Software for Modeling and Analysis of Linear Systems with Delays

Pascal Gahinet
The MathWorks
3 Apple Hill, Natick, MA, 01760
pascal@mathworks.com

Lawrence F. Shampine
Southern Methodist University
Dallas, TX 75275
lshampin@mail.smu.edu

***Abstract*— This paper proposes a new framework for modeling linear time-invariant (LTI) systems with delays. Key benefits of this framework are that it can handle delays in feedback loops, is general enough for most control applications, and lends itself well to computer-aided analysis. The accompanying software tools should facilitate the design of control systems in the presence of delays, as well as stimulate research into efficient numerical algorithms for assessing the properties and performance of such systems.**

I. Introduction

Time delays, also referred to as time lags, dead times, or transport delays, arise in many control applications. Long delays are common in process control and can severely limit the performance of control systems. Short delays are common in automotive and aerospace applications and can degrade performance if not properly accounted for. While much research has been devoted to extending classical and modern control techniques to accomodate delays, there is a lack of comprehensive software tools for manipulating and analyzing systems with delays. Most popular packages for linear analysis and design [4], [3], [17] offer limited support for systems with delays, and most available packages for delay differential equations (DDE) [10], [5] are either too restrictive or too cumbersome for control design purposes.

This paper proposes a new framework for computer-aided manipulation and analysis of linear time-invariant (LTI) models with delays. At the heart of this framework is an LFT-based representation of such systems (LFT stands for linear fractional transformation, see [14] and references therein). This representation has several key advantages:

- It can handle delays in feedback loops and is general enough for most control applications
- Most classical software tools for analyzing delay-free LTI systems can be extended to this class of LTI systems with delays
- Efficient and specialized DDE solvers can be developed for time-domain simulation of such systems.

Given the widespread use of linear techniques in control system design, this framework and the accompanying software tools should facilitate the computer-aided analysis and design of control systems in the presence of delays, as well as stimulate more research into efficient numerical algorithms for assessing the properties and performance of such systems.

The paper is organized as follows. First we formally define the class of delayed LTI systems under consideration, and show that it covers most practical needs in control applications. We then discuss algorithms for computing the time and frequency response of such systems. Finally, we show how existing software tools for delay-free LTI systems can be seamlessly extended to this class of systems, and illustrate the convenience and power of the resulting tools on a process control example.

II. LFT-Based Modeling of LTI Systems with Delays

Recall that the linear-fractional transformation (LFT) is defined for matrices by

$$\mathcal{L}(\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}, \Theta) := M_{11}+M_{12}\Theta(I-M_{22}\Theta)^{-1}M_{21} .$$

An equivalent expression when Θ is invertible is

$$\mathcal{L}(\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}, \Theta) := M_{11}+M_{12}(\Theta^{-1}-M_{22})^{-1}M_{21} . \tag{1}$$

The LFT has been extensively used in robust control theory for representing models with uncertainty, see [14] for details.

Consider the class GLTI of continuous-time LTI systems whose transfer function is of the form

$$H(s,\tau) = \mathcal{L}(\underbrace{\begin{pmatrix} H_{11}(s) & H_{12}(s) \\ H_{21}(s) & H_{22}(s) \end{pmatrix}}_{H(s)}, \Theta(s,\tau))$$
$$\Theta(s,\tau) := \mathrm{Diag}\left(e^{-\tau_1 s},\ldots,e^{-\tau_N s}\right) \tag{2}$$

where $H(s)$ is a rational (delay free) MIMO transfer function, and $\tau = (\tau_1,\ldots,\tau_N)$ is a vector of nonnegative time delays. Systems in this class are modeled as the LFT interconnection of a delay-free LTI model and a bank of pure delays (see Figure 1). As such, there are clearly linear time-invariant. Also, pure delays are in this class since $e^{-\tau s} = \mathcal{L}(\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, e^{-\tau s})$. Key properties of the GLTI class are captured in the next two results.

Theorem 2.1: Any block diagram interconnection of GLTI systems is a GLTI system. In other words, the class of GLTI systems is closed under series, parallel, and feedback connections as well as branching/summing junctions.

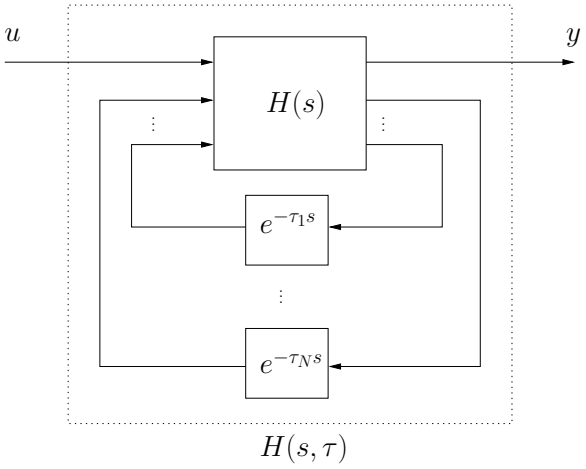

Fig. 1. LFT-based modeling of LTI systems with delays.

Proof: This property of LFT-based models is well known in robust control theory and can be established as follows. Consider a collection of GLTI models $H_i(s,\tau_i) = \mathcal{L}(H_i(s),\Theta(s,\tau_i))$. Any interconnection of these models is of the form $\mathcal{L}(M,\mathrm{Diag}(H_1(s,\tau_1),\ldots,H_N(s,\tau_N)))$ where M is a Boolean matrix describing the block diagram connectivity. Straightforward diagram manipulations show that

$$\mathrm{Diag}(H_1(s,\tau_1),\ldots,H_N(s,\tau_N))) = \mathcal{L}(H(s),\Theta(s,\tau))$$

where

- $H(s)$ is $\mathrm{Diag}(H_1(s),\ldots,H_N(s))$ up to some reordering of the input and output channels
- τ is the concatenation of the vectors $\tau_1,\ldots,\tau_N$.

The proof is complete by observing that

$$\mathcal{L}(M,\mathcal{L}(H(s),\Theta(s,\tau))) = \mathcal{L}(\mathcal{L}(M,H(s)),\Theta(s,\tau))$$

where $\mathcal{L}(M,H(s))$ is rational as the LFT interconnection of M and $H(s)$. ■

Theorem 2.2: The linearization of any nonlinear block diagram with time delays is a GLTI system.

These two results show that the GLTI class is general enough to model any (linearized) system with a finite number of delays, including delays in the feedback path. Delays at the inputs or outputs are clearly covered as the series connection of a delay-free model with pure delays. Yet, for efficiency reasons, such delays are best kept separate from the internal delays $\Theta(s,\tau)$ (see [4] for details). Finally, note that GLTI models cannot represent time-varying delays or distributed delays (see [7], [11] for alternatives).

Discrete-time LTI systems with delays can be represented in a similar fashion as

$$H(z,\delta) = \mathcal{L}(\underbrace{\begin{pmatrix} H_{11}(z) & H_{12}(z) \\ H_{21}(z) & H_{22}(z) \end{pmatrix}}_{H(z)}, \Theta(z,\delta))$$
$$\Theta(z,\delta) := \mathrm{Diag}\left(z^{-\delta_1},\ldots,z^{-\delta_N}\right) \qquad (3)$$

where $\delta = (\delta_1,\ldots,\delta_N)$ is a vector of delays expressed as integer multiples of the sampling period. While $H(z,\delta)$ is a rational transfer function that can be analyzed with standard LTI tools, its order can grow large when the delays are long compared to the sampling period. The GLTI representation is an attractive alternative in such cases because it keeps the delays separate from the model dynamics $H(z)$ and lends itself to efficient time- and frequency-response computations.

Interestingly, GLTI models are also well suited for computer-aided manipulation and analysis. Because LFT and transfer functions do not mix well numerically, from now on we will work with the state-space counterpart of (2).

III. State-Space Equations for GLTI Systems

Let

$$\begin{pmatrix} H_{11}(s) & H_{12}(s) \\ H_{21}(s) & H_{22}(s) \end{pmatrix} = \left(\begin{smallmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{smallmatrix}\right) + \left(\begin{smallmatrix} C_1 \\ C_2 \end{smallmatrix}\right)(sI-A)^{-1}\left(\begin{smallmatrix} B_1 & B_2 \end{smallmatrix}\right)$$

be a minimal realization of $H(s)$ in (2). State-space equations for $H(s,\tau) = \mathcal{L}(H(s),\Theta(s))$ are readily obtained as

$$\frac{dx}{dt} = Ax(t) + B_1u(t) + B_2w(t) \qquad (4)$$
$$y(t) = C_1x(t) + D_{11}u(t) + D_{12}w(t) \qquad (5)$$
$$z(t) = C_2x(t) + D_{21}u(t) + D_{22}w(t) \qquad (6)$$
$$w(t) = (\Delta_\tau z)(t) \qquad (7)$$

where

- $u(t)$ and $y(t)$ are the input and output vectors, respectively
- $w(t)$ and $z(t)$ are internal signals commensurate with the vector τ of time delays
- $\Delta_\tau z$ is the vector-valued signal defined by $(\Delta_\tau z)(t) := \begin{pmatrix} z_1(t-\tau_1) \\ \vdots \\ z_N(t-\tau_N) \end{pmatrix}$

Note that standard delay-free state-space models are just a special case of (4)-(7) corresponding to $N=0$, a handy fact when it comes to integrating GLTI models with existing sofware for manipulating delay-free state-space models.

Delay LTI systems of the form

$$\frac{dx}{dt} = A_0x(t) + B_0u(t) + \sum_{j=1}^{M}(A_jx(t-\theta_j) + B_ju(t-\theta_j)) \qquad (8)$$
$$y = C_0x(t) + D_0u(t) + \sum_{j=1}^{M}(C_jx(t-\theta_j) + D_ju(t-\theta_j)) \qquad (9)$$

are often considered in the literature with various restrictions on the number and locations of the delays $\theta_1,\ldots,\theta_M$. It turns out that any model of this form belongs to the class GLTI. To see this, use the SVD to compute a minimum rank factorization

$$\begin{pmatrix} A_j & B_j \\ C_j & D_j \end{pmatrix} = \begin{pmatrix} P_j \\ Q_j \end{pmatrix}\begin{pmatrix} R_j & S_j \end{pmatrix}$$

let ρ_j be the rank of this factorization, and introduce the auxiliary vector-valued signals

$$z(t) = \begin{pmatrix} R_1 x(t) + S_1 u(t) \\ \vdots \\ R_M x(t) + S_M u(t) \end{pmatrix}, w(t) = \begin{pmatrix} z_1(t-\theta_1) \\ \vdots \\ z_M(t-\theta_M) \end{pmatrix}$$

With this notation, (8)-(9) can be rewritten as

$$\begin{aligned} \frac{dx}{dt} &= A_0 x(t) + B_0 u(t) + \begin{pmatrix} P_1 & \dots & P_M \end{pmatrix} w(t) \\ y(t) &= C_0 x(t) + D_0 u(t) + \begin{pmatrix} Q_1 & \dots & Q_M \end{pmatrix} w(t) \\ z(t) &= \begin{pmatrix} R_1 \\ \vdots \\ R_M \end{pmatrix} x(t) + \begin{pmatrix} S_1 \\ \vdots \\ S_M \end{pmatrix} u(t) \\ w(t) &= (\Delta_\tau z)(t) \end{aligned}$$

where

$$\tau = (\underbrace{\theta_1, \dots, \theta_1}_{\rho_1}, \dots, \underbrace{\theta_M, \dots, \theta_M}_{\rho_M}) .$$

Conversely, note that any GLTI model with strictly upper-triangular D_{22} (up to reordering the delays $\tau_1, \dots, \tau_N$) is a special case of (8)-(9). Indeed, combining (6)-(7) yields the functional equation:

$$(I - D_{22}\Delta_\tau) z = C_2 x + D_{21} u \tag{10}$$

When D_{22} is strictly upper-triangular, the operator $D_{22}\Delta_\tau$ is nilpotent with index at most N, and $(I - D_{22}\Delta_\tau)^{-1} = \sum_{k=1}^{N-1} (D_{22}\Delta_\tau)^k$. It follows that

$$\begin{aligned} z &= \sum_{k=1}^{N-1} (D_{22}\Delta_\tau)^k (C_2 x + D_{21} u) \\ &= \sum_{j=1}^{s} (\mathcal{C}_j x(t-\theta_j) + \mathcal{D}_j u(t-\theta_j)) \end{aligned}$$

where the time delays θ_j include $\tau_1, \dots, \tau_N$ and additive combinations thereof. The result follows by observing that $w = \Delta_\tau z$ is itself of this form, and substituting w into (4)-(5).

IV. Time and Frequency Response Computation

Computing the frequency response of GLTI models presents no particular difficulty when working with the characterization (2). For a given frequency grid, the frequency response of the rational part $H(s)$ is computed using standard algorithms [9] and the frequency response of $\Theta(s, \tau)$ is readily evaluated. The LFT formula (1) is then evaluated at each frequency point to compute the overall response of $H(s, \tau)$.

Computing time-domain responses such as the step response is more challenging. The equations (4), (6), and (7) form a system of delay differential-algebraic equations (DDAEs). As explained in the introductory texts [2], [12], we still have much to learn about how to solve numerically DAEs and DDEs by themselves, much less in combination. DDAEs have received little attention in the literature and the results available are rather specific to the application, c.f. [1] and the references therein.

A crude approach to integrating (4)-(7) consists of using some off-the-shelf variable-step ODE solver together with finite-length buffers for keeping track of the past values of $z(t)$ needed to evaluate $w(t)$. Assuming the solver has so far computed as solution on $[0, t_n]$, values of $z(t)$ are obtained by interpolation for $t \le t_n$, and by zero-order-hold extrapolation for $t > t_n$. While this approach works well on many problems, it comes with no guarantee when taking steps longer than the smallest delay due to the adhoc nature of the z-extrapolation scheme. This encouraged us to develop specialized solvers for GLTI problems.

An equivalent set of DDEs is obtained by differentiating the algebraic equation (6). In the parlance of DAEs, this shows the problem is of index 1. In the parlance of DDEs, it shows that the equations are of neutral type because the equation for $\dot{z}(t)$ involves $\dot{z}(t)$ itself with delayed arguments (here $\dot{z}$ denotes the first derivative of z). If we want to compute the response to a step input

$$u(t) = \begin{cases} 0 & \text{for } t < 0 \\ 1 & \text{for } t \ge 0 \end{cases} ,$$

the discontinuity at $t = 0$ generally introduces a jump in z and $\dot{x}$ there. It is characteristic of DDEs of neutral type that the delays cause these jumps to propagate to all times $t = \sum_{i=1}^{N} m_j \tau_j$ where $m_1, \dots, m_N$ are nonnegative integers. Numerical methods do not have their usual behavior when straddling a jump discontinuity, so we must track the discontinuities and apply the method only where the solution is smooth, or we must develop a numerical method that can cope with a solution that is only piecewise smooth. We have done both in our solvers.

The algebraic variables $z(t)$ appear in a simple way, so our programs for solving (4,6,7) have much more in common with a DDE solver than a DAE solver. Many DDE solvers restrict the step size to the length of the shortest delay because an explicit formula for ODEs can be used then. We can neither use an explicit method nor accept this restriction. For one thing, small delays are common in control applications, delays so small that simulation is impractical with this restriction. If we take a step longer than the shortest delay, the numerical solution is defined implicitly even when the formula itself is explicit. By evaluating the numerical solution iteratively in this situation, it is practical to simulate systems with short delays. We have to use an implicit method anyway because GLTI problems can be stiff. The Radau IIa two-stage implicit Runge–Kutta formula is a one-step method that is L-stable and of order 3. A one-step method facilitates the handling of propagated discontinuities. The stability is all that we could hope for and order 3 is reasonable for simulation. Because the DDAEs are linear and the matrices are constant, it is possible to evaluate this implicit method very efficiently simply by solving a system of linear equations. When solving DDEs we must be able to approximate the solution

everywhere so that we can evaluate delayed terms. A cubic Hermite polynomial interpolant to the values of x and $\dot{x}$ at both ends of a step provides an accurate approximation over the span of the step. The algebraic variables are less smooth, so we form a cubic polynomial interpolant to the values of z at the ends of the step and two interior points.

DDEs of neutral type can be exceedingly difficult to solve because of propagated discontinuities. Widely-used programs for the task may track discontinuities, but all have at least the option of not tracking. Tracking discontinuities means here that we locate the jumps in $z(t)$, integrate the smooth solution between successive jumps, and compute exactly the sizes of the jumps. We do this in our programs, but there can be so many jumps that it is impractical to track them all. As a compromise, we track only the first 100 jumps and thereafter approximate $z(t)$ by a continuous function. Fortunately, for stable GLTI there are reasons to think that the sizes of jumps decay as the simulation proceeds.

After we stop tracking discontinuities, we integrate equations with solutions that are only piecewise smooth. The residual of a numerical solution is the amount by which it fails to satisfy the equations. Because we determine a piecewise-cubic polynomial solution, we can evaluate the residual wherever we like. We use this to estimate and control the size of the residual at each step. A reliable measure of size when the residual might be only piecewise smooth is obtained by using an integral norm and a quadrature rule of order 8.

By exploiting the very special form of the DDAEs that describe GLTI, we can reliably simulate the step response with performance comparable to that of standard tools for delay-free state-space models (e.g., `step` command in [4]). Indeed, we can simulate the response to arbitrary input signals. More details about the numerical difficulties and how we deal with them are found in [13].

V. Discretization

Approximate Tustin discretization of continuous-time GLTI models can be performed by discretizing the rational part $H(s)$, approximating the delays $e^{-\tau_k s}$ by the nearest discrete delay z^{-m}, and combining the results into a discrete GLTI model. For more accuracy, neglected fractional delays can be further modeled by Thiran filters [16].

Exact ZOH discretization of continuous-time GLTI models in the sense of [6] is not possible in general. However, it can be shown that GLTI models for which

1) D_{22} is strictly upper triangular (up to reordering the delays $\tau_1, \ldots, \tau_N$)
2) there are no state delays, i.e., $B_2\Delta_\tau(I - D_{22}\Delta_\tau)^{-1}C_2 = 0$

admit a ZOH discretization which is itself a GLTI model with a properly augmented state vector. This result generalizes well-known discretization formulas for LTI models with input or output delays [6] and details are omitted for brevity. Note that this subset of GLTI models coincides exactly with the set of affine models (8)-(9) where $A_1 = \ldots = A_M = 0$ (no state delays). To see this, recall from Section III that $B_2\Delta_\tau(I - D_{22}\Delta_\tau)^{-1}C_2 = 0$ implies that $B_2 w(t)$ depends only on $u(t)$ and its past value, whence $A_j = 0$. Conversely, the GLTI realization of any model (8)-(9) satisfies $D_{22} = 0$ and

$$B_2\Delta_\tau(I - D_{22}\Delta_\tau)^{-1}C_2 = B_2\Delta_\tau C_2 = \sum_{k=1}^{M} P_k\Delta_{\theta_k}R_k = \sum_{k=1}^{M} P_k R_k\Delta_{\theta_k} = \sum_{k=1}^{M} A_j\Delta_{\theta_k} = 0$$

provided that $A_1 = \ldots = A_M = 0$.

VI. Software and Example

As mentioned earlier, GLTI models naturally dovetail into existing CACSD packages for LTI modeling and control. In the 6.0 release of MathWorks' Control System Toolbox, most functions have been extended to handle GLTI models. These include functions for modeling such as `series`, `parallel`, and `feedback`, for time and frequency response computation such as `step` and `bode`, for stability analysis such as `margin`, and for discretization such as `c2d`. Dedicated algorithms have been devised and coded in C for efficient time-domain simulation. This extended functionality allows for seamless and *approximation-free* analysis of control systems with delays. Successive loops can be closed without approximation and without losing the ability to simulate time responses. The benefits of these new tools are now illustrated on a process control example.

The example below is inspired by [8] and compares two control strategies, standard PI control and the Smith Predictor, for a first-order plus dead-time plant. The plant transfer function is (see [8] for details):

$$P(s) = e^{-93.9s}\frac{5.6}{40.2s + 1}\ . \tag{11}$$

Note that the dead time is more than twice larger than the time constant. We first consider the standard PI control structure shown in Figure 2, where

$$C(s) = K(1 + \frac{1}{T_i s})\ .$$

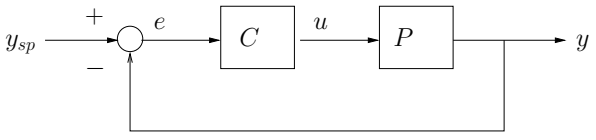

Fig. 2. PI Control Loop.

The large dead time severely limits performance and the loop gain. We settled for the values $K = 0.1$ and $T_i = 100$. To visualize the closed-loop responses in MATLAB, we first construct a GLTI model T_{PI} of the closed-loop transfer function by defining the plant and compensators and closing the loop with the `feedback` command:

```
P = tf(5.6,[40.2 1],...
              'OutputDelay',93.9)
```

```
C = 0.1 * (1 + tf(1,[100 0]))

% Closed-loop transfer [ysp,d] -> y
Tpi = feedback([P*ss(C),1],1,1,1);
```

We then use `step(Tpi,1000)` to plot the closed-loop step response, the resulting plot appearing in Figure 3.

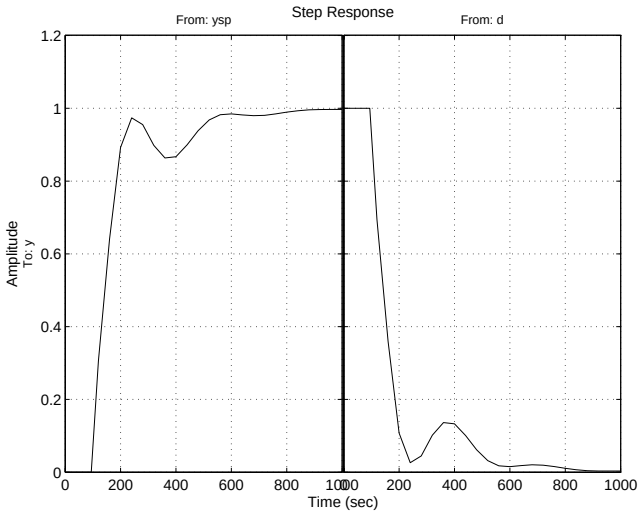

Fig. 3. Step Response of the PI Loop.

Next we consider the Smith Predictor control structure sketched in Figure 4. The Smith Predictor uses an internal model to predict the delay-free response $y_p(t)$ of the plant, and seeks to correct discrepancies between this prediction and the setpoint $y_{sp}(t)$, rather than between the delayed output measurement $y(t)$ and $y_{sp}(t)$. To prevent drifting, an additional compensator $F(s)$ is used to eliminate steady-state drifts and disturbance-induced offsets. See [15] for more details on the Smith Predictor.

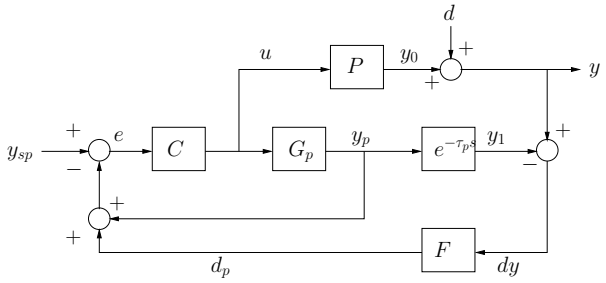

Fig. 4. Smith Predictor.

We first assume that the prediction model $P_p(s) = e^{-\tau_p s}G_p(s)$ matches the plant model $P(s)$ in (11), and use the following compensator settings:

$$C(s) = 0.5(1 + \frac{1}{40s}), \quad F(s) = \frac{1}{20s+1}$$

To compare performance of the PI and Smith Predictor designs, we start by building a GLTI model of the closed-loop transfer function $T(s)$ from (y_{sp}, d) to y. To facilitate the task of connecting all the blocks in Figure 4, we name the I/O signals of each block (including the summation blocks) and use the `connect` command to automatically build the resulting closed-loop model:

```
s = tf('s');

% LTI blocks
P = exp(-93.9*s) * 5.6/(40.2*s+1);
P.InputName = 'u'; P.OutputName = 'y0';

Gp = 5.6/(40.2*s+1);
Gp.InputName = 'u'; Gp.OutputName = 'yp';

Dp = exp(-93.9*s);
Dp.InputName = 'yp'; Dp.OutputName = 'y1';

C = 0.5 * (1 + 1/(40*s));
C.InputName = 'e';  C.OutputName = 'u';

F = 1/(20*s+1);
F.InputName = 'dy'; F.OutputName = 'dp';

% Sum blocks
Sum1 = ss([1,-1,-1],'InputName',...
       {'ysp','yp','dp'},'OutputName','e');
Sum2 = ss([1,1],...
       'InputN',{'d','y0'},'OutputN','y');
Sum3 = ss([1,-1],...
       'InputN',{'y','y1'},'OutputN','dy');

% Build interconnection model
T = connect(P,Gp,Dp,C,F,Sum1,Sum2,Sum3,...
                    {'ysp','d'},'y');
```

Note that `T` is a GLTI model with two internal delays in separate feedback paths. Given this model, the Smith Predictor and PI responses can be compared by `step(T,Tpi,500)` and are shown in Figure 5.

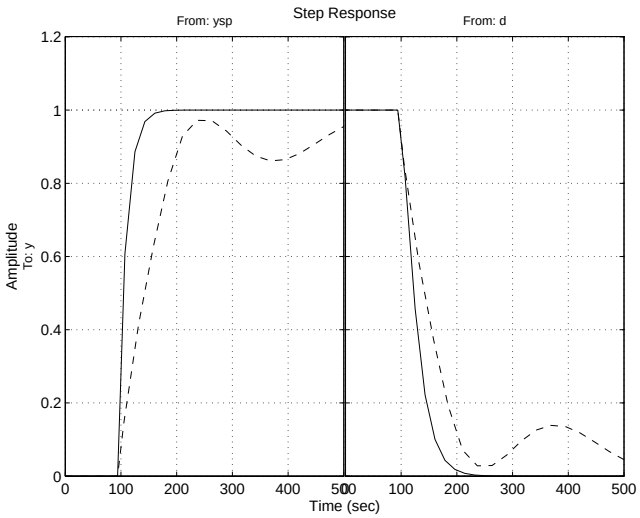

Fig. 5. Step Responses of the Smith Predictor (–) vs. PI (- -).

Finally, robustness to mismatch between the prediction and plant models is easily investigated with these tools. For

example, consider two perturbed plant models

$$P_1(s) = e^{-90s}\frac{5}{38s+1}, \quad P_2(s) = e^{-100s}\frac{6}{42s+1}\ .$$

To assess the Smith predictor robustness when the true plant model is $P_1(s)$ or $P_2(s)$ rather than the prediction model $P(s)$, simply bundle P, P_1, P_2 into an LTI array, rebuild the closed-loop model(s), and replot the step response:

```
P1 = exp(-90*s) * 5/(38*s+1);
P2 = exp(-100*s) * 6/(42*s+1);
Plants = stack(1,P,P1,P2);
T = connect(Plants,Gp,Dp,C,F,...
          Sum1,Sum2,Sum3,{'ysp','d'},'y');

step(T,Tpi,500)
```

The resulting plot in Figure 6 shows a slight performance degradation, but the Smith predictor still retains an edge over the pure PI design.

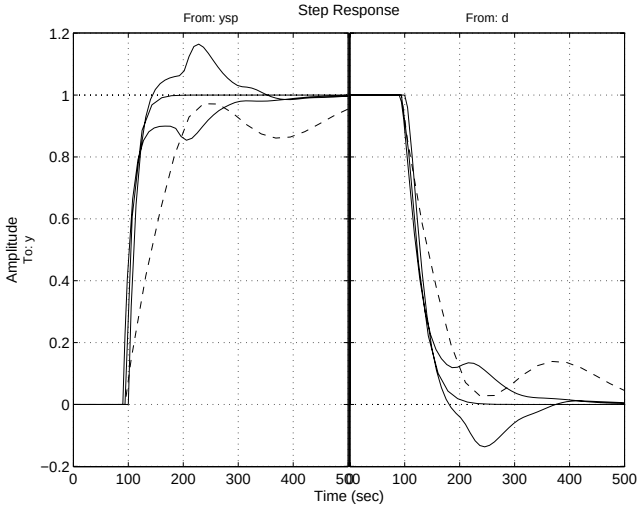

Fig. 6. Robustness of the Smith Predictor (–) to Model Mismatch

The closed-loop frequency response for the nominal and perturbed plants is obtained by

```
bode(T(1,1))
```

and shown in Figure 7.

VII. Conclusion

We have shown that the GLTI representation is highly suitable to computer-aided manipulation and analysis of control systems with delays, regardless of the control structure and number of delays. Most Control System Toolbox functions have been extended to work on GLTI models, all this without additional complexity or new syntax for the user. We hope that these new tools will facilitate the design of control systems with delays and bring new insights into their behavior.

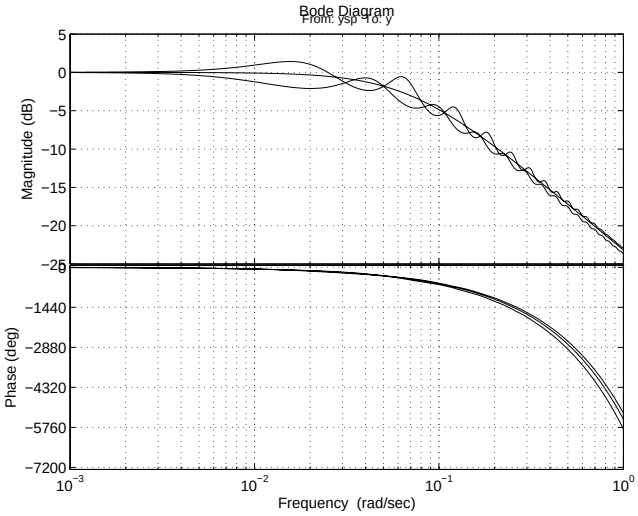

Fig. 7. Closed-Loop Response from y_{sp} to y.

References

[1] U. Ascher and L. Petzold, The numerical solution of delay-differential-algebraic equations of retarded and neutral type, *SIAM J. Numer. Anal.*, 32 (1995) 1635–1657.

[2] U. Ascher and L. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, Philadelphia, 1998.

[3] *Control System Professional*, Wolfram Research, Champaign, 2003.

[4] *Control System Toolbox*, MathWorks Inc., Natick, 2000.

[5] W.H. Enright and H. Hayashi, A delay differential equation solver based on a continuous RungeKutta method with defect control, *Numerical Algorithms*, 16, 1997, pp. 349364.

[6] G.F. Franklin, J.D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, Prentice Hall, 2002.

[7] M. Gza, *DifEqu: Solver for Ordinary, Functional and Partial Differential Equations*, http://www.math.u-szeged.hu/ makay, Hungary.

[8] A. Ingimundarson and T. Hagglund, Robust Tuning Procedures of dead-time compensating controllers, *Control Engineering Practice*, 9, 2001, pp. 1195-1208.

[9] A.J. Laub, Efficient Multivariable Frequency Response Computations, *IEEE Trans. Aut. Contr.*, AC-26 (1981), pp. 407-408.

[10] E. Hairer and G. Wanner, *RETARD: Software for Delay Differential Equations*, http://www.unige.ch/math/folks/hairer/software.html, Switzerland.

[11] A. Kim, W.H. Kwon, and L. Volkanin, *Time-Delay System Toolbox*, http://fde.usaaa.ru, Russia, 2001.

[12] L.F. Shampine, I. Gladwell, and S. Thompson, *Solving ODEs with* MATLAB, Cambridge Univ. Press, New York, 2003.

[13] L.F. Shampine and P. Gahinet, *DDAEs in Control Theory*, Southern Methodist University, Dallas, TX, 2003.

[14] S. Skogestad and Ian Postlethwaite, *Multivariable Feedback Control*, John Wiley, 1996.

[15] O.J.M Smith, Closer Control of Loops with Dead Time, *Chemical Engineering Process*, 53, 1957, pp. 217-219.

[16] J.-P. Thiran, Recursive Digital Filters with Maximally Flat Group Delay, *IEEE. Trans. Circ. Theory*, 18, 1971, pp. 659-664.

[17] *Xmath Control Design Module*, National Instruments Corporation, Austin, 2002.