Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

# Synchronization of State Based Control Processes with Delayed and Asynchronous Measurements

Haitham Hindi, Lara S. Crawford, and Markus P.J. Fromherz
Palo Alto Research Center (PARC)
Palo Alto, California
{hhindi,lcrawford,fromherz}@parc.com

***Abstract*— This paper addresses the problem of controller state synchronization in a networked control system with distributed sensing and actuation, where actuators must hand off and switch controllers "on the fly" as they go from performing one task to another, in the presence of fixed communication delays and asynchronous measurements. We present a technique that enables new controllers to seamlessly join and leave a task, by encapsulating the controller in a finite state machine that handles the synchronization. We also discuss issues related to the real-time implementation of this technique and we finish with a demonstration on an example from the document printing domain.**

## I. Introduction

This paper addresses the problem of controller state synchronization in a networked control system. This problem arises quite naturally in a distributed sensing and distributed actuation setting. Consider the scenario where objects are being handled with several actuators, each with its own controller, and where sensor measurements arrive asynchronously and are broadcasted to the actuators over a network with delays. Somehow, the actuators need to switch controllers "on the fly" as they go from handling one object to another. Also, as a new actuator joins a new group that is already handling an object, its controller must be synchronized with the actions of the others in the group, toward achieving the same overall objective as the whole group. Related work on synchronization has recently appeared in other areas, notably networked computer games (see Mauve [10] or Owada and Asahara [12]). Other work on networked control systems with delays can be found in Nilsson [11], Luck and Ray [9], and Lincoln and Bernhardsson [8]).

In the sections to follow, we first motivate this problem by a concrete application from document printing system control. Next the problem is defined formally and then solved for both the synchronous and asynchronous detection scenario. We then present a state machine implementation which describes a practical real-time software implementation. Finally, the method is demonstrated on a numerical example. Since this technique is quite likely to be useful to practitioners from other fields, this paper is written in a rather expository style – without compromising rigor or precision.

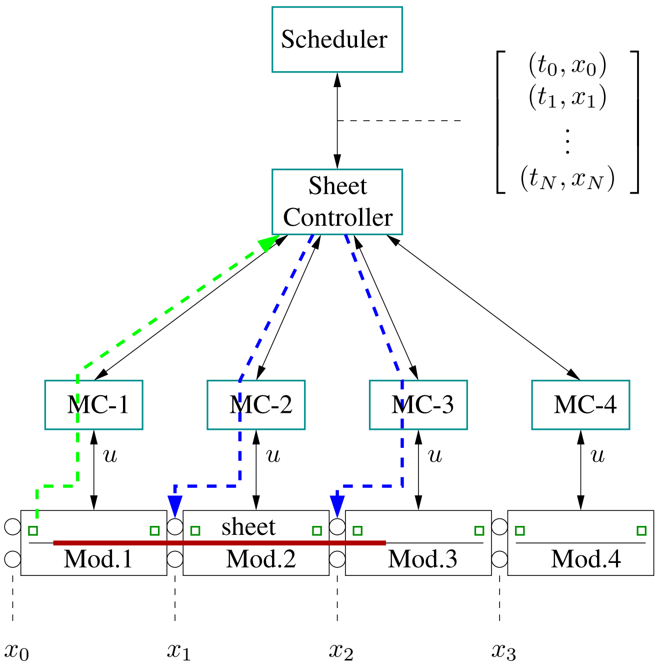

Fig. 1. Hierarchical networked control system. The scheduler specifies way points, the sheet controller issues corresponding commands to the appropriate module controllers over a network (not shown). The module controllers apply the control signals to the rollers ("nips"), which move the paper. The paper is in contact with more than one nip at any given time. The thin lines show these communication pathways. The sheet controller also relays sensor (edge detection) information to appropriate module controllers via a publish-subscribe protocol. A specific example of this is shown in thicker lines: a sensor (small square) generates information that is passed up to the sheet controller, which then relays it to all modules that need the feedback.

## II. Motivation

Consider the hierarchical networked document printing paper path control system shown in Figure 1. The scheduler specifies a series of way points which are desired arrival times and locations for the leading edge of the sheet of paper (see Figure 2). This information is then passed to a sheet controller which passes it to the relevant module controllers over a network (not shown). The sheet controller also relays the sensor information to appropriate module controllers via a publish-subscribe protocol. The sensors are edge detection sensors and since the exact arrival time of the sheet is not known in advance (due to noise disturbances), the sensing

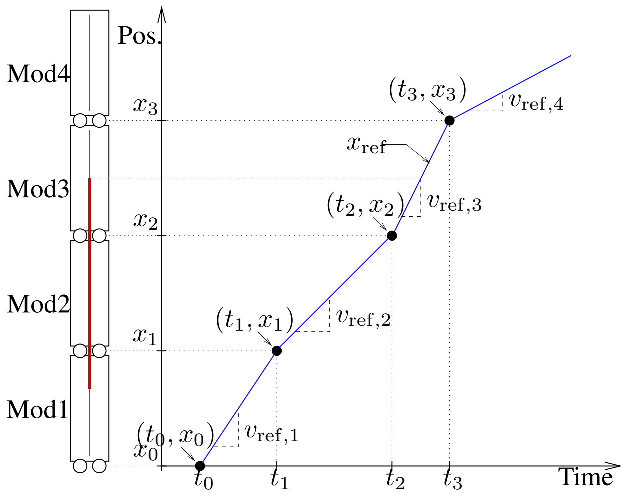

Fig. 2. Way points along with interpolated reference trajectory.

is *asynchronous*. The module controllers apply the control signals to the rollers ("nips"), which move the paper. This example system is discussed in more detail in Fromherz, *et al.* [3]. Further control issues in printing systems can be found in Hamby and Gross [4], Li, *et al.* [7], Chen and Chiu [2], and Krucinski, *et al.* [6].

The control signal $u$ is computed by generating a reference signal from the interpolated way points which is then applied to a two degree of freedom LQG controller (see Åström and Wittenmark [1]) to produce the appropriate control feedforward and feedback signals. The controller uses a time varying Kalman filter to handle the asynchronous arrival of the measurements. The network is modeled as a guaranteed service time network, where the maximum delay is known. Thus using the technique of sensor message time stamping and buffering [11], [9], [8], it is possible to make all the measurements *appear* as if they had the same constant (maximum) network delay. While conservative, this technique allows us to use the LQG framework by modeling the delay as an LTI system that is incorporated into the overall plant model. All the modules have access to a precise common global time, and we do not discuss issues of clock synchronization here; it has been studied extensively in the literature (see, for example, Tanenbaum and van Steen [14]).

A critical feature of the system in Figure 1 is that the paper can be in contact with more than one nip at any given time. Thus all the nips in contact with the same sheet must apply the *same control*, otherwise the sheet would be damaged by compression or stretching. This raises the practical question of how to implement the LQG controller described above in this distributed actuation and sensing environment. There are at least two alternatives.

One reasonable method might be to implement the LQG controller *centrally* in the sheet controller, and simply communicate the control signals to the appropriate nips via the module controllers. However, in our application, the control updates happen much more frequently than sensor messages, so a lot of network bandwidth could be saved by transmitting only the sensor messages.

Therefore we have instead chosen a more *distributed* implementation, at the cost of redundancy. In this distributed implementation, each module controller runs its own local replica of the LQG controller, *i.e.*, the *same controller* is used in all the nips in contact with the sheet. When properly initialized, since all these controllers receive the same sensor information, they will apply the same control signal to the sheet. As the sheet moves through the system, new nips will constantly be joining and leaving the actuation of the paper. Hence the need for the synchronization method presented here, which can be viewed as a mechanism for *controller hand-off*. As we shall show, this synchronization can be effected by embedding the LQG controller in a finite state machine (FSM), which takes care of the synchronization (see Figure 5). Now the sheet controller only needs to send very infrequent messages to the module controllers consisting of high level commands telling the nips when to switch on and off and from handling one sheet to another and relays of the relatively infrequent sensor information. Thus network bandwidth is used more efficiently.

## III. Synchronized Control Processes

Consider a set of control processes $\{p_0, \ldots, p_{n-1}\}$, where each process $p_i$ runs the following state-based iterations over time $t = 0, 1, 2, \ldots$

$$\begin{aligned} x_i(t+1) &= f\left(x_i(t), y_i(t-d), t\right); \quad x_i(0) = x_{i0} \\ u_i(t) &= g\left(x_i(t), y_i(t-d), t\right) \end{aligned}$$

where $x_i$ is the state, $u_i$ is the control output, $y_i$ is measurement input, $d$ is some nonnegative fixed integer delay (*e.g.*: maximum network delay), and $f$ and $g$ are some functions of state, measurement and time. Note that we make no assumptions about the spaces over which $x$, $u$ or $y$ are defined: they could be numbers, symbols, discrete or continuous. At every time step $t$, each process receives a new measurement input $y_i(t-d)$, and uses the recursions above to compute the next state $x_i(t+1)$ and the current control output $u_i(t)$. [For $t < d$, we assume that $f$ and $g$ are functions of only $x$ and $t$, and that they do not depend explicitly on $y$. Hence, we can take $y_i(t) = \emptyset$ (undefined) for $t < d$.]

The evolution of the states and the controls is completely determined by the initial states $x_{i0}$ and the measurement inputs $\{y_i(t-d) \mid t \geq 0\}$. Thus, it is clear that if the initial conditions are all equal and the processes are driven with the same measurements, then the states and control outputs are *identical* for all time. In other words, if

$$\begin{aligned} x_{i0} &= x_0; \quad \forall i \\ y_i(t) &= y(t); \quad \forall i, t, \end{aligned}$$

then the processes then all run the same recursion:

$$\begin{aligned} x(t+1) &= f\left(x(t), y(t-d), t\right); \quad x(0) = x_0 \\ u(t) &= g\left(x(t), y(t-d), t\right). \end{aligned} \tag{1}$$

Note that this is true for *any* functions $f$ and $g$ of $x$, $y$ and $t$. We refer to such a set of processes, where the $x(t)$ and $u(t)$ are identical for all time, as *synchronized*.

> **Problem Statement:** We seek a method for synchronizing a new process $p_n$, which starts at some time $t' \geq d$, to the existing processes $\{p_0, \ldots, p_{n-1}\}$, for all time $t \geq t'$. We assume $p_n$ knows $f$ and $g$, but not $x_0$. We would like this method to work for *any* choice of functions $f(x(t), y(t-d), t)$ and $g(x(t), y(t-d), t)$.

## IV. Synchronous Measurements

At the heart of our development is the following property of the state recursions in (1): *The current state captures all the past.* Specifically, for any time $t'$, future values of the state $\{x(t) \mid t \geq t'\}$ only depend on the current state $x(t')$ and *future* inputs $\{y(t-d) \mid t \geq t'\}$. All the effects of past inputs are "summarized" in the current state.

The property above suggests the following obvious method for synchronization. For any $t' \geq 0$, it follows immediately from (1) that a *sufficient* condition for $p_n$ to be synchronized with $\{p_0, \ldots, p_{n-1}\}$ for all time $t \geq t'$, and for any functions $f$ and $g$ of $x$, $y$ and $t$, is to set

$$\begin{array}{rcll} x_n(t') & = & x(t') & ;\text{ at time } t' \\ y_n(t-d) & \equiv & y(t-d) & ;\forall t \geq t' \end{array} \tag{2}$$

In fact, this condition is also *necessary*, since it is easy to construct simple examples of functions $f$ and $g$ for which synchronization for all $t \geq t'$ fails, if any part of condition (2) does not hold. We call this method *instantaneous initialization*, since $p_n$ receives $x(t')$ instantly, without any delay.

Now suppose that we can set $y_n(t-d) \equiv y(t-d)$ for all $t \geq t'$ but, because of the delay, we cannot receive the current state $x(t')$ immediately. Instead, $p_n$ only has access to the delayed history of the states and measurements:

$$\begin{aligned} \mathcal{I}_d(t') \;=\; & \{(x(0), y(0)), (x(1), y(1)), \ldots, \\ & (x(t'-d), y(t'-d))\}, \end{aligned}$$

which does not explicitly contain $x(t')$.

It turns out that synchronization based on $\mathcal{I}_d(t')$ is still possible, albeit with a little more effort. Observe that $x(t')$ can be computed from the information in $\mathcal{I}_d(t')$ by first performing $d$ iterations of the state recursion in (1):

$$\begin{array}{rcl} x(t'-d+1) & = & f\left(x(t'-d), y(t'-2d), t'-d\right) \\ x(t'-d+2) & = & f(x(t'-d+1), y(t'-2d+1), \\ & & \qquad\qquad t'-d+1) \\ & \vdots & \\ x(t') & = & f\left(x(t'-1), y(t'-d-1), t'-1\right). \end{array} \tag{3}$$

We refer to this operation as *forward propagating* the state from $x(t'-d)$ to $x(t')$, and we represent it using the following shorthand notation

$$x(t') = \Phi\left(x(t'-d) \mid y(t'-2d), \ldots, y(t'-d-1)\right)$$

[As before, we take $y(t) = \emptyset$ for $t < d$.]

Note that, provided that $t' \geq d$, all of the information required for the forward propagation operation, namely $x(t'-d)$ and $\{y(t'-2d), \ldots, y(t'-d-1)\}$, is available in $\mathcal{I}_d(t')$. Furthermore, this is the only information that we need from $\mathcal{I}_d(t')$. In other words, we only need a delayed history that is *$d$ time steps deep*. And from that, the only value of the states that we need is the most recent, namely $x(t'-d)$.

Thus for $t' \geq d$, a *sufficient* condition for $p_n$ to be synchronized with $\{p_0, \ldots, p_{n-1}\}$ for all time $t \geq t'$, and for any functions $f$ and $g$ of $x$, $y$ and $t$, is to set

$$\begin{array}{rcl} x_n(t') & = & \Phi\left(x(t'-d) \mid y(t'-2d), \ldots, y(t'-d-1)\right); \\ y_n(t-d) & \equiv & y(t-d); \quad \forall t \geq t' \end{array} \tag{4}$$

Once again, it can be shown that this condition is also *necessary*, as it is easy to construct simple examples of $f$ and $g$ where synchronization would fail if any part of the conditions (4) were not true[1].

We summarize our findings in the following:

> **Proposition:** Let $\{p_0, \ldots, p_{n-1}\}$ be a set of processes running (1). A new process $p_n$, which knows $f$ and $g$, can be synchronized with the given set at from time $t'$ onwards, and for any functions $f(x(t), y(t-d), t)$ and $g(x(t), y(t-d), t)$, if and only if, the following conditions hold: $p_n$ receives the same input measurements $\{y(t-d) \mid t \geq t'\}$ and either:
>
> 1) $t' \geq 0$ and, at time $t'$, $p_n$ has access to $x(t')$, for synchronization via instantaneous initialization (2);
> 2) $t' \geq d$ and, at time $t'$, $p_n$ has access to $x(t'-d)$ and $\{y(t'-2d), \ldots, y(t'-d-1)\}$, for synchronization via forward propagation (4).

## V. Asynchronous Measurements

We now consider the problem of synchronization with asynchronous measurements. By asynchronous measurements, we mean that at certain times, some of the elements of the measurement sequence $\{y(t-d) \mid t \geq 0\}$ could be missing, but the ones that arrive do so in the right order. Also, some of the states could be missing from the history $\mathcal{I}_d$. We will use the symbol $\emptyset$ to denote missing measurements or states; it should be interpreted as meaning "no information".

This asynchronous measurements scenario can still be modeled by (1) as follows: at each time $t$, define $y(t-d)$ as:

$$y(t-d) = \begin{cases} y_m(t-d); & \text{if measurement arrives} \\ \emptyset; & \text{otherwise} \end{cases}$$

[1]Specific examples of $f$'s which could not be synchronized if the conditions do not hold: (a) if $x(t-d)$ were missing, consider the "identity system" $x(t+1) = f(x(t), y(t-d), t) \doteq x(t)$; (b) if $y(t-d-i), i = 1, \ldots, d$ were missing, consider the "shift register" system with state defined as $x(t) = (y(t-d-1), \ldots, y(t-2d))$, and take $f(x(t), y(t-d), t)$ as $x(t+1) \doteq Ax(t) + By(t-d)$, where $A$ is the shift matrix with ones on the lower subdiagonal and zeros elsewhere, and $B$ is the unit vector with one in the top entry and zeros elsewhere.

where $\{y_m(t-d) \mid t \geq 0\}$ is some uncorrupted sequence of measurements. Thus the asynchronous measurements scenario is essentially nothing but a particular instance of (1), for some specific measurement sequence $\{y(t-d) \mid t \geq 0\}$ defined above. The fact that for some values of $t$, $y(t-d)$ might take on the value of $\emptyset$ is immaterial since, as mentioned in the first section, we have made no particular assumptions about the spaces over which the measurements are defined. Also, $f$ and $g$ should be well defined for all possible values of $y$, including $\emptyset$. Practically, this means that $f$ and $g$ will have the form:

$$f(x(t), y(t-d), t) = \begin{cases} f_m(x(t), y_m(t-d), t); & \text{if meas. arrives} \\ f_\emptyset(x(t), t); & \text{otherwise} \end{cases}$$

$$g(x(t), y(t-d), t) = \begin{cases} g_m(x(t), y_m(t-d), t); & \text{if meas. arrives} \\ g_\emptyset(x(t), t); & \text{otherwise} \end{cases} \quad (5)$$

In other words, when $y$ supplies no information, then $f$ and $g$ do not depend explicitly on $y$.

Since we have shown that the asynchronous delayed measurements scenario can be modeled by (1), the conditions for synchronization are given in our Proposition. We will now apply the conditions of the Proposition to this specific asynchronous measurements context.

It follows immediately from the Proposition that synchronization using instantaneous initialization always works in this asynchronous case.

Now consider forward propagation. In this case, the Proposition states that synchronization at a time $t' \geq d$ using forward propagation (4) is only possible if and only if: $p_n$ receives the same measurements for all $t' \geq d$ and, at time $t'$, $p_n$ has access to $x(t'-d)$ and $\{y(t'-2d), \dots, y(t'-d-1)\}$. Note that, due to missing measurements or states, in general, at a given time $t'$, the delayed state and measurement history $\mathcal{I}_d(t')$ will have the form:

$$\mathcal{I}_d(t') = \begin{cases} \{\dots, (x(t'-d), y(t'-d))\}; & \text{if state arrives} \\ \{\dots, (\emptyset, y(t'-d))\}; & \text{otherwise} \end{cases} \quad (6)$$

where "state" in (6) refers to the delayed state $x(t'-d)$. From (6), we see that for all $t' \geq d$, $\mathcal{I}_d(t')$ will always contain the information $\{y(t'-2d), \dots, y(t'-d-1)\}$. Entries of $\emptyset$ for $y$ pose no problem, since they represent what was actually used in $f$ and $g$ in (5) for $\{p_0, \dots, p_{n-1}\}$. However, (6) also shows that it could happen that, at certain times $t' \geq d$, $\mathcal{I}_d(t')$ does not contain $x(t'-d)$. At such times, synchronization using the forward propagation technique in (4) is not possible, and one would have to wait until a time $t'' > t'$, when $x(t''-d)$ is available, to use forward propagation.

Thus we conclude that in the asynchronous case, synchronization using forward propagation is only possible at times $t'$ when delayed state $x(t'-d)$ is available. Otherwise it is necessary to wait until a time at which the delayed state is available.

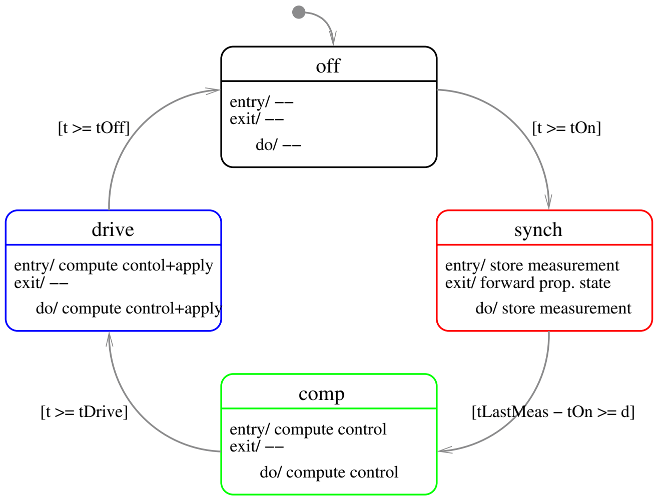

Fig. 3. Simplified statechart of synchronization state machine (FSM).

## VI. Real-Time Finite State Machine Implementation

This section gives an example of how the synchronization mechanism can be used in practice. The goal in this example is to synchronize $p_n$ to $\{p_0, \dots, p_{n-1}\}$ from time tDrive until a time tOff. This will be accomplished by *embedding the process in a finite state machine*. Figure 3 shows the finite state machine (FSM)[2], and Figure 4 is an associated timing diagram detailing an example trace of the execution of the state machine.

The FSM is described by a statechart (see Harel [5] and Samek [13]). The statechart is event driven by the clock tick events, which occur at integer multiples of $T_s$, the control sample period. At each clock tick, the FSM performs actions based on which state it is in. Usually, this is the action specified in the "do" statement. However, upon the assertion of certain guard conditions, shown in square brackets, the state machine may transition to another state. If this is the case, then the overall transition operation will consist of three steps: performing the exit actions of the current state, changing the name of the state, and performing the entry actions of the new state.

It is assumed that all the processing takes place very quickly and this is shown in the timing diagram (Figure 4) by the black slabs on the time axis. Processing includes all the discrete state machine operations such as accepting inputs, exporting outputs, checking guard conditions, entry and exit actions, changing state, etc., as well as the continuous operations such as control computation and forward propagation, using (1) and (3), etc. To keep things simple in this example timing diagram, we assume the delayed $y$ and $x$ measurements always arrive (or fail to arrive) simultaneously,

[2]Note that, to avoid clutter, a few unusual transitions are omitted from the statechart: for certain values of tOn/tDrive/tOff and d, there could be transitions from the Off state directly to Comp or Drive, and from Synch to Drive, etc.

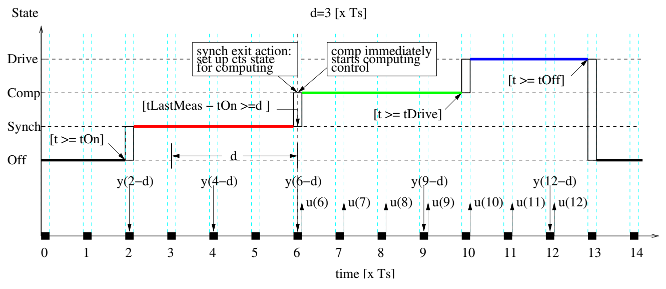

Fig. 4. Timing diagram showing the FSM execution.

thus the $y$ symbol shown denotes $(x, y_m)$ pairs, and $(\emptyset, \emptyset)$ pairs are omitted for clarity.

The FSM has four states: Off, Synch, Compute, and Drive. In the Off state, the FSM waits until a time tOn, at which point it transitions to the Synch state. The time tOn is chosen to be sufficiently in advance of tDrive, such that there is enough time and enough measurements arrive before tDrive that synchronization can be completed. The Synch state collects measurements, until a time when it has a delayed measurement and state history that is $d$ time steps deep *and* a state measurement arrives. At that point it exits the Synch state, initializing the control process by forward propagation. Then it transitions to the Compute state, and executes the entry action, namely performing the first iteration of (5). It then continues to perform the control computation (5), as shown in the do-statement, until a time tDrive, at which point it transitions to the Drive state. The Drive state is very similar to the Compute state, except that the control is actually applied to the target system. Then, at a time tOff, the FSM turns itself off. This whole process is illustrated in the timing diagram.

Hence by embedding the process $p_n$ in an FSM, the desired synchronization can be accomplished in practice (see Figure 5).

## VII. Simulation Results

We will now describe the role of the synchronization technique described here in the distributed control architecture currently being used in the modular paper path prototype (see Figure 5). The network delay $d$ is three control time samples.

In this application, we use an LQG controller (see Åström and Wittenmark [1]) with a model of the time delay, implemented with a time varying Kalman filter in the measurement-update time-update form, to handle the asynchronous measurements, which gives rise to $f$ and $g$ functions of the form (5). Following the logic from Section V, then, the controller update will have a different form depending on whether or not measurement data arrives at a given time step. If a measurement does arrive, the Kalman filter update will consist of both the measurement update and the time update, but if a measurement does not arrive, only the time update will be performed.

The LQG controller uses these measurements to compute controls for the nips to ensure that the paper accurately follows a desired trajectory. Thus each process $p_i$ runs an LQG controller and is responsible for controlling (the motor of) one nip. Each process is in turn embedded in an FSM which enables it to synchronize with the other processes. As in Section V, synchronization must await the arrival of a delayed state $x(t' - d)$. In this simulation, $y$ and $x$ always arrive (or not) together, as in Figure 4.

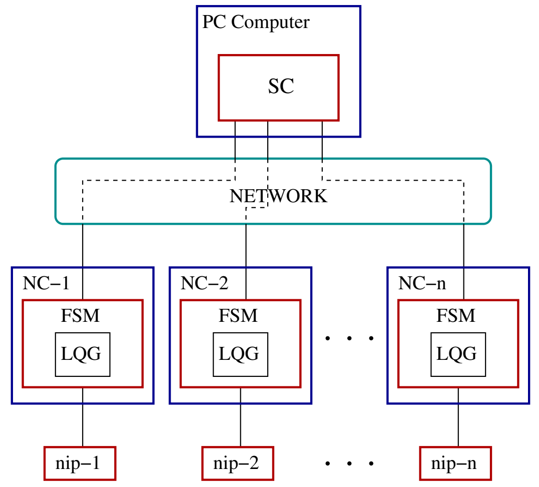

Fig. 5. Networked FSM-embedded LQG controllers (sensing paths not shown). (SC="Sheet Controller"; NC="Nip Controller", the software component of the Module Controllers that handles the nips.)

Simulation results are shown in Figures 6, 7, and 8. Figure 6 shows the trajectory being simulated. The dark top line shows the nominal trajectory for the leading edge of the sheet, and the bottom for the trailing edge. The module boundaries are shown with dotted lines. The amount of time the nip for each module is turned on (is in the Drive state) can be seen in the thick dark horizontal lines. The other FSM states are shown as well. Figure 7 shows the tracking error (a few millimeters at its maximum), the state estimation error, and the control signal (nip speed). Only one control signal plot is shown because, since the nips are all synchronized, they all use identical control signals. The estimation error plot also shows the calculated estimation error variance, which demonstrates how, in the asynchronous measurement case, the uncertainty grows during times when there are no measurements (sensor crossings are shown as asterisks in the top plot). Figure 8 depicts the hand-off among the different controllers. With the parameters in this simulation, three nips could be active (in Drive) simultaneously, as can be seen through the solid thick bars. The nips all produce the same control signal, since they are synchronized.

The synchronization technique presented here has been implemented and is currently running in a real physical system. It enables synchronized control among distributed, networked controllers in a system in which such tight coordination is essential. Testing and evaluation in the physical prototype is in progress.

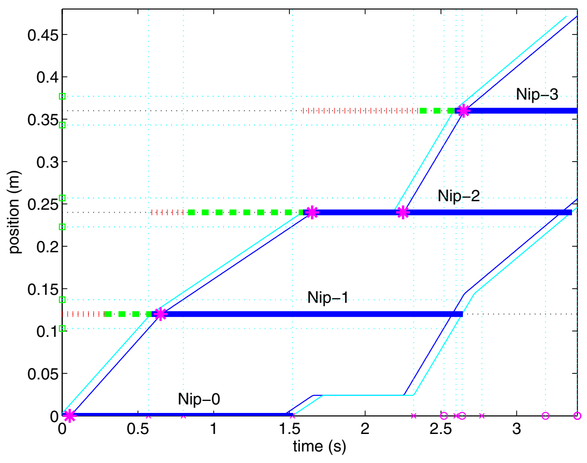

Fig. 6. An example of a nominal sheet trajectory obtained by direct interpolation of way points (asterisks). The leading edge (upper) and trailing edge (lower) are both shown. The sheet controller uses this trajectory to compute tOn, tDrive, and tOff for the different nips. The dark dotted lines are the module boundaries. The sheet is in a given module from the time its leading edge crosses into the module boundary, until its trailing edge leaves. The squares on the y-axis are the positions of the edge sensors. The x-es and circles on the x-axis show the nominal leading edge and trailing edge, respectively, sensor crossing times. Also shown are the nominal states of the FSM: dotted=Synch, dashed=Comp, solid=Drive. For practical reasons, the Drive state starts a bit before and ends a bit after the nominal times the sheet will be inside the module.

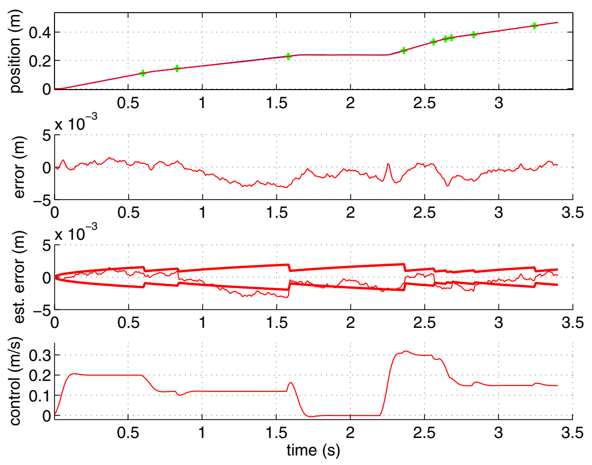

Fig. 7. Plots of: overall trajectory for a single sheet (asterisks are sensor crossings), tracking error, position state estimation error and variance, and overall nip speed control signal. Various portions of this nip speed control signal must be applied by the different nips using the synchronization mechanism.

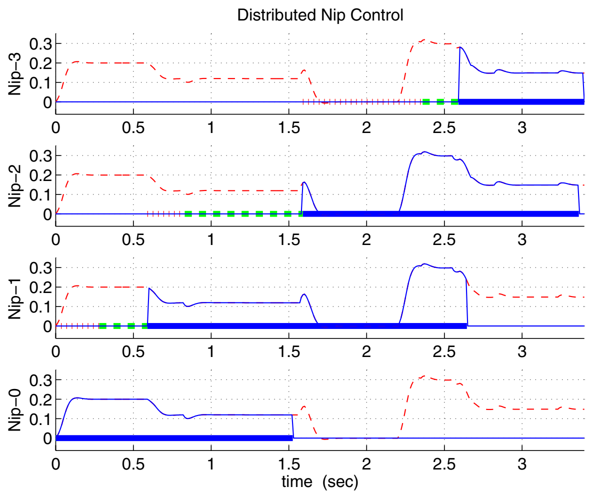

Fig. 8. Overall nip speed control signal (dashed) along with the drive signals (solid) superimposed for the first four nips. The drive times are from Figure 6. Along the x-axis, the line styles show the states of the FSMs (dotted=Synch, dashed=Comp, solid=Drive). The nips are actuated by stepper motors with extremely fast dynamics compared to the sheet system.

## VIII. Conclusion

This paper has addressed the problem of controller state synchronization in a networked control system with distributed sensing and actuation, where actuators need to hand off and switch controllers "on the fly," as they go from performing one task to another, in the presence of fixed communication delays and asynchronous measurements. We presented a technique which enables new controllers to seamlessly join and leave a task, by encapsulating the controller in a finite state machine that handles the synchronization and hand-off. We also discussed issues related to the real-time implementation of this technique and demonstrated it on an example from the document printing domain.

## References

[1] K.J. Åström and B. Wittenmark. *Computer Controlled Systems*. Prentice Hall, 1997.
[2] C.-L. Chen and G. Chiu. Incorporating human visual model and spatial sampling in banding artifact reduction. In *Proceedings of the American Control Conference*, Boston, MA, June 2004.
[3] M. P.J. Fromherz, L. S. Crawford, and H. A. Hindi. Coordinated control for highly reconfigurable systems. In *Hybrid Systems: Computation and Control (HSCC)*, Zurich, Switzerland, 2005. Springer-Verlag.
[4] E. Hamby and E. Gross. A control-oriented survey of xerographic systems: basic concepts to new frontiers. In *Proceedings of the American Control Conference*, Boston, MA, June 2004.
[5] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8, 1987.
[6] M. Krucinski, C. Cloet, R. Horowitz, and M. Tomizuka. A mechatronics approach to copier paperpath control. In *Proceedings of the first IFAC conference on mechatronics systems*, Darmstadt, Germany, September 2000.
[7] P. Li, T. Sim, and D. Lee. Time sequential sampling and reconstruction of tone and color reproduction functions for xerographic printing. In *Proceedings of the American Control Conference*, Boston, MA, June 2004.
[8] B. Lincoln and B. Bernhardsson. Optimal control over networks with long random delays. In *Proceedings of the International Symposium on Mathematical Theory of Networks and Systems*, 2000.
[9] R. Luck and A. Ray. An observer-based compensator for distributed delays. *Automatica*, 26(5):903 – 908, May 1990.
[10] M. Mauve. Consistency in continuous distributed interactive media. *ACM CSCW*, pages 181–190, 2000.
[11] J. Nilsson. Real-time control systems with delays, 1998.
[12] Y. Owada and S. Asahara. Distributed processing system, distributed processing method and client terminal capable of using the method. Technical Report US 2002/0194269 A1, US Patent Application Publication, December 2002.
[13] M. Samek. *Practical statecharts in C/C++*. CMP Books, 2002.
[14] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, New Jersey, 2002.