Predicting Time-Delays under Real-Time Scheduling for Linear Model Predictive Control

Zhenwu Shi and Fumin Zhang, Georgia Institute of Technology

Abstract— **Digital implementations of model predictive controllers as tasks on an embedded computer introduce time-delays in feedback control loops. Time-delays may cause performance degradation or even instability, but are difficult to predict when multiple tasks are executing concurrently on the same processor under real-time scheduling. In this paper, we propose an analytical timing model for real-time scheduling. Based on this model, we are able to perform an online prediction of time-delays caused by real-time scheduling. Then, a model predictive controller is designed on the basis of a process model that contains the predicted time delays. Example designs are demonstrated to show the effectiveness of the proposed method.**

I. Introduction

Cyber-physical system (CPS) theory studies the tight integration of computing and physical process [1]–[3]. Within this scope, we develop results for model predictive controllers that are implemented as tasks on an embedded computer running real-time operating system (RTOS). When multiple tasks are running concurrently in RTOS, the processor of the embedded computer can only execute one task at a time. The contention for the processor among multiple tasks is resolved by real-time scheduling, which determines the execution sequence of tasks.

Real-time scheduling is an active research area in computer science. From the 1970s, significant progress has been made in the real-time scheduling theory [4]–[6]. In recent years, the interaction between real-time scheduling and control design has received interest in the literature and a considerable amount of results have been reported [7], [8]. However, in order to fully exploit the benefit of integrating feedback control and real-time scheduling, many research issues are still open. As discussed in [9], one of the fundamental research issues is to establish an analytical timing model of real-time scheduling for control purpose, which may lead to better integration of control theory and real-time scheduling.

In this paper, we study digital implementations of MPC (Model Predictive Control) controllers as tasks in RTOS. The basic idea of MPC controller is to iteratively use a process model of the physical system to predict and optimize future system behaviors [10]. Digital implementations of MPC controllers introduce variable time-delays between the sampling and actuation due to real-time scheduling of multiple tasks in RTOS. Such variable time-delays may degrade the performance of MPC designed without considering this latency and even destabilize the whole system.

Two major approaches that address time-delays in MPC design have been reported in the literature. The first approach is to treat the variable time-delay as an uncertain parameter of the system and design MPC controllers that are robust to this uncertainty [11]–[13]. The second approach is to develop a process model with time-delays, and then design a MPC controller on the basis of this process model. However, the second approach assumes that the time-delays are always predictable. Due to lack of an analytical timing model for real-time scheduling, predictions of the time-delays are usually performed at the design stage through offline methods. One common offline method is to maintain a constant delay between the sampling and actuation by means of proper system architecture design [9]. To guarantee that the actuation takes place after the control command is computed, this method must choose the constant delay larger than the worst-case reponse time of the corresponding MPC task in RTOS, which is calculated at the design stage. Another offline method is to predict variable time-delays at the design stage through intensive offline simulations of real-time scheduling and then save up those offline predictions in a look-up table for online use [14]. However, these offline methods have a primary drawback of relying on the nominal characteristics of tasks in RTOS. If the actual task characteristics deviate from their nominal values due to disturances that happen in RTOS at runtime, the offline predictions of time-delays may not reflect the actual time-delays. Using the imprecise offline predictions leads to an inaccurate process model, which compromises the effect of the second approach.

Following the second approach, we establish an analytical timing model for real-time scheduling. Based on this analytical timing model, we can perform online prediction of time-delays at runtime, which uses the actual characteristics of tasks in RTOS. This model allows the performance of MPC controllers to be improved when disturbance in RTOS frequently happens.

The rest of this paper is organized as follows. In Section II, we establish an analytical timing model for real-time scheduling. In Section III, we formulate a problem of designing a MPC controller based on the analytical timing model for real-time scheduling. In Section IV, an example is demonstrated to show the effectiveness of our analytical

The research is supported by ONR grants N00014-08-1-1007, N00014-09-1-1074, and N00014-10-10712 (YIP), and NSF grants ECCS-0841195 (CAREER), CNS-0931576, and ECCS-1056253.

Zhenwu Shi and Fumin Zhang are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332, USA zwshi, fumin@gatech.edu

978-1-4673-5288-8/13/$31.00 ©2013 IEEE

timing model based MPC controller design.

II. Real-time Scheduling Model

Real-time scheduling studies the scheduled behavior of multiple tasks that are running concurrently in RTOS. In this section, we establish an analytical timing model for real-time scheduling. Based on this analytical timing model, we can predict future scheduled behavior of tasks at runtime.

A. Task Characteristics

We consider a set of independent tasks $\{\tau_1, \cdots, \tau_N\}$ that are concurrently running in RTOS. A task in RTOS is comprised of a sequence of instances, where each instance corresponds to one invocation of that task. For any task τ_n in RTOS, we define the effective instance of τ_n and then characterize τ_n on the basis of effective instance.

Definition 2.1: At any time t, the *effective* instance of τ_n is an instance that arrives before time t and expires after t.

Definition 2.2: τ_n is characterized by two functions $C_n(t)$ and $T_n(t)$. $C_n(t)$ represents the *actual computation time* of the effective instance of τ_n at time t and $T_n(t)$ represents the *actual inter-arrival time* between the effective instance of τ_n at time t and the next instance of τ_n.

We assume that an instance of τ_n expires when the next instance of τ_n arrives. Therefore, $T_n(t)$ also denotes the *actual relative deadline* of the effective instance of τ_n at time t.

Remark 2.3: Actual task characteristics $\{C_n(t)\}_{n=1}^{N}$ and $\{T_n(t)\}_{n=1}^{N}$ may deviate from the nominal characteristics denoted as $\{C_n^*(t)\}_{n=1}^{N^*}$ and $\{T_n^*(t)\}_{n=1}^{N^*}$, when disturbance in RTOS frequently happens at runtime.

B. State Variables

To describe the scheduled behaviors of $\{\tau_1, \cdots, \tau_N\}$ at any time t, we define a state vector $Z(t) = [Q(t), S(t), O(t)]$ and an auxiliary variable $R(t)$. With the state vector well defined, we can later represent the dynamics of real-time scheduling through the evolution of $Z(t)$.

Definition 2.4: The *dynamic deadline* $Q(t)$ is the first state variable in $Z(t)$. It is defined as a vector $Q(t) = [q_1(t), \cdots, q_N(t)]$, where $q_n(t)$, for $n = 1, 2, \cdots, N$, denotes how long after t the effective instance of τ_n will expire.

Definition 2.5: The *spare* $S(t)$ is the second state variable in $Z(t)$. It is defined as a vector $S(t) = [s_1(t), \cdots, s_N(t)]$, where $s_n(t)$, for $n = 1, 2, \cdots, N$, denotes how much time of the CPU is available for computing the effective instance of τ_n since its time of arrival till time t.

Definition 2.6: The *delay* $O(t)$ is the third state variable in $Z(t)$. It is defined as a vector $O(t) = [o_1(t), ..., o_N(t)]$, where $o_n(t)$, for $n = 1, 2, ..., N$, denotes how much time has been delayed for computing the effective instance of τ_n since its time of arrival till time t.

Definition 2.7: The *residue* $R(t)$ is an auxiliary variable that is defined as a vector $R(t) = [r_1(t), ..., r_N(t)]$, where $r_n(t)$, for $n = 1, 2, ..., N$, denotes the remaining computation time required *after* time t by the current instance of τ_n.

For any task τ_n, the residue $r_n(t)$ can be expressed as

$$r_n(t) = \max\{0, C_n(t) - s_n(t)\} \tag{1}$$

which implies that $r_n(t)$ solely depends on $s_n(t)$. Therefore, $R(t)$ is not a state variable but an auxilary variable solely depending on $S(t)$. As we will see in Section II-D, $R(t)$ is convenient to use for developing the evolution of $Z(t)$.

C. State Transition Rules

From Definition 2.4 to Definition 2.6, we can see that the state vector $Z(t) = [Q(t), S(t), R(t)]$ evolves as the execution of $\{\tau_1, \cdots, \tau_N\}$ continues. Since the execution of $\{\tau_1, \cdots, \tau_N\}$ is determined by the real-time scheduling algorithm, the evolution of $Z(t)$ must follow the same rules.

In this section, we discuss how to rigorously represent a real-time scheduling algorithm by an analytical function $hp(n, t)$, which is the rules of state transitions.

Definition 2.8: For a task τ_n, $hp(n, t)$ represents the indices of tasks, whose priorities assigned by the real-time scheduling algorithm is higher than that of τ_n at time t.

The following example demonstrates the use of $hp(n, t)$ to represent a real-time scheduling algorithm.

Example 1: Suppose a set of tasks $\{\tau_1, \cdots, \tau_N\}$ are running in RTOS. The earliest deadline first scheduling algorithm (EDF) is applied to determine the sequence of execution of tasks. EDF assigns priorities to tasks on the basis of their deadlines: the closer the deadline is, the higher the task's priority is.

For any task τ_n, the expression of $hp(n, t)$ can be written as

$$\begin{aligned} &hp(n,t) = \\ &\{i \mid q_i(t) < q_n(t)\} \cup \{i \mid \tau_i \; r_i(t) < C_i(t)\}. \end{aligned} \tag{2}$$

where the first term represents the indices of tasks that are assigned higher priority than τ_n by EDF; and the second term represents the indices of non-preemptive tasks that are currently being executed by the CPU.

D. Evolution of State Variables

In this section, we discuss how the state vector evolves according to the state transition rules. At any time t_a^-, given the initial state vector $Z(t_a^-)$, and actual task characteristics $\{C_n(t)\}_{n=1}^{N}$ and $\{T_n(t)\}_{n=1}^{N}$ for $t \in [t_a, t_b]$, we are interested in the evolution of $Z(t)$ within $[t_a, t_b]$. Here, t_a^- denotes a time point that is arbitrarily close but less than t_a and this notation will be used throughout this paper.

To simplify the derivation of the evolution of $Z(t)$ within $[t_a, t_b]$, we further divide $[t_a, t_b]$ into a series of consecutive sub-intervals denoted as $[t_s(w), t_s(w+1)^-]$, within which task instances only arrive at $t_s(w)$ but not at any other time point within $[t_s(w), t_s(w+1)^-]$. The reason for such division is that the evolution of $Z(t)$ within each sub-interval $[t_s(w), t_s(w+1)^-]$ is relatively easier to be modeled, as will

be shown below. Then, the models in individual sub-intervals can be concatenated together to constitute the more complex model for the evolution of $Z(t)$ within $[t_a, t_b]$.

Consider a sub-interval $[t_s(w), t_s(w+1)^-]$. For any task τ_n in RTOS (where $n = 1, \cdots, N$), the evolution of $q_n(t)$, $s_n(t)$ and $o_n(t)$ within $[t_s(w), t_s(w+1)^-]$ can be derived through two steps: from $t_s(w)^-$ to $t_s(w)$, and then from $t_s(w)$ to any time $t \in [t_s(w), t_s(w+1)^-]$.

From $t_s(w)^-$ to $t_s(w)$: First, we discuss the evolution of $[q_n(t), s_n(t), o_n(t)]$ from $t_s(w)^-$ to $t_s(w)$.

(1) if no instance of τ_n arrives at $t_s(w)$, i.e. $q_n(t_s(w)^-) > 0$. In this case, $[q_n(t), s_n(t), o_n(t)]$ hold their values from $t_s(w)^-$ to $t_s(w)$, i.e.

$$\begin{aligned} &\text{if } \ q_n(t_s(w)^-) > 0, \text{we have :} \\ &[\ q_n(t_s(w)) \ \ , \ s_n(t_s(w)) \ \ , \ o_n(t_s(w)) \ \ ] \\ &= [\ q_n(t_s(w)^-)\ ,\ s_n(t_s(w)^-)\ ,\ o_n(t_s(w)^-)\ ] \end{aligned} \tag{3}$$

(2) if a new instance of τ_n arrives at $t_s(w)$ and the old instance of τ_n has finished its computation before $t_s(w)$, i.e. $q_n(t_s(w)^-) = 0$ and $r_n(t_s(w)^-) = 0$. In this case, the new instance of τ_n replaces the old instance of τ_n. Therefore, $[q_n(t), s_n(t), o_n(t)]$ are updated according to the new instance.

$$\begin{aligned} &\text{if } \ q_n(t_s(w)^-) = 0 \text{ and } r_n(t_s(w)^-) > 0, \text{we have :} \\ &[\ q_n(t_s(w))\ ,\ s_n(t_s(w)),\ o_n(t_s(w))\ ] \\ &= [\ T_n(t_s(w))\ ,\quad 0\quad ,\quad 0\quad] \end{aligned} \tag{4}$$

(3) if a new instance of τ_n arrives at $t_s(w)$ but the old instance of τ_n has not finished its computation before $t_s(w)$, i.e. $q_n(t_s(w)^-) = 0$ and $r_n(t_s(w)^-) > 0$. In this case, an overload situation happens. To handle the overload situation, we use a simple yet efficient method of skipping any new instance of τ_n until the old instance of τ_n finishes its computation. Therefore, the spare $s_n(t_s(w))$ and delay $o_n(t_s(w))$ hold their values from $t_s(w)^-$ to $t_s(w)$. Moreover, the dynamic deadline $q_n(t_s(w))$ is reset to the relative deadline $T_n(t_s(w))$ of the dismissed instance., i.e.

$$\begin{aligned} &\text{if } \ q_n(t_s(w)^-) = 0 \text{ and } r_n(t_s(w)^-) > 0, \text{we have :} \\ &[\ q_n(t_s(w)) \ \ , \ s_n(t_s(w)) \ \ , \ o_n(t_s(w)) \ \ ] \\ &= [\ T_n(t_s(w))\ ,\ s_n(t_s(w)^-)\ ,\ o_n(t_s(w)^-)\ ] \end{aligned} \tag{5}$$

In summary, (3), (4) and (5) constitute the evolution of $[q_n(t), s_n(t), o_n(t)]$ from $t(w)^-$ to $t(w)$.

From $t_s(w)$ to $t \in [t_s(w), t_s(w+1)^-]$: Next, we discuss the evolution of $[q_n(t), s_n(t), o_n(t)]$ from $t_s(w)$ to $t \in [t_s(w), t_s(w+1)^-]$.

(1) For the dynamic deadline $q_n(t)$, according to Definition 2.4, we know that the effective instance of τ_n at time $t_s(w)$ will expire at $t_s(w)+q_n(t_s(w))$ and the effective instance of τ_n at time t will expire at $t+q_n(t)$. Since no new instance of τ_n arrives within $[t_s(w), t_s(w+1)^-]$, the effective instance at $t(w)$ and the effective instance at t are one and the same. Therefore, $t + q_n(t)$ and $t_s(w) + q_n(t_s(w))$ represent the same time point, i.e. $t + q_n(t) = t_s(w) + q_n(t_s(w))$. We can write the equation for $q_n(t)$ as

$$q_n(t) = t_s(w) + q_n(t_s(w)) - t \tag{6}$$

(2) For the spare $s_n(t)$, we know that the computation of the effective instance of τ_n is preempted until the computation of all higher priority task instances are completed. Then, the amount of time within $[t_s(w), t]$ that is available to compute the effective instance of τ_n is

$$\max\{0, t - t_s(w) - \sum_{i \in hp(n, t_s(w))} r_i(t_s(w))\}.$$

where $\sum_{i \in hp(n,t_s(w))} r_i(t_s(w))$ denotes the total time that is allocated to compute task instances with higher priories than the effective instance of τ_n. The function max guarantees that it will not give a negative result. Therefore, the amount of time that is available to compute the effective instance of τ_n from its arrival time to t is

$$s_n(t) = s_n(t_s(w)) + \max\{0, t - t_s(w) - \sum_{i \in hp(n, t_s(w))} r_i(t_s(w))\} \tag{7}$$

(3) For the delay $o_n(t)$, if the effective instance of τ_n has finished its computation before $t_s(w)$, i.e. $r_n(t_s(w)) = 0$, the delay $o_n(t)$ will not increase within $[t_s(w), t_s(w+1)^-]$,

$$\text{if } \ r_n(t_s(w)) = 0 : o_n(t) = o_n(t_s(w)) \tag{8}$$

On the other hand, if the effective instance of τ_n has finished its computation before $t_s(w)$, i.e. $r_n(t_s(w)) \neq 0$, then the value of $o_n(t)$ will continue to increase until the effective instance of τ_n finishes its computation or time t is passed, whichever occures first. Therefore, we have that

$$\begin{aligned} &\text{if } \ r_n(t_s(w)) \neq 0 : \\ &o_n(t) = o_n(t_s(w)) + \min\{\sum_{i \in hp(n, t_s(w))} r_i(t_s(w)) + r_n(t_s(w)), t - t_s(w)\} \end{aligned} \tag{9}$$

According to (8) and (9), the evolution of $o_n(t)$ from $t_s(w)$ to t can be written in a compact form as follows

$$\begin{aligned} &o_n(t) = o_n(t_s(w)) + \\ &\mathsf{sgn}(r_n(t_s(w)))\ \min\{\sum_{i \in hp(n, t_s(w))} r_i(t_s(w)) + r_n(t_s(w)), t - t_s(w)\} \end{aligned} \tag{10}$$

In summary, (6), (7) and (10) constitue the evolution for the state variables from $t_s(w)$ to $t \in [t_s(w), t_s(w+1)^-]$.

E. Analytical Timing Model

As discussed above, equation (3), (4), (5), (6), (7) and (10) together constitute the evolution of $[q_n(t), s_n(t), o_n(t)]$ within a sub-interval $[t_s(w), t_s(w+1)^-]$. Applying the above equations to all tasks $\{\tau_1, \cdots, \tau_N\}$ in RTOS and putting results together, we have the evolution of $Z(t) = [Q(t), S(t), O(t)]$ within $[t_s(w), t_s(w+1)^-]$. Then, concatenating the evolution of $Z(t)$ within all sub-intervals $[t_s(w), t_s(w+1)^-] \in [t_a, t_b]$, we have an analytical timing model for the evolution of $Z(t)$ within $[t_a, t_b]$, which is denoted as H.

In summary, at any time t_a^-, given the initial state vector $Z(t_a^-)$, and actual task characteristics $\{C_n(t)\}_{n=1}^N$ and $\{T_n(t)\}_{n=1}^N$ for $t \in [t_a^-, t_b]$, we have the evolution of $Z(t)$ for any $t \in [t_a^-, t_b]$ as

$$Z(t) = H(Z(t_a^-), \{C_n(t)\}_{n=1}^N, \{T_n(t)\}_{n=1}^N) \tag{11}$$

where H represents the analytical timing model for the real-time scheduling.

III. Timing-Model based MPC controller Design

We consider a CPS with several independent feedback control loops. Each feedback control loop consists of a physical plant, a sensor, a MPC controller and an actuator. The MPC controller in each feedback control loop is implemented as a task in RTOS. The execution of multiple tasks in RTOS is determined by real-time scheduling, as discussed in Section II.

A. MPC Design

Consider the n-th feedback control loop in CPS. We assume that the physical plant is a Multiple-Input and Multiple-Output (MIMO) Linear Time-Invariant System (LTIS), i.e.

$$\begin{aligned} \dot{x}_n(t) &= Ax_n(t) + Bu_n(t) \\ y_n(t) &= Cx_n(t) \end{aligned} \tag{12}$$

where $u_n(t)$ is the continuous time control signal applied on the plant; $y_n(t)$ is the plant output; $x_n(t)$ is the state vector; and A, B and C are state matrix of proper dimension. Moreover, we assume that $u_n(t)$ is obtained from the computed discrete control command $\mu_n[k]$ through zero-th order hold, i.e.

$$u_n(t) = \mu_n[k], \text{ for } t \in [\sigma_n[k] + \delta_n[k], \sigma_n[k+1]) \tag{13}$$

where $\sigma_n[k]$ represents a sampling time and $\delta_n[k]$ a time-delay between sampling and actuation.

At any i-th sampling time $\sigma_n[i]$, the goal of MPC is to find an optimal control signal that brings the future system behavior as close as possible to the reference trajectory within a prediction horizon $[\sigma_n[i], \sigma_n[i] + T_p]$, where T_p is the length of the prediction horizon. This goal is achieved through optimizing the following objective function [10]

$$\begin{aligned} &J(u_n(\sigma_n[i]+\tau)) = \\ &\int_0^{T_p} (\gamma_n(\sigma_n[i]+\tau) - y_n(\sigma_n[i]+\tau))^T V (\gamma_n(\sigma_n[i]+\tau) - \\ &y_n(\sigma_n[i]+\tau)) d\tau + \int_0^{T_p} \dot{u}_n(\sigma_n[i]+\tau)^T W \dot{u}_n(\sigma_n[i]+\tau)\, d\tau \end{aligned} \tag{14}$$

where $\gamma_n(\sigma_n[i]+\tau)$ is the reference trajectory, $y_n(\sigma_n[i]+\tau)$ is the future system output, $\dot{u}_n(\sigma_n[i]+\tau)$ is the derivative of control signal, and V and W are diagnoal weight matrices.

Since MPC iteratively uses this process model in equation (12) and (13) to predict and optimize future system behaviors, the accurate prediction of $\delta_n[k]$ is crucial to the performance of MPC controllers.

B. Online Prediction of $\delta_n[k]$

Consider the n-th feedback control loop in CPS. At any sampling time $\sigma_n[i]$, given the current state vector $Z(\sigma_n[i])$ and the future task characteristics $\{C_n(t)\}_{n=1}^N$ and $\{T_n(t)\}_{n=1}^N$, we can predict the future state vector as

$$Z(t) = H(Z(\sigma_n[i]), \{C_n(t)\}_{n=1}^N, \{T_n(t)\}_{n=1}^N) \tag{15}$$

where the analytical timing model for the real-time scheduling in equation (11) is applied. Moreover, according to Definition 2.6, we know that the time-delay $\delta_n[k]$ in any n-th feedback control loop can be represented as

$$\delta_n[k] = o_n(\sigma_n[k+1]^-) \tag{16}$$

where $o_n(t)$ is an element of $O(t)$ and $O(t)$ belongs to $Z(t)$.

Remark 3.1: $\{C_n(t)\}_{n=1}^N$ and $\{T_n(t)\}_{n=1}^N$ in equation (15) is the actual task characteristics, which contain latest available information of online perturbations that happen in RTOS at runtime. If we use the nominal task characteristics $\{C_n^*(t)\}_{n=1}^{N^*}$ and $\{T_n^*(t)\}_{n=1}^{N^*}$ as the input to equation (15), the online prediction of $\delta_n[k]$ in (16) will be same as the offline prediction.

Remark 3.2: Running online prediction incurs extra computation overhead. However, since the timing model is composed of simple mathematic equations, such computation overhead is negligible as compared with the computation of MPC, which often involves quardratic programming that requires large computation time.

C. Optimization Problem

From above analysis, we know that in the n-th feedback control loop, the timing model based MPC controller design at any i-th sampling time $\sigma_n[i]$ can be formulated as follows

$$\text{Given } x_n(\sigma_n[i]), Z(\sigma_n[i]), \quad \min_{u_n(\sigma_n[i]+\tau)} \quad J(u_n(\sigma_n[i]+\tau)) \tag{17}$$

s.t.

(1)
$$u_{\min} \le u_n(t) \le u_{\max} \tag{17.a}$$

(2)
$$\begin{aligned} \dot{x}_n(t) &= Ax_n(t) + Bu_n(t) \\ y_n(t) &= Cx_n(t) \end{aligned} \tag{17.b}$$

(3)
$$u_n(t) = \mu_n[k], \text{ for } t \in [\sigma_n[k] + \delta_n[k], \sigma_n[k+1]) \tag{17.c}$$

(4)
$$Z(t) = H(Z(\sigma_n[i]), \{C_n(t)\}_{n=1}^N, \{T_n(t)\}_{n=1}^N), \tag{17.d}$$

(5)
$$\delta_n[k] = o_n(\sigma_n[k+1]), \tag{17.e}$$

Note that equation (17.d) and (17.e) clearly show that our MPC design uses online predictions of time-delays, which is based on the analytical timing model of real-time scheduling. The optimization problem in (17) can be solved using the quardratic programming discussed in [10].

IV. Simulation

Consider a CPS with three physical plants. Each physical plant is represented by a state space model as follows

$$\dot{x}_n(t) = \begin{bmatrix} 0 & 1 \\ a & b \end{bmatrix} x_n(t) + \begin{bmatrix} 0 \\ c \end{bmatrix} u_n(t)$$
$$y_n(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x_n(t) \quad (18)$$

where $u_n(t)$ is the control input and $y_n(t)$ is the plant output. We assume that the three physical plants have different coefficients as $[a_1, a_2, a_3] = [98, 65, 44]$, $[b_1, b_2, b_3] = [120, 52, 30]$ and $[c_1, c_2, c_3] = [20, 13, 10]$. The control purpose is to make $y_n(t)$ track a given reference trajectory $\gamma_n(t)$ as close as possible, under the constraint that $-4 \leq u_n(t) \leq 4$. To meet this purpose, we design three MPC controllers that are implemented as $\{\tau_1, \tau_2, \tau_3\}$ in RTOS. We assume that $\{\tau_1, \tau_2, \tau_3\}$ are all preemptive and have the nominal task characteristics as $[C_1^*(t), C_2^*(t), C_3^*(t)] = [50, 50, 50]$ms and $[T_1^*(t), T_2^*(t), T_3^*(t)] = [200, 250, 300]$ms. All tasks in RTOS are scheduled under the Earliest Deadline First (EDF) scheduling algorithm.

At run time, we assume that RTOS is affected by two types of disturbances. One is the perturbation on the nominal computation times of MPC tasks, which is assumed to be a stochastic process with its value at each time point t being a random variable uniformly distributed within $[0, 10]$ms. The other type of disturbance is the arrival of two unexpected tasks $\{\tau_4, \tau_5\}$, with characteristics $[C_4(t), C_5(t)] = [52.5, 47.5]$ms and $[T_4(t), T_5(t)] = [200, 250]$ms. The online perturbations in RTOS make the actual task characteristics deviates from the nominal task characteristics.

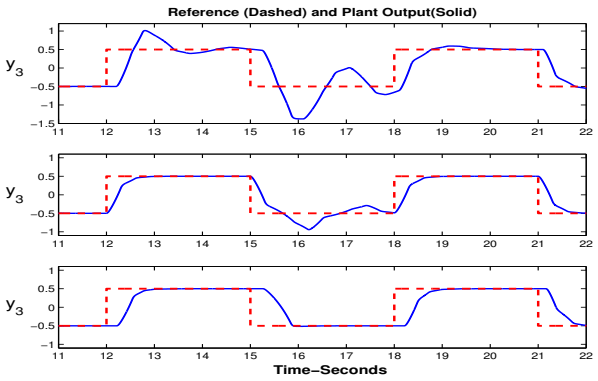

Fig. 1. Performance of the Third Physical Plant

In this paper, we compare the performance of MPC controllers under three different strategies of predicting time-delays $\delta_n[k]$ in the process model. In the first strategy $\delta_n[k]$ is made constant by choosing a regular interval between sampling and actuation at the design stage, as the worst case reponse time of the corresponding MPC task in RTOS. In the second strategy, $\delta_n[k]$ is predicted at the design stage through extensive offline simulation and then saved up in a look-up table for the online use. In the third strategy, $\delta_n[k]$ is predicted online on the basis of the analytical timing model for real-time scheduling. Fig. 1 shows the performance of the third physical plant under three different strategies of predicting time-delays. The other two physical plants has the similar performance as the third physical plant.

As we can see, the plant performance is poor under the first and second strategies. This is because the first two strategies predict time-delays at the design stage using the nominal task characteristics $\{C_n^*(t)\}_{n=1}^3$ and $\{T_n^*(t)\}_{n=1}^3$, which may deviate from the actual task characteristics in the presence of online perturbations. Using inaccurate offline predictions of time-delays leads to an imprecise process model, which results in less desirable performance of MPC. On the other hand, the plant output tracks the reference much better in the third strategy, where time-delays are predicted online using the actual task characteristics $\{C_n(t)\}_{n=1}^5$ and $\{T_n(t)\}_{n=1}^5$.

V. Conclusion and Future Work

This paper proposes a Model Predictive Control design approach for Cyber-physical System under time-delays caused by real-time scheduling. The time-delays are predicted through an online method, which is based on an analytical timing model for real-time scheduling. The simulations show that the effectiveness of timing model based MPC design in the presence of unpredictable disturbance in RTOS.

References

[1] E. A. Lee, "Cyber Physical Systems : Design Challenges," in *Proc. of IEEE Symposium on Object Oriented Real-Time Distributed Computing*, 2008, pp. 363–369.

[2] W. Wolf, "Cyber-physical systems," *Computers*, vol. 42, no. 3, pp. 88–89, 2009.

[3] F. Zhang, Z. Shi, and W. Wolf, "A Dynamic Battery Model for Co-design in Cyber-Physical Systems," in *Proceedings of 29th IEEE International Conference on Distributed Computing Systems Workshops*, 2009, pp. 51–56.

[4] L. Sha, T. Abdelzaher, K.-E. Å rzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real Time Scheduling Theory: A Historical Perspective," *Real-Time Systems*, vol. 28, pp. 101–155, 2004.

[5] T. Chantem, X. S. Hu, and M. D. Lemmon, "Generalized Elastic Scheduling for Real-Time Tasks," *IEEE Transactions on Computers*, vol. 58, no. 4, pp. 480–495, Apr. 2009.

[6] F. Zhang, Z. Shi, and S. Mukhopadhyay, "Robustness Analysis for Battery Supported Cyber-Physical Systems," *ACM Transaction on Embedded Computing Systems, in print*, 2012.

[7] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "Trade-Off Analysis of Real-Time Control Performance and Schedulability," *Real-Time Systems*, vol. 21, pp. 199–217, 2001.

[8] M. Mazo, A. Anta, and P. Tabuada, "An ISS Self-triggered Implementation of Linear Controllers," *Automatica*, vol. 46, pp. 1310–1314, 2010.

[9] F. Xia and Y. Sun, *Control and Scheduling Codesign*. Springer, 2008.

[10] L. Wang, *Model Predictive Control System Design and Implementation Using MATLAB*. Springer, 2009.

[11] G. P. Liu, J. X. Mu, D. Rees, and S. C. Chai, "Design and stability analysis of networked control systems with random communication time delay using the modified MPC," *International Journal of Control*, vol. 79, no. 4, pp. 288–297, Apr. 2006.

[12] S. C. Jeong and P. Park, "Constrained MPC algorithm for uncertain time-varying systems with state-delay," *IEEE Transactions on Automatic Control*, vol. 50, no. 2, pp. 257–263, Feb. 2005.

[13] Y. Xia and Y. Jia, "Robust Sliding-Mode Control for Uncertain Time-Delay Systems An LMI Approach," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 1086–1092, 2003.

[14] P. Martí, J. M. Fuertes, G. Fohler, and H. Ramamritham, "Jitter Compensation for Real-Time Control Systems," in *Proceedings of IEEE International Conference on Real-Time Systems Symposium*, 2001, pp. 39–48.