bmatrix} \bar{\mathbf{N}}_{\mathrm{C}_{AB}} \\ \bar{\mathbf{N}}_{\mathrm{CF}_{AB}} \end{bmatrix} & \begin{bmatrix} \bar{\mathbf{N}}_{\mathrm{C}_{BB}} \\ \bar{\mathbf{N}}_{\mathrm{CF}_{BB}} \end{bmatrix} & \begin{bmatrix} \bar{\mathbf{N}}_{\mathrm{C}_{CB}} \\ \bar{\mathbf{N}}_{\mathrm{CF}_{CB}} \end{bmatrix} \\ \begin{bmatrix} \bar{\mathbf{N}}_{\mathrm{C}_{AC}} \\ \bar{\mathbf{N}}_{\mathrm{CF}_{AC}} \end{bmatrix} & \begin{bmatrix} \bar{\mathbf{N}}_{\mathrm{C}_{BC}} \\ \bar{\mathbf{N}}_{\mathrm{CF}_{BC}} \end{bmatrix} & \begin{bmatrix} \bar{\mathbf{N}}_{\mathrm{C}_{CC}} \\ \bar{\mathbf{N}}_{\mathrm{CF}_{CC}} \end{bmatrix} \end{bmatrix} \tag{14}$$

where $[\bar{\mathbf{N}}_{\mathrm{C}_{AA}}^T \quad \bar{\mathbf{N}}_{\mathrm{CF}_{AA}}^T]^T$ is the result of filtering the columns of $\bar{\mathbf{N}}_{q_d}$ through $G_{J,AA}^{-1}$, and so on for the other blocks of the matrix.

In practice, the current and future windows are overlapped for continuity during computation, but only L_C points are updated during each sequence [3]. Each overlapped window has $2L_C$ trajectory points, meaning that the time complexity for computing the transfer function coefficients is $O(2L_C)$ (assuming parallel computation) and the space complexity is $O(2L_c u_a)$, where u_a is the order of the transfer functions. Some computers may not have enough processing power to complete these calculations while maintaining real-time printing—especially the smaller micro-processors commonly used by 3D printer manufacturers. Additionally, allocating the memory resources required to store the coefficients may limit the computer's ability to allocate quick-access memory to other important functions like storing the print trajectory.

To prevent such deleterious effects, we select one of the first L_C points from each window at which a time-invariant transfer function $G^{-1}_{J,(\cdot,\cdot)}(s)$ is computed, which reduces the time and space complexity to $O(1)$ and $O(u_a)$, respectively. This trade-off is reasonable because: (a) only the first L_C points will be commanded, and (b) L_C generally represents a small distance where the dynamics do not change significantly. For example, L_C typically ranges from 100 to 200 points, which represents 100 to 200 ms for a standard sampling interval of 1 ms. For most practical applications, the 3D printer will not cover large enough distances in $\leq$200 ms to create significant dynamic variation.

For our implementation of the single point selection described above, we select the median point (i.e., the point in the middle of the window) as the representative point. One can select other points such as the mean point (i.e., the average point in the window for each configuration variable, considered independently) or the point with the minimum total Euclidean distance from all the other points in the same window. In simulations, we found that the tracking accuracy is not significantly different when any reasonable central point is selected. In Section 4, we demonstrate that selecting one point in each window does not significantly degrade the accuracy of the controller through simulations and experiments, especially when the technique for smooth switching is added as discussed in the following subsection.

3.3 Smoothly switching models between windows

One drawback to selecting a different model for each window is that switching models can lead to discontinuities in the feedforward controller's predicted output trajectories. In the standard FBS approach, continuity of the predicted trajectories is preserved by using the same LTI model for every window [3].

To demonstrate what happens in the LPV case, suppose we used model 1 for the past window and update it to model 2 for the current window as shown in Fig. 3. When we switch from model 1 to model 2, the prediction of the output trajectory in the current window will be different depending on if we use model 1 (i.e., $\bar{\mathbf{N}}_{1,\mathrm{PC}}\bar{\mathbf{p}}_{\mathrm{P}}$ as the prediction) or model 2 (i.e., $\bar{\mathbf{N}}_{2,\mathrm{PC}}\bar{\mathbf{p}}_{\mathrm{P}}$ as the prediction). Since model 2 captures the dynamics in the current window more accurately than model 1, $\bar{\mathbf{N}}_{2,\mathrm{PC}}$ should be used for the prediction. However, using $\bar{\mathbf{N}}_{2,\mathrm{PC}}$ may result in a discontinuity in the prediction of the machine's motion at the point where the window changes because the

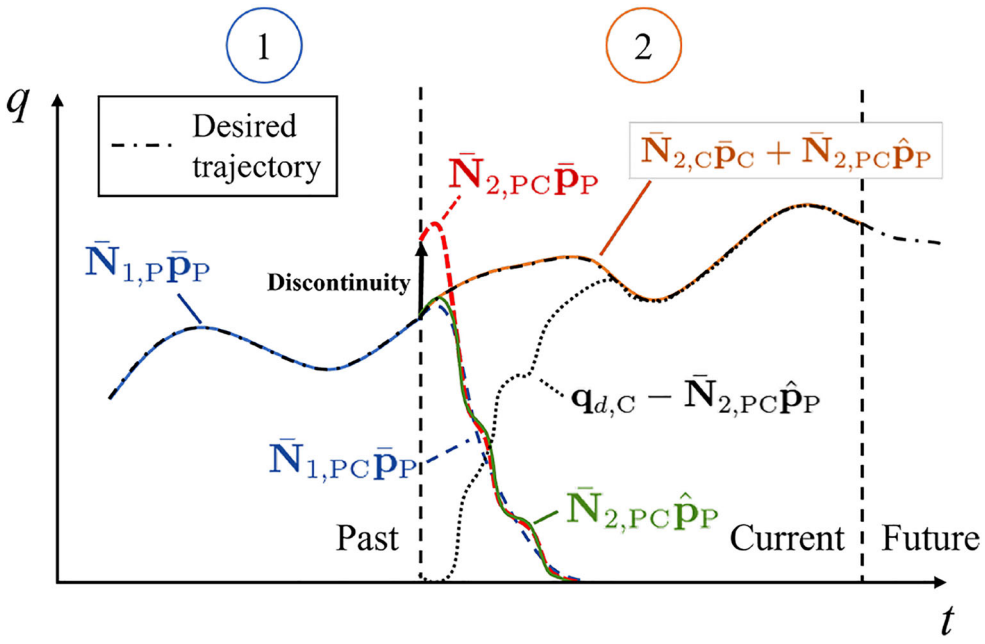

Fig. 3 Illustration of the switching compensation technique to maintain continuity described in Section 3.3. The B-spline coefficients (or control points) from the previous window $\bar{\mathbf{p}}_{\mathrm{P}}$ are approximated as $\hat{\mathbf{p}}_{\mathrm{P}}$ to maintain continuity when the model is switched from model 1 to model 2. Note that $\bar{\mathbf{N}}_{1,\mathrm{PC}}\bar{\mathbf{p}}_{\mathrm{P}}$ does not have the correct dynamics for the current window and $\bar{\mathbf{N}}_{2,\mathrm{PC}}\bar{\mathbf{p}}_{\mathrm{P}}$ creates a discontinuity at the window boundary. The difference between the desired trajectory and the approximate residual motion is also shown as $\mathbf{q}_{d,\mathrm{C}} - \bar{\mathbf{N}}_{\mathrm{PC}}\hat{\mathbf{p}}_{\mathrm{P}}$

previous window's control points, $\bar{\mathbf{p}}_{\mathrm{P}}$, were computed using model 1.

To resolve this discrepancy, we compute an approximate prediction by generating a set of approximate control points that ensure continuity with the output from the past window. The approximate control points are selected to minimize the difference between the new prediction and the original prediction while preserving continuity. We write the optimization problem as

$$\begin{aligned} \hat{\mathbf{p}}_{\mathrm{P}} = \arg\min_{\bar{\mathbf{p}}_{\mathrm{P}}} \quad & \|\bar{\mathbf{N}}_{2,\mathrm{PC}}\hat{\mathbf{p}}_{\mathrm{P}} - \bar{\mathbf{N}}_{2,\mathrm{PC}}\bar{\mathbf{p}}_{\mathrm{P}}\|_2^2 \\ s.t. \quad & \bar{N}^T_{2,\mathrm{PC}}(t_k)\hat{\mathbf{p}}_{\mathrm{P}} = \bar{N}^T_{1,\mathrm{PC}}(t_k)\bar{\mathbf{p}}_{\mathrm{P}} \\ & \bar{N'}^T_{2,\mathrm{PC}}(t_k)\hat{\mathbf{p}}_{\mathrm{P}} = \bar{N'}^T_{1,\mathrm{PC}}(t_k)\bar{\mathbf{p}}_{\mathrm{P}} \end{aligned} \tag{15}$$

where $\bar{N}^T_{1,\mathrm{PC}}(t_k)$ and $\bar{N}^T_{2,\mathrm{PC}}(t_k)$ are the first rows of $\bar{\mathbf{N}}_{1,\mathrm{PC}}$ and $\bar{\mathbf{N}}_{2,\mathrm{PC}}$ in the window, respectively, $\bar{N'}^T_{1,\mathrm{PC}}(t_k)$ and $\bar{N'}^T_{2,\mathrm{PC}}(t_k)$ are the first rows of $\bar{\mathbf{N}}'_{1,\mathrm{PC}}$ and $\bar{\mathbf{N}}'_{2,\mathrm{PC}}$, respectively (which are the time derivatives of $\bar{\mathbf{N}}_{1,\mathrm{PC}}$ and $\bar{\mathbf{N}}_{2,\mathrm{PC}}$), and $\hat{\mathbf{p}}_{\mathrm{P}}$ is the approximate control points. Note that the products

$$\bar{N}^T_{1,\mathrm{PC}}(t_k)\bar{\mathbf{p}}_{\mathrm{P}} \quad \text{and} \quad \bar{N}^T_{2,\mathrm{PC}}(t_k)\hat{\mathbf{p}}_{\mathrm{P}} \tag{16}$$

represent positions at the window boundary, and

$$\bar{N'}^T_{1,\mathrm{PC}}(t_k)\bar{\mathbf{p}}_{\mathrm{P}} \quad \text{and} \quad \bar{N'}^T_{2,\mathrm{PC}}(t_k)\hat{\mathbf{p}}_{\mathrm{P}} \tag{17}$$

and represent velocities at the boundary. Additional kinematic constraints, such as acceleration and jerk, can be included in the optimization problem from Eq. 15 by taking additional derivatives of the B-splines as described in [38] and [39]. More kinematic constraints leads to smoother transitions when the dynamics change significantly or when the window length is long. In our simulations of the machine used in Section 4, we found that position and velocity constraints led to similar tracking accuracy when compared to optimizing Eq. 15 with acceleration and jerk constraints. Hence, our implementation only uses the position and velocity constraints for Eq. 15.

Using the approximate control points, the coefficients that minimize the tracking error in the current window are obtained by solving

$$\bar{\mathbf{p}}_{\mathrm{C}} = \arg\min_{\bar{\mathbf{p}}_{\mathrm{C}}} \Big[\Big((\mathbf{q}_{d,\mathrm{C}} - \bar{\mathbf{N}}_{\mathrm{PC}}\hat{\mathbf{p}}_{\mathrm{P}}) - \bar{\mathbf{N}}_{\mathrm{C}}\bar{\mathbf{p}}_{\mathrm{C}} \Big)^T \Big((\mathbf{q}_{d,\mathrm{C}} - \bar{\mathbf{N}}_{\mathrm{PC}}\hat{\mathbf{p}}_{\mathrm{P}}) - \bar{\mathbf{N}}_{\mathrm{C}}\bar{\mathbf{p}}_{\mathrm{C}} \Big) \Big]. \tag{18}$$

Solving the constrained optimization problem in Eq. 15 in real-time could be challenging. Similarly, we can speed up the computation of Eq. 18 by using a least squares optimization method that is faster than the pseudoinverse. To ensure fast computations, we employ the LU and QR factorization methods for solving Eqs. 15 and 18, respectively, as discussed in the following subsection.

3.4 Command generation with LU and QR factorization

The optimization problem in Eq. 15 can be solved with a number of gradient-based algorithms. For example, MATLAB provides functions *fmincon* and *lsqlin* to solve constrained optimization problems. However, such algorithms may require a large number of iterations to converge to a solution, which can stall our controller. To circumvent this problem, we can solve the constrained least squares problem with LU factorization by leveraging properties of the filtered B-splines. To simplify notation, we define the following from Eq. 15:

$$\mathbf{A} = \bar{\mathbf{N}}_{2,\mathrm{PC}}, \quad \mathbf{b} = \bar{\mathbf{N}}_{2,\mathrm{PC}}\bar{\mathbf{p}}_{\mathrm{P}} \tag{19}$$

$$\mathbf{C} = \begin{bmatrix} \bar{N}^T_{2,\mathrm{PC}}(t_k) \\ \bar{N'}^T_{2,\mathrm{PC}}(t_k) \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} \bar{N}^T_{1,\mathrm{PC}}(t_k)\bar{\mathbf{p}}_{\mathrm{P}} \\ \bar{N'}^T_{1,\mathrm{PC}}(t_k)\bar{\mathbf{p}}_{\mathrm{P}} \end{bmatrix} \tag{20}$$

Then, the problem can be written as

$$\begin{aligned} \hat{\mathbf{p}}_{\mathrm{P}} = \arg\min_{\hat{\mathbf{p}}_{\mathrm{P}}} \quad & \|\mathbf{A}\hat{\mathbf{p}}_{\mathrm{P}} - \mathbf{b}\|_2^2 \\ s.t. \quad & \mathbf{C}\hat{\mathbf{p}}_{\mathrm{P}} = \mathbf{d} \end{aligned} \tag{21}$$

We make two assumptions:

1. The stacked matrix
$$\begin{bmatrix} \mathbf{A} \\ \mathbf{C} \end{bmatrix} \tag{22}$$
has linearly independent columns; and
2. $\mathbf{C}$ has linearly independent rows.

As discussed in [32], the filtered B-splines satisfy the above assumptions with high probability and, in the case they do not, the B-splines can be freely selected by the user to satisfy the assumptions. Then, we can construct the Lagrangian,

$$\mathscr{L}(\hat{\mathbf{p}}_{\mathrm{P}}, \lambda) \triangleq \frac{1}{2}\|\mathbf{A}\hat{\mathbf{p}}_{\mathrm{P}} - \mathbf{b}\|_2^2 + \lambda^T(\mathbf{C}\hat{\mathbf{p}}_{\mathrm{P}} - \mathbf{d}), \tag{23}$$

where λ is a set of Lagrange multipliers, and find where its partial derivatives equal zero to obtain the following linear system

$$\begin{bmatrix} \mathbf{A}^T\mathbf{A} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{p}}_{\mathrm{P}} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{A}^T\mathbf{b} \\ \mathbf{d} \end{bmatrix}. \tag{24}$$

Note that the matrix

$$\begin{bmatrix} \mathbf{A}^T\mathbf{A} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \tag{25}$$

is nonsingular when the above assumptions hold. Therefore, the linear equation given by Eq. 24 can be efficiently solved with LU factorization [40].

We also use QR factorization to efficiently compute the control points. Using the pseudoinverse to solve the optimization problem in Eq. 18 accumulates the following

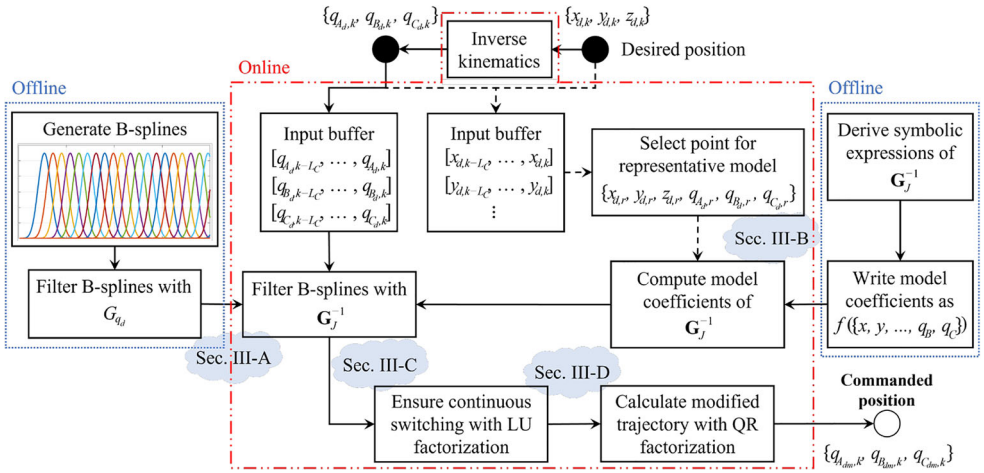

Fig. 4 Flowchart of FBS implementation on delta 3D printer. First, the B-splines are generated offline and filtered with the carriage-only position-to-position dynamics. G_{q_d} as described Section 3.1 (left side). Online, L_C points from the desired Cartesian and joint coordinates are buffered for a new window and a representative configuration is selected from the buffer—the median configuration is used in this paper. Then, the representative configuration $\{x_{d,r}, ...q_{C_d,r}\}$ is used to compute the transfer function model coefficients of $\mathbf{G}_J^{-1}$ (see Section 3.2). This transfer function is then used to filter the offline B-splines. Finally, we compute the approximate B-spline coefficients from the previous window to maintain continuity during switching (Section 3.3) and calculate the modified trajectory (Section 3.4)

number of floating-point operations (flops) for each step [40]:

$$\bar{\mathbf{N}}_{\mathrm{C}}^T\bar{\mathbf{N}}_{\mathrm{C}} \;:\; L_C n^2 \;\text{ flops} \tag{26}$$

$$(\bar{\mathbf{N}}_{\mathrm{C}}^T\bar{\mathbf{N}}_{\mathrm{C}})^{-1} \;:\; n^3 + L_C n^2 \;\text{ flops} \tag{27}$$

$$\bar{\mathbf{N}}_{\mathrm{C}}\tilde{\mathbf{q}}_{d,\mathrm{C}} \;:\; n^3 + L_C n^2 + 2L_C n \;\text{ flops} \tag{28}$$

$$(\bar{\mathbf{N}}_{\mathrm{C}}^T\bar{\mathbf{N}}_{\mathrm{C}})^{-1}\left(\bar{\mathbf{N}}_{\mathrm{C}}\tilde{\mathbf{q}}_{d,\mathrm{C}}\right) \;:\; n^3 + L_C n^2 + 4L_C n \;\text{ flops.} \tag{29}$$

where $\tilde{\mathbf{q}}_{d,\mathrm{C}} = \mathbf{q}_{d,\mathrm{C}} - \bar{\mathbf{N}}_{\mathrm{PC}}\hat{\mathbf{p}}_{\mathrm{P}}$. By factoring

$$\bar{\mathbf{N}}_{\mathrm{C}} = \mathbf{QR} \tag{30}$$

with the modified Gram Schmidt algorithm [41], where $\mathbf{Q} \in \mathbb{R}^{L_C \times L_C}$ is an orthogonal matrix (i.e., $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$) and $\mathbf{R} \in \mathbb{R}^{L_C \times n}$ is an upper triangular matrix, the problem in Eq. 18 can be written as

$$\mathbf{R}\bar{\mathbf{p}}_{c,\mathrm{C}} = \mathbf{Q}^T\tilde{\mathbf{q}}_{d,\mathrm{C}} \tag{31}$$

which can be solved using backward substitution. The number of operations accumulated using this method are

$$\bar{\mathbf{N}}_{\mathrm{C}} = \mathbf{QR} \;:\; L_C n^2 \;\text{ flops} \tag{32}$$

$$\mathbf{w} = \mathbf{Q}^T\tilde{\mathbf{q}}_{d,\mathrm{C}} \;:\; L_C n^2 + 2L_C n \;\text{ flops} \tag{33}$$

$$\mathbf{R}\bar{\mathbf{p}}_{c,\mathrm{C}} = \mathbf{w} \;:\; n^2 + L_C n^2 + 2L_C n \;\text{ flops.} \tag{34}$$

Note, from the last step in each method, that the QR factorization solution is more efficient to compute than the pseudoinverse.

Remark: The techniques described in Sections 3.2, 3.3, and 3.4 have been implemented in MATLAB Simulink to control the MP Delta Pro 3D printer like the implementation of the standard FBS approach in [3]. Standard FBS controllers have also been implemented on standalone micro-controllers for 3D printers by a company called Ulendo, which indicates that our techniques can also be applied to similar micro-controllers. As a visual representation, Fig. 4 shows a flowchart of the code implementation. In the following section, simulations and experiments are used to characterize and validate the proposed advantages of efficient computation and improved tracking accuracy.

4 Simulation and experimental validation

4.1 Simulation validation

In this subsection, we simulate the measured dynamics of the MP Delta Pro 3D printer with a variety of controllers to compare the computation time and accuracy of the controller proposed in Section 3 to that of potential alternatives. We evaluate the performance of following controllers:

1. a LPV FBS controller where the transfer functions are computed in matrix form using Eq. 4 at every point in each window and the coefficients are calculated with the pseudoinverse;
2. a controller that is the same as controller (a), except the transfer functions are computed using the parameterized model from Section 3.2;
3. a controller that is the same as controller (b), except the transfer functions are computed for only one point in the window—*without* the switching compensation discussed in Section 3.3;
4. a controller that is the same as controller (c), except *with* the switching compensation; and

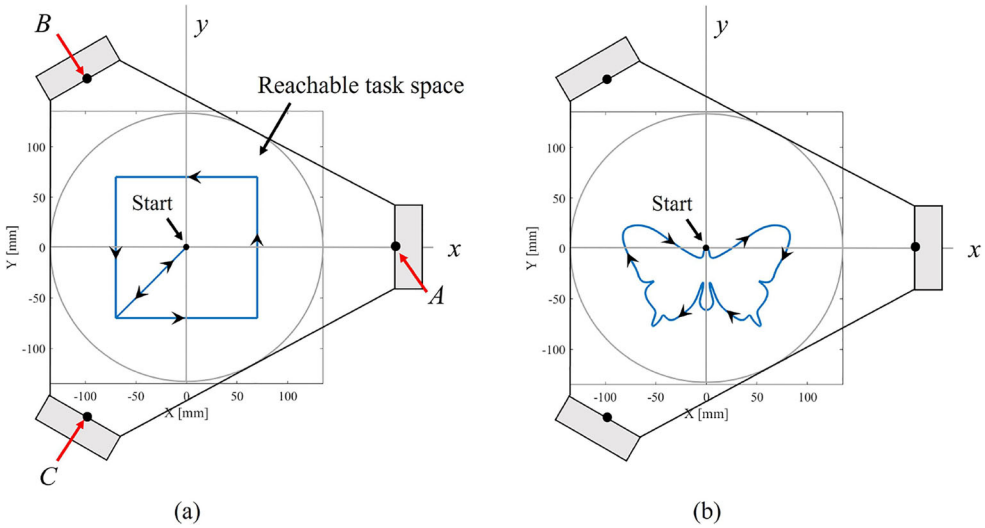

Fig. 5 Trajectories of (a) a square and (b) a butterfly used for simulations overlaid on the task space of the delta 3D printer. The square has a side length of 140 mm and is centered at the origin. The butterfly spans $x \in [-82, 82]$ mm and $y \in [-77, 23]$ mm. Both trajectories have with a maximum motion speed of 150 mm/s and a maximum acceleration of 20 m/s^2

5. a controller that is the same as controller (d), except the coefficients are calculated with QR factorization as described in Section 3.4.

By adding the modifications one at a time, we can distinguish the effects on computational efficiency and accuracy of each modification. Each controller's performance is also compared to a baseline controller—the standard FBS controller that uses an LTI model measured at $(x, y) = (0, 0)$ mm for the carriages. We use $(0, 0)$ because the model at the origin is a reasonable choice for the LTI model for FBS when the control designer does not have a model for the position-dependent dynamics.

The simulations are conducted using two trajectories:

1. the trajectory of a square shown in Fig. 5(a) with a side length of 140 mm and its center at the origin; and
2. the trajectory of a butterfly shown in Fig. 5(b), which spans $x \in [-83, 83]$ mm and $y \in [-77, 23]$ mm.

Both trajectories have a maximum speed of 150 mm/s and a maximum acceleration of 20 m/s^2. The square is selected to emphasize straight-line motions and sharp 90° turns which are prone to vibration. The butterfly trajectory is selected to emphasize curved motions with tight corners that are prone to contour errors. Each trajectory lasts about 5 s with a sample time of 1 ms. To simulate the system's response, the LSR of $\mathbf{G}(s)$ is computed using the known trajectory points. Parameter-varying compensation for all points (controllers (a) and (b)) is implemented by computing a point-by-point LSR matrix for each window. In other words, we compute the transfer function at each point in the window and compute its impulse response, which becomes the time-shifted columns of the LSR matrix (see Appendix). For the single point compensation (controllers (c)–(e)), we compute the transfer function and impulse response for the median point in the window, whose time-shifted impulse response is repeated to construct the LSR matrix for each window. All controllers use B-spline basis functions of degree $m = 5$, a window length $L_C = 196$ points, number of B-spline coefficients $n = 44$, and number of updated coefficients $n_{up} = 22$ (see [3] for details). The window length is determined by the amount of time required for the impulse response of the transfer function used for filtering (i.e., IIR filter) to settle close to zero. Since the transfer functions vary with position, we select the window length for the delta 3D printer using a one-time offline procedure: we construct a grid of positions in the reachable workspace that are 5 mm apart, compute the impulse response for the transfer

Table 1 Simulation results comparing the total computation time and accuracy of different controllers for generating modified trajectories of the square and butterfly

	Computation time	RMS contour error	Accuracy improvement from baseline
(*) Baseline LTI, standard FBS controller	0.014 / 0.044 s	5.94 / 11.13 μm	–
(a) Matrix TFs, all points, pseudoinverse	671.55 / 820.06 s	0.32 / 0.38 μm	94.6 / 96.6%
(b) Parameterized TFs, all points, pseudoinverse	181.16 / 226.30 s	0.32 / 0.38 μm	94.6 / 96.6%
(c) Parameterized TFs, single point, pseudoinverse	8.17 / 9.86 s	0.64 / 3.21 μm	89.3 / 71.1%
(d) Same as above with switching compensation	12.28 / 13.46 s	0.34 / 0.53 μm	94.3 / 95.3%
(e) Same as above with QR factorization	9.19 / 9.65 s	0.34 / 0.53 μm	94.3 / 95.3%

Results are listed as [Square / Butterfly]

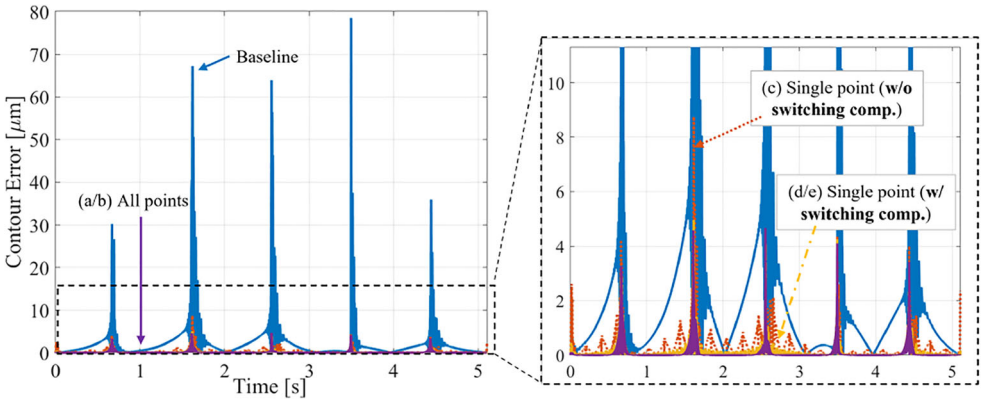

Fig. 6 Contour error of the modified "square" trajectory generated by the baseline controller (solid blue line) and controllers (a/b) using all trajectory points (solid purple line), (c) a single point without switching compensation (dotted red line), and (d/e) a single point with switching compensation (dash-dotted yellow line)

function at each position, and use the worst-case settling time to determine the window length. The number of B-spline coefficients is computed from the window length as described in [3].

The simulations are run in MATLAB (version R2022a) on a 64-bit Microsoft Surface Book with an Intel Core i5-6300U CPU processor and 8 GB of RAM. The computation time of the entire modified trajectory and the root-mean-square (RMS) contour error for each trajectory and each controller is reported in Table 1. The percent difference of the RMS contour error of other controllers compared to the RMS contour error of the baseline controller simulation is also reported as "Accuracy improvement from baseline." Time-series plots of the contour error comparisons are shown in more detail in Figs. 6 and 7 for the square and butterfly trajectories, respectively. For the butterfly trajectory, the RMS contour error of the baseline controller is 11.13 μm and the trajectory is computed in 44 ms since the filtering and inversion is completed offline. However, note that the RMS error of the exact LPV model is almost 30 times less than the baseline at 0.38 μm, although the computation of the matrix model online is much longer (820 s), which is also about 4 times greater than the computation time of the parameterized model (226 s) without any change in the RMS error. When we compute the model for a single point in each window, we can reduce the computation time by about 20x (to ~10 s) but at a cost of about 10x increase in RMS contour error (to 3.21 μm). The accuracy is improved to only be about 1.3× worse than the exact LPV model (about 0.5 μm) when the switching compensation is implemented, which indicates that we can significantly reduce the computation time while maintaining relatively high accuracy with controller (e). The results for the square trajectory follow a similar trend to the butterfly trajectory. The baseline controller starts with less absolute contour error compared to the butterfly because the trajectory is mostly composed of straight lines which are less prone to contour errors when compared to the butterfly's curved paths. Hence, we have less relative loss of accuracy using controller (c) for the square (~5% from 94.6 to 89.3%) when compared to the butterfly (~25% from 96.6 to 71.1%). However, we see a similar order of magnitude reduction in computation time from controller (a) to controllers (c) and (e) between the two trajectories, which indicates that the computational benefit of the proposed methodology is trajectory-agnostic. Furthermore, Table 2 reports the average (mean) computation time per window using each controller for both trajectories. Note that there is

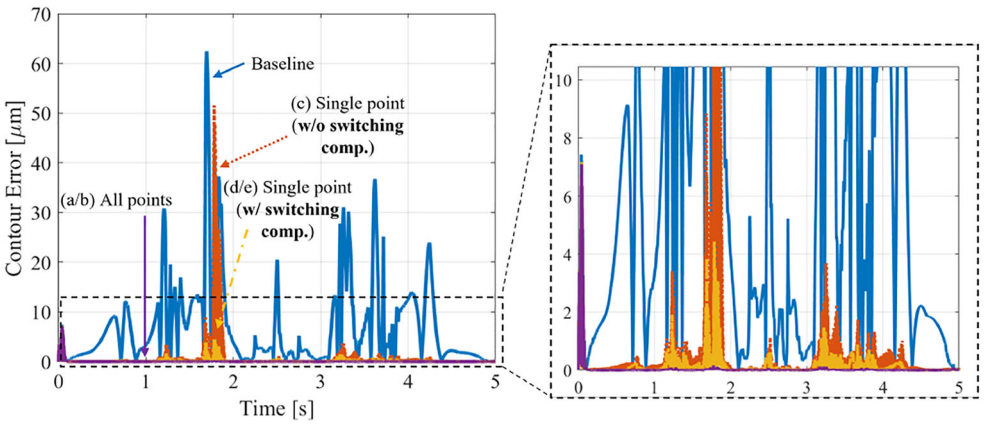

Fig. 7 Contour error of the modified "butterfly" trajectory generated by the baseline controller (solid blue line) and controllers (a/b) using all trajectory points (solid purple line), (c) a single point without switching compensation (dotted red line), and (d/e) a single point with switching compensation (dash-dotted yellow line)

Table 2 Mean computation time per window for generating modified trajectories using each controller

	Mean computation time per window
Baseline controller	4.57×10^{-4} / 7.04×10^{-4} s
Controller (a)	24.87 / 35.52 s
Controller (b)	6.71 / 12.27 s
Controller (c)	0.30 / 0.41 s
Controller (d)	0.46 / 0.52 s
Controller (e)	0.38 / 0.41 s

Results are listed as [Square / Butterfly]

a similar order of magnitude reduction in computation time when comparing the parameterized, single-point controller (controller (c)) to the exact LPV controller (controller (a)).

Shifting focus to the accuracy, we note a spike in the contour error of controller (c) around 2 s into the motion in Fig. 7. The spike represents a difficult portion of the butterfly trajectory—the bottom right of the butterfly wing—where a switch in models occurs for controllers (c)-(e). Here, the points are close to the far side of carriage B's line of action (see Figs. 2 and 5) which, as discussed in [17], is prone to larger dynamic variation. (The symmetric points on the bottom left side of the butterfly are not on the far side of carriage C's line of action and, thus, have less dynamic variation). Similarly, controller (c) leads to small spikes in error throughout the square trajectory that are not present for controller (d) as shown in Fig. 6. The spikes are exacerbated when a switch in windows coincides with a 90° corner on the square, which can be seen about 1.7 s into the motion. Generally, the contour error increases for all single point controllers (c), (d), and (e) when compared to the all-point controllers ((a) and (b)), but they are worse for controller (c) which does not include switching compensation. Finally, note that the total computation time is reduced by 28% for the butterfly trajectory and 25% for the square trajectory using QR factorization instead of the pseudoinversion (controller (e)). Overall, the accuracy of the single point approach is worse than using all the points in a window, but the overall accuracy improvement is acceptable given the (up to) 23× decrease in computation time compared to using the parameterized exact LPV controller (b), which is challenging to implement on hardware in real-time, as discussed in the following subsection on experiments.

4.2 Experimental validation

To study the impact of the proposed methodology on fabricated parts, we printed a "calibration cube"[1] and an extruded butterfly on the delta 3D printer using two control strategies: the baseline controller and controller (e) above. We focus most of our attention on the calibration cube since it is a well-known benchmark to characterize vibration in the 3D printing industry. Some layers of the cube's trajectory have square paths like those studied in the simulations. The cube also contains letter indentations which create additional sources of accuracy error. The butterfly is printed to evaluate the impact of each controller on curved layer parts. Our aim is to demonstrate the utility of our contributions by comparing (a) the quality of parts printed with our controller and the baseline controller at different positions and (b) acceleration amplitudes of the carriages during the execution of each print to understand the effects of the proposed methodology.

We position the center of each part at the following locations: $(x, y) = (0, 0), (-80, 0), (40, -69)$, and $(40, 69)$ mm. Each position, except the origin, is chosen to target each carriage independently; they are located 80 mm from the origin along the far side of the respective carriage's line-of-action. As shown in Figs. 2 and 8, the position $(-80, 0)$ mm primarily tests variation in carriage A's dynamics, $(40, -69)$ mm primarily tests variation in carriage B's dynamics, and $(40, 69)$ mm primarily tests variation in carriage C's dynamics [17]. The maximum speed and acceleration for both parts are 150 mm/s and 20 m/s^2, respectively. Both the baseline controller and controller (e) are implemented in MATLAB Simulink, which sends the motion commands through a dSPACE MicroLabBox to Pololu DRV8825 stepper motor drivers to move the stepper motors on the delta 3D printer. Using the dSPACE hardware and Simulink interface, only controller (e) compiles successfully for real-time implementation. Successful compilation of the program means that the commands will be sent to the machine at the MicroLabBox's command frequency of 1 kHz. In other words, despite the increase in each window's computation time for the proposed method compared to the baseline, there is no difference in the production time for the same part using the either method.

To quantify the quality differences between cubes printed at different positions, we use a laser scanner—the Romer Absolute Arm from Hexagon Metrology (model #7525SI)—to scan a face from each part. We chose to scan a flat face (without letter indentations) to isolate the vibration errors. (Unfortunately, some of the fine features of the butterfly cannot be captured accurately by the laser scanner so we only use it for the cube). Figure 9 shows a color map of three scanned parts which are all printed at $(-80, 0)$ mm using different compensation techniques. The uncompensated part has a clear vibration mark on the left side that is largely eliminated when using FBS compensation. The baseline FBS compensation leads to larger variations of surface position when compared to the

[1]The standard XYZ calibration cube used in this paper can be found at: https://www.thingiverse.com/thing:1278865

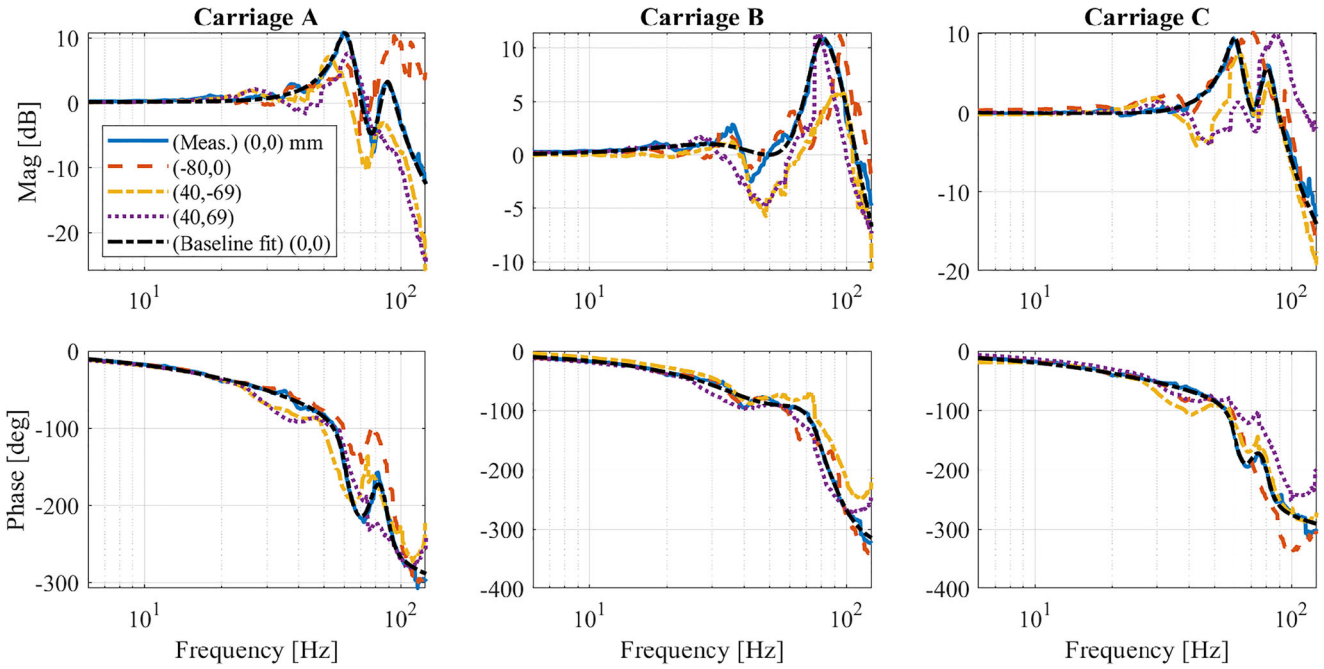

Fig. 8 Measured frequency response functions of the carriage position dynamics at $(x, y) = (0, 0)$ mm (blue solid lines), $(-80, 0)$ mm (red dashed lines), $(40, -69)$ mm (yellow dash-dotted lines), and $(40, 69)$ mm (indigo dotted lines) of the Monoprice Delta Pro 3D printer. The black dashed lines indicate the fitted transfer functions of the baseline model (at $(x, y) = (0, 0)$ mm) that is used for the baseline controller. Note that the carriage dynamics vary most on the far side of the carriages line of action (see Fig. 2): carriage A's dynamics vary significantly at position $(-80, 0)$ mm, carriage B's dynamics vary significantly at position $(40, -69)$ mm, and carriage C's dynamics vary significantly at position $(40, 69)$ mm

proposed controller. This rougher surface of the baseline FBS part can also be seen in Figs. 10 and 11, which show the respective images of the X and Y lettered faces of the calibration cube at all locations. To conserve page space, the color map images and point cloud data from the laser scans of the remaining parts are not shown but are publicly available on Github.[2] Here, we report the standard deviation of the point cloud surface for each scanned part in Table 3. Note that the proposed controller has lower point cloud surface variance across all the measured parts, which corresponds to smoother surfaces with less vibration errors.

For comparison, we also show the cube printed without vibration compensation in Figs. 10 and 11. A visual inspection of the all the fabricated parts reveals the following observations:

1. In the uncompensated parts, there are vibration marks at the edges where there is a change of direction, which are largely eliminated with FBS compensation.
2. The quality of the parts printed at $(0, 0)$ mm are similar for both the baseline and proposed controllers.
3. The surface quality of the part printed at $(-80, 0)$ mm with the baseline controller is worse than the quality of the part printed with the proposed controller.
4. The quality of the parts printed at $(40, -69)$ mm are similar for both controllers.
5. The part printed at $(40, 69)$ mm with the baseline controller drifts from its starting position in the middle of the print, while the part printed with the proposed controller stays aligned.
6. The quality of the parts printed with the proposed controller are always either similar to or better than the parts printed with the baseline controller.

Observation 2 is expected since the baseline controller performs optimally at $(0, 0)$ mm and observation 3 is expected due to the model mismatch. However, observation 4 appears to be an anomaly. A closer look at carriage B's FRFs in Fig. 8 reveals that the measured FRFs from $(0, 0)$ and $(40, -69)$ mm have similar resonance frequencies. Also note that carriages A and C have measured FRFs from $(0, 0)$ and $(40, -69)$ mm that also have similar resonance frequencies. Hence, the baseline controller is able to adequately compensate vibrations while printing at $(40, -69)$ mm. The drifting signal in observation

[2] https://github.com/nosaedoimioya/delta-printer-experiments.git

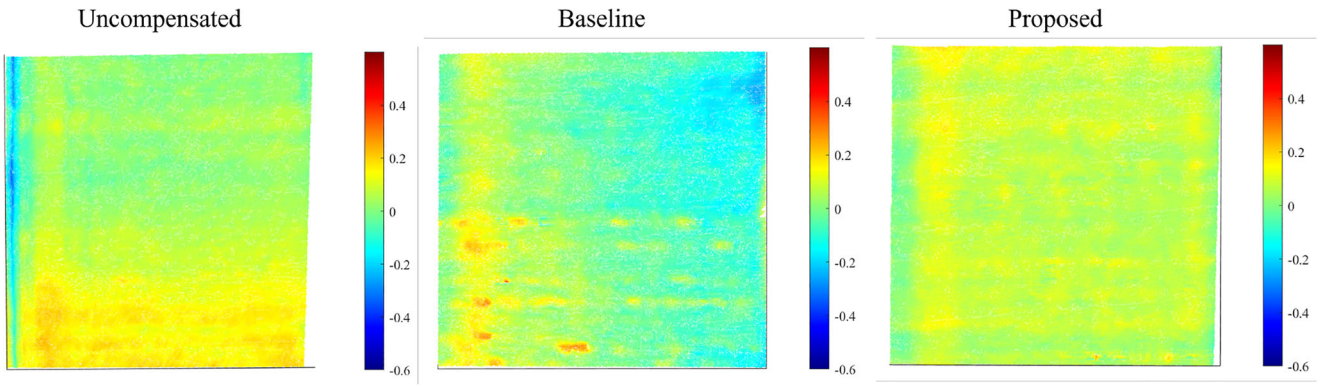

Fig. 9 Laser scanned color map of parts printed at (−80, 0) mm. The scanning shows that the surface quality of the uncompensated and baseline FBS strategies are worse than the surface quality of the proposed strategy, indicating improved surface accuracy

5 is due to the baseline controller overcompensating for the fast changes in acceleration on the top half of the Y-face of the cube. Note that the bottom half of the Y-face only has one indentation, while the top half has two indentations in succession, which increases the high frequency content of the acceleration profile. Figure 12 shows the modified motion commands of the baseline and proposed controllers in this region of the print, which shows that the commanded motion of the baseline controller drifts from the desired command while the proposed controller does not. Overcompensation occurs because the baseline model for carriage *C* (at (0, 0) mm in Fig. 8) shows that the amplitude of high frequency content is reduced. Hence, the baseline controller attempts to increase the input of the high frequency commands to achieve the desired motion. However, we know from the measured FRF at (40, 69) that the command does not need to be amplified. Thus, the proposed controller, with more accurate dynamics, can compensate correctly.

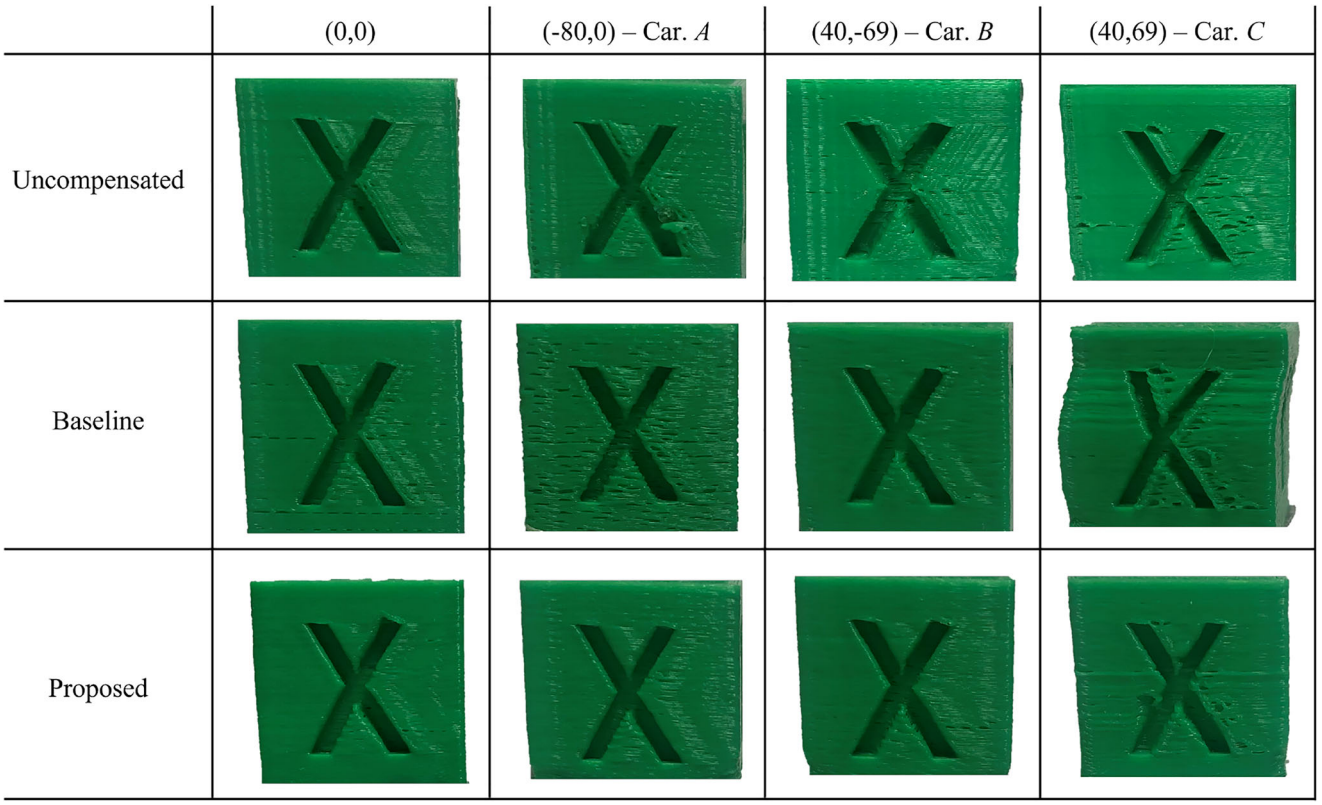

Fig. 10 *X*-axis face of calibration cubes fabricated with the baseline and proposed controllers centered at different positions that target different carriages

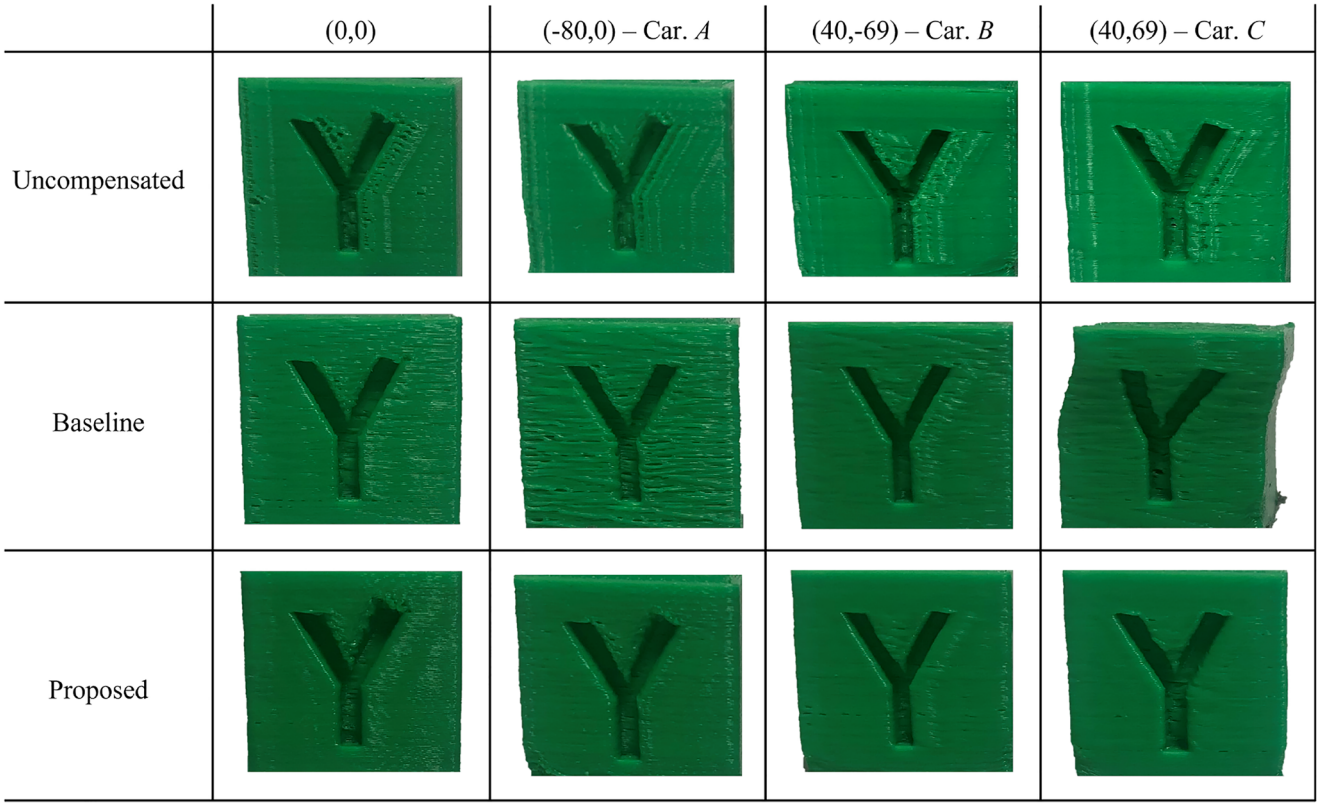

Fig. 11 Y-axis face of calibration cubes fabricated with the baseline and proposed controllers centered at different positions that target different carriages

Similar behavior is observed on the printed butterfly part. Figure 13 shows a comparison of the butterfly printed at (0, 0) and (40, 69) mm using both compensation strategies. The part that uses the baseline FBS compensation at (40, 69) drifts in a similar fashion to the cube. The other butterfly parts printed at different locations do not have such stark visual differences between the baseline FBS compensation and compensation with the proposed controller.[3] Hence, to quantify the reduction of vibration-induced acceleration, we also measure the acceleration of the carriages during each print using the vertical axis of an ADXL335 3-axis accelerometer from Sparkfun Electronics and compare the acceleration for both controllers to the acceleration of the desired trajectories for both the cube and butterfly. Tables 4 and 5 give the RMS and maximum values of the carriage acceleration, respectively, measured over the course of a few layers of the calibration cube print. In absolute terms, the proposed controller accelerations are closer to desired acceleration in all cases, illustrating reduction in vibration errors. The maximum difference of deviation reduction between the proposed controller and the baseline controller is 8.9% for carriage C at (40, 69) in the RMS acceleration and 20.8% for carriage A at $(-80, 0)$ in the maximum acceleration. Following the result from observation 4, we note that the least deviation from the desired acceleration occurs for carriage B at $(40, -69)$ for both RMS and maximum acceleration. Tables 6 and 7 give the RMS and maximum values of the carriage acceleration for a few layers of the butterfly print. The proposed method's acceleration measurements are also closer to desired acceleration in all cases for the butterfly part. The butterfly part also has the maximum difference of deviation reduction between the proposed controller and the baseline controller occurring at (40, 69) for the RMS acceleration and at $(-80, 0)$ for the maximum acceleration. Here, the deviation is reduced by 39.1% and 39.6%, respectively.

[3]The butterfly images are also publicly available online to conserve space: https://github.com/nosaedoimioya/delta-printer-experiments.git

Table 3 Standard deviation of point cloud data on the surface of scanned calibration cubes printed at various positions using different compensation strategies

	(0,0)	(−80,0)	(40,−69)	(40,69)
Uncompensated	80 μm	72 μm	160 μm	122 μm
Baseline	74 μm	83 μm	143 μm	772 μm
Proposed	53 μm	36 μm	130 μm	72 μm

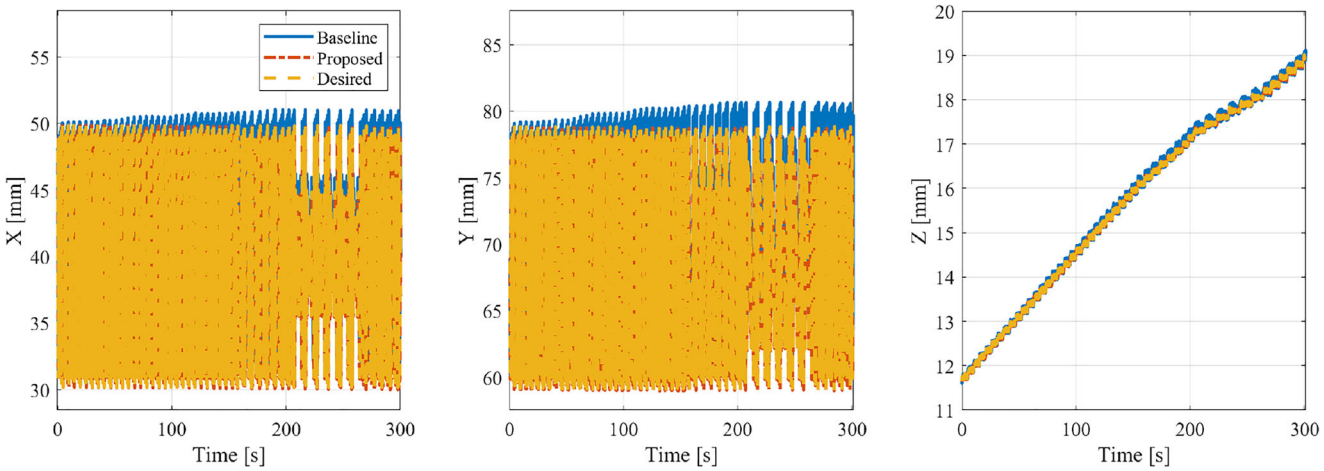

Fig. 12 Commanded motion from the baseline (blue solid line) and proposed (red dash-dot line) controller during the drifting motion for the baseline controller while fabricating the part at $(x, y) = (40, 69)$ mm (triggering carriage C). Note that the baseline model commands increasing deviations from the nominal position (yellow dashed line), which leads to the drifting part in Figs. 10 and 11. The baseline controller creates the drifting commands because of the differences between the baseline frequency response function and the actual frequency response function at (40, 69) mm

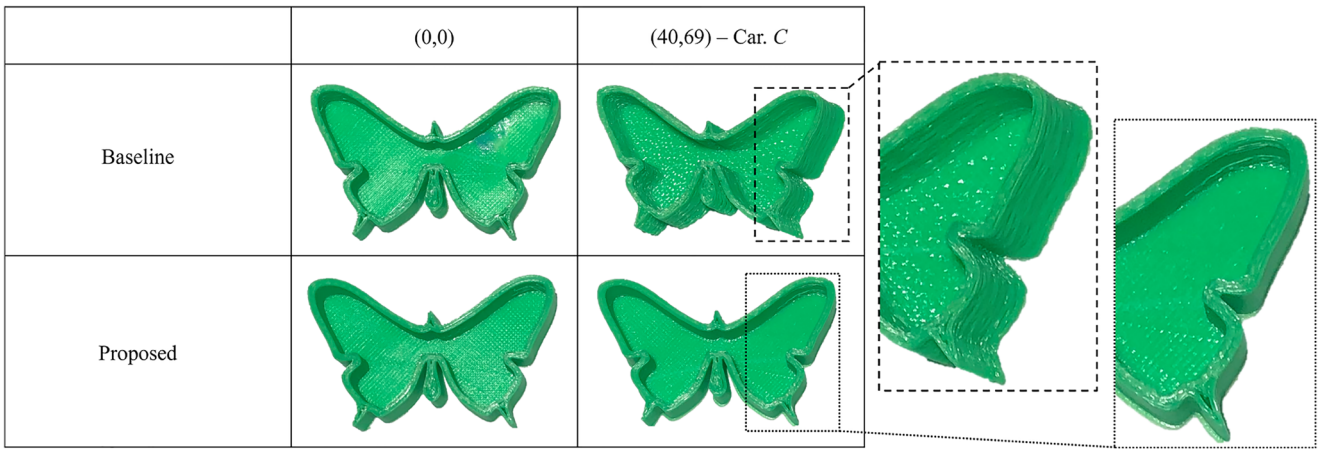

Fig. 13 Examples of the butterfly part printed at (0, 0) and (40, 69) mm using the baseline FBS and proposed compensation strategies. A drift in trajectory can be seen in the part printed at (40, 69) using the baseline FBS strategy, similar to the drifts seen in the calibration cube in Figs. 10 and 11

Table 4 Root mean square (RMS) acceleration of carriages during print of the calibration cube

	Baseline [m/s^2]	Proposed [m/s^2]	Desired [m/s^2]
(−80, 0) – Car. *A*	3.73 (+0.38)	3.24 (−0.11)	3.35
(40, −69) – Car. *B*	4.04 (+0.08)	3.95 (−0.01)	3.96
(40, 69) – Car. *C*	4.16 (+0.35)	3.82 (+0.01)	3.81

Table 5 Maximum acceleration of carriages during print of the calibration cube

	Baseline [m/s^2]	Proposed [m/s^2]	Desired [m/s^2]
(−80, 0) – Car. *A*	22.46 (+4.57)	17.04 (−0.85)	17.89
(40, −69) – Car. *B*	24.67 (+0.81)	24.05 (+0.19)	23.86
(40, 69) – Car. *C*	25.53 (+1.67)	23.68 (−0.18)	23.86

Table 6 Root mean square (RMS) acceleration of carriages during print of the butterfly

	Baseline [m/s^2]	Proposed [m/s^2]	Desired [m/s^2]
(−80, 0) – Car. *A*	4.31 (+1.38)	3.63 (+0.70)	2.93
(40, −69) – Car. *B*	2.96 (−0.13)	2.82 (−0.27)	3.09
(40, 69) – Car. *C*	4.29 (+1.35)	2.74 (−0.20)	2.94

Table 7 Maximum acceleration of carriages during print of the butterfly

	Baseline [m/s^2]	Proposed [m/s^2]	Desired [m/s^2]
(−80, 0) – Car. *A*	30.20 (+8.64)	21.47 (−0.09)	21.56
(40, −69) – Car. *B*	19.19 (−2.39)	21.41 (−0.17)	21.58
(40, 69) – Car. *C*	21.14 (+2.65)	19.38 (+0.89)	18.49

5 Conclusions

This paper proposes practical techniques to enable real-time, accurate vibration compensation on the prismatic-joint delta 3D printer. Previous work on improving accuracy of delta manipulators has focused on servo motor actuated machines and has relied on sensor measurements and feedback control. For most delta 3D printers, feedback sensors are not available so we must use feedforward control with an accurate dynamic model. Machine learning models have been recently adopted to model parameter varying machines and tune their controller gains, but their impact as end-to-end controllers is limited due to the difficulty of tuning hyper-parameters and guaranteeing stability. Here, we use an accurate physics-based LPV model of the delta 3D printer recently proposed in [17] with the feedforward model-based FBS approach. FBS can theoretically reduce vibration on delta 3D printers but the need to recompute the model and controller at each new position during real-time control is computationally challenging. Therefore, we propose the following techniques to decrease the computational burden: (1) parameterization and pre-filtering of portions of the model for fast online operations, (2) computation of the model at sampled points along the trajectory (while preserving continuity of the controller's predictions when the model changes), and (3) utilization of matrix methods that yield faster matrix inversion.

Simulations are used to assess the trade-off between computation time and accuracy. We report that the techniques presented in this paper result in a 23× reduction in computation time from the exact parameter varying controller which re-computes the model/controller at every point. Thus, our approximations save significant computational effort while only increasing contour errors by up to about 1.3× compared to the exact LPV controller. Printed parts from our experiments, which are pictured and scanned, also show an overall improvement in the quality of parts printed at different locations using the proposed controller compared to using a baseline controller that is optimized for the center of the workspace. Furthermore, acceleration measurements during printing show more than 39% reduction of vibration-induced accelerations for the proposed controller when compared to the baseline. This work shows that we can take advantage of the high speed motion of the delta 3D printer (compared to traditional 3D printers) and apply feedforward controllers like FBS to maintain accuracy during vibration-prone motion. This paper's contributions bring us one step closer to the promise of high speed and high quality additive manufacturing.

Appendix: Lifted system representation of a digital filter

As discussed in [32], consider digital filter p, input signal u, and output signal y defined as:

$$p = \{p_{-2}p_{-1}p_0p_1p_2\} \tag{35}$$
$$u = \{u_0u_1u_2\} \tag{36}$$
$$y = \{y_0y_1y1\} \tag{37}$$

Signals y and u and filter p are related by the convolution operator as follows:

$$y = u * p \tag{38}$$

From Eqs. 35 to 38,

$$y_0 = p_0u_0 + p_{-1}u_1 + p_{-2}u_2 \tag{39}$$
$$y_1 = p_1u_0 + p_0u_1 + p_{-1}u_2 \tag{40}$$
$$y_2 = p_2u_0 + p_1u_1 + p_0u_2 \tag{41}$$

This can be expressed in matrix form as

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} p_0 & p_{-1} & p_{-2} \\ p_1 & p_0 & p_{-1} \\ p_2 & p_1 & p_0 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} \tag{42}$$

Note that the main diagonal element (p_0) represents the influence of the current input on the current output; the first upper diagonal element (p_{-1}) represents the influence of the succeeding input on the current output and the second upper diagonal element (p_{-2}) represents the influence of the second succeeding input on the current output. Similarly, the first (p_1) and second lower (p_2) elements represent the influence of the first and second preceding inputs on the current output, respectively. Hence, the discrete time transform of p obtained from Eq. 42 is given by

$$p_2z^{-2} + p_1z^{-1} + p_0z^0 + p_{-1}z^1 + p_{-2}z^2 \tag{43}$$

which is in accordance with the time-domain definition given in Eqs. 35–37.

Author contribution All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Nosakhare Edoimioya. Conception and implementation of QR factorization for FBS was performed by Cheng-Hao Chou. The first draft of the manuscript was written by Nosakhare Edoimioya and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding This work was supported in part by the National Science Foundation [grant numbers 2054715 and DGE 1256260] and a Michigan Space Grant Consortium (MSGC) graduate fellowship from the National Aeronautics and Space Administration (NASA), under award number 80NSSC20M0124.

Declarations

Conflict of interest Authors N. Edoimioya and C.-H. Chou declare they have no financial interests. A company founded by C.E. Okwudire holds a commercial license for the filtered B-splines (FBS) algorithm.

References

1. Zakharov OV, Pugin KG, Ivanova TN (2022) Modeling and analysis of delta kinematics FDM printer. J. Physics: Conf. Series, vol. 1:2182. https://doi.org/10.1088/1742-6596/2182/1/012069
2. Ramani KS, Edoimioya N, Okwudire CE (2020) A robust filtered basis functions approach for feedforward tracking control – with application to a vibration-prone 3D printer, IEEE/ASME Trans. Mechatronics 25(5):2556–2564. https://doi.org/10.1109/TMECH.2020.2983680
3. Duan M, Yoon D, Okwudire CE (2018) A limited-preview filtered B-spline approach to tracking control – with application to vibration-induced error compensation of a 3D printer. Mechatronics 56:287–296. https://doi.org/10.1016/j.mechatronics.2017.09.002
4. Kim H, Okwudire CE (2020) Simultaneous servo error precompensation and feedrate optimization with tolerance constraints using linear programming. Int. J. of Adv. Manufac. Tech. 109(3-4):809–821. https://doi.org/10.1007/s00170-020-05651-w
5. Okwudire CE, Huggi S, Supe S, Huang C, Zeng B (2018) Low-level control of 3D printers from the cloud: a step toward 3D printer control as a service. Inventions 3(3):56. https://doi.org/10.3390/inventions3030056
6. Codourey A (1998) Dynamic modeling of parallel robots for computed-torque control implementation. Int. J. Robot. Res. 17(18):1325–1336. https://doi.org/10.1177/027836499801701205
7. Angel L, Viola J, Fractional order PID (2018) For tracking control of a parallel robotic manipulator type delta. ISA Trans 79:172–188. https://doi.org/10.1016/j.isatra.2018.04.010
8. Boudjedir CE, Boukhetala D, Bouri M (2018) Nonlinear PD plus sliding mode control with application to a parallel delta robot. J Electr Eng 69(5):329–336. https://doi.org/10.2478/jee-2018-0048
9. Ramirez-Neria M, Sira-Ramírez H, Luviano-Juárez A, Rodriguez-Ángeles A (2015) Active disturbance rejection control applied to a delta parallel robot in trajectory tracking tasks. Asian J. Control 17(2):636–647. https://doi.org/10.1109/ACC.2012.6314934
10. Castañeda LA, Luviano-Juárez A, Chairez I (2015) Robust trajectory tracking of a delta robot through adaptive active disturbance rejection control. IEEE Trans Control Syst Technol 23(4):1387–98. https://doi.org/10.1109/TCST.2014.2367313
11. Escorcia-Hernandez JM, Aguilar-Sierra H, Aguilar-Mejia O, Chemori A, Arroyo-Nunez JH (2019) An intelligent compensation through B-spline neural network for a delta parallel robot. In: Proc. 6th Int. Conf. Control, Decis. Inf. Technolo. (CoDIT), pp 361–366. https://doi.org/10.1109/CoDIT.2019.8820472
12. Pham P-C, Kuo Y-L (2022) Robust adaptive finite-time synergetic tracking control of delta robot based on radial basis function neural networks. Appl Sci 12:10861. https://doi.org/10.3390/app122110861
13. Su Y, Sun D, Ren L, Mills JK (2006) Integration of saturated PI synchronous control and PD feedback for control of parallel manipulators. IEEE Trans. Robot. 22(1):202–207. https://doi.org/10.1109/TRO.2005.858852
14. Chiacchio P, Pierrot F, Sciavicco L, Siciliano B (1993) Robust design of independent joint controllers with experimentation on a high-speed parallel robot. IEEE Trans Ind Electron 40(4):393–403. https://doi.org/10.1109/41.232228
15. Zhiyong Y, Tian H (2004) A new method for tuning PID parameters of a 3 DoF reconfigurable parallel kinematic machine. Proc IEEE Int Conf Robot Autom 2249–2254. http://doi.org/10.1109/ROBOT.2004.1307396
16. Zhou Q, Panfeng W, Jiangping M (2015) Controller parameter tuning of delta robot based on servo identification. Chinese J Mech Eng 28(2):267–275. https://doi.org/10.3901/CJME.2014.1117.169
17. Edoimioya N, Okwudire CE (2022) A generalized and efficient control-oriented modeling approach for vibration-prone delta 3D printers using receptance coupling, Trans. Autom. Science Eng (TASE). https://doi.org/10.1109/TASE.2022.3197057
18. Bègon P., Pierrot F, Dauchez P (1995) Fuzzy sliding mode control of a fast parallel robot. Proc IEEE Int Conf Robot Autom 1178–1183. https://doi.org/10.1109/ROBOT.1995.525440
19. Yang W, Liu W (2021) Delta robot trajectory tracking based on fuzzy adaptive sliding mode controller, Proc IEEE China Autom. Cong. (CAC) 7041–7046. https://doi.org/10.1109/CAC53003.2021.9727620
20. Azad FA, Rahimi S, Yazdi MRH, Masouleh MT (2020) Design and evaluation of adaptive and sliding mode control for a 3-DOF Delta parallel robot. Proc. IEEE 28th Iranian Conf Elec Eng 1–7. https://doi.org/10.1109/ICEE50131.2020.9261040
21. Liu Y-P, Altintas Y (2022) Predicting the position-dependent dynamics of machine tools using progressive network. Precis Eng 73:409–422. https://doi.org/10.1016/j.precisioneng.2021.10.010
22. Patiño HD, Carelli R, Kuchen BR (2002) Neural networks for advanced control of robot manipulators. IEEE Trans Neural Networks 13(2):343–354. https://doi.org/10.1109/72.991420
23. Carelli R, Camacho EF, Patiño HD (1995) A neural network based feedforward adaptive controller for robots. IEEE Trans Syst Man Cybernetics 25(9):1281–1288. https://doi.org/10.1109/21.400506
24. Cheng L, Hou Z-G, Tan M (2009) Adaptive neural network tracking control for manipulators with uncertain kinematics, dynamics and actuator model. Automatica 45(10):2312–2318. https://doi.org/10.1016/j.automatica.2009.06.007
25. Rahimi S, Jalali H, Yazdi MRH, Kalhor A, Masouleh MT (2772) Design and practical implementation of a neural network self-tuned inverse dynamic controller for a 3-DoF delta parallel robot based on arc length function for smooth trajectory tracking. Mechatonics 84(10):2022. https://doi.org/10.1016/j.mechatronics.2022.102772
26. Park SS, Altintas Y, Movahhedy M (2003) Receptance coupling for end mills. Int J Mach Tools Manuf 43(9):889–896. https://doi.org/10.1016/S0890-6955(03)00088-9
27. Law M, Ihlenfeldt S (2015) A frequency-based substructuring approach to efficiently model position-dependent dynamics in machine tools. Proceedings of the Institution of Mechanical Engineers Part K: Journal of Multi-body Dynamics, 229(3):304–317. https://doi.org/10.1177/1464419314562264
28. van Zundert J, Oomen T (2018) On inversion-based approaches for feedforward and ILC. Mechatronics 50:282–291. https://doi.org/10.1016/j.mechatronics.2017.09.010
29. Erkorkmaz K, Altintas Y (2001) High speed CNC system design. Part I: jerk limited trajectory generation and quintic spline interpolation. Int. J. Mach. Tools Manuf. 41(9):1323–1345. https://doi.org/10.1016/S0890-6955(01)00002-5
30. Tajima S, Sencer B (2022) Online interpolation of 5-axis machining toolpaths with global blending. Int J Mach Tools Manuf 175:103862. https://doi.org/10.1016/j.ijmachtools.2022.103862
31. Singhose W (2009) Command shaping for flexible systems: a review of the first 50 years. Int J Precis Eng Manuf 10(4):153–168. https://doi.org/10.1007/s12541-009-0084-2

32. Ramani KS, Duan M, Okwudire CE, Ulsoy AG (2017) Tracking control of linear time-invariant nonminimum phase systems using filtered basis functions. J. Dyn. Syst. Meas. Control 139(1):01,1001. https://doi.org/10.1016/j.cirp.2016.04.100
33. Rigney BP, Pao LY, Lawrence DA (2009) Nonminimum phase dynamic inversion for settle time applications. IEEE Trans Control Syst Technol 17(5):989–1005. https://doi.org/10.1109/TCST.2008.2002035
34. Clayton GM, Tien S, Leang KK, Zou Q, Devasia S (2009) A review of feedforward control approaches in nanopositioning for high-speed SPM. J Dyn Syst Meas Control 131(6):06,1101. https://doi.org/10.1115/1.4000158
35. Okwudire CE, Ramani KS, Duan M (2016) A trajectory optimization method for improved tracking of motion commands using CNC machines that experience unwanted vibration, CIRP. Ann Manuf Technol 65(1):373–376. https://doi.org/10.1016/j.cirp.2016.04.100
36. Kasemsinsup Y, Romagnoli R, Heertjes M, Weiland S, Butler H (2017) Reference-tracking feedforward control design for linear dynamical systems through signal decomposition. Am Control Conf 2387–2392. https://doi.org/10.23919/ACC.2017.7963310
37. Edoimioya N, Ramani KS, Okwudire CE (2021) Software compensation of undesirable racking motion of H-frame 3D printers using filtered B-splines. Additive Manuf 47. https://doi.org/10.1016/j.addma.2021.102290
38. Piegl L, Tiller W (1995) The NURBS book, Heidelberg. Springer, Berlin
39. Duan M, Okwudire C (2016) Minimum-time cornering for CNC machines using an optimal control method with NURBS parameterization. Int J Adv Manuf Technol 85:1405–1418. https://doi.org/10.1007/s00170-015-7969-2
40. Golub GH, Van Loan CF (1996) Matrix computations, 3rd edn. The Johns Hopkins University Press, Baltimore
41. Schwarz HR, Rutishauser H, Stiefel E (1973) Numerical analysis of symmetric matrices, translation of Numerik symmetrischer Matrizen, Prentice-Hall, Englewood Cliffs N.J.