Building Blocks for Co-Design of Controllers and Implementation Platforms in Embedded Systems

by

Leslie Grace Maldonado

Submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author ....
Department of Mechanical Engineering
May 10, 2013

Certified by ....[illegible]..........
Anuradha Annaswamy
Senior Research Scientist
Thesis Supervisor

Accepted by ....[illegible]..........
David E. Hardt
Chairman, Department Committee on Graduate Theses

Building Blocks for Co-Design of Controllers and Implementation Platforms in Embedded Systems

by

Leslie Grace Maldonado

Submitted to the Department of Mechanical Engineering
on May 10, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

One of the biggest challenges in implementing feedback control applications on distributed embedded platforms is the realization of required control performance while utilizing minimal computational and communication resources. Determining such tradeoffs between control performance (e.g., stability, peak overshoot, etc.) and resource requirements is an active topic of research in the domain of cyber-physical systems (CPS). In this thesis, a setup is considered where multiple distributed controllers communicate using a hybrid (i.e., time- and event-triggered) communication protocol like FlexRay (which is commonly used in automotive architectures). Mapping all control messages to time-triggered slots results in deterministic timing and hence good control performance, but time-triggered slots are more expensive. The event-triggered slots, while being less expensive, result in variable message delays and hence poor control performance. In order to tradeoff between cost and control performance, a number of recent papers proposed a switching scheme where messages are switched between time- and event-triggered slots based on the state of the plant being controlled. However, all of these studies were based on a monotonic approximation of the system dynamics. This while simplifying the resource dimensioning problem (i.e., the minimum number of time-triggered slots required to realize a given control performance) leads to pessimistic results in terms of usage of time-triggered communication. In this thesis, it is shown that the usage of time-triggered communication (i.e., the requirement on the minimum number of time-triggered slots for a given control performance) is reduced when an accurate, non-monotonic behavior of the system dynamics is considered in the analysis. This technique is illustrated using a number examples and a real-life case study. While the focus is on communication resources in this thesis, these results are general enough to be applied to a wide range of problems from the CPS domain.

Thesis Supervisor: Anuradha Annaswamy
Title: Senior Research Scientist

Acknowledgments

I would like to thank Dr. Anuradha Annaswamy for her support in my research and all the mentoring she has provided me. I would like to thank Ken Butts and Prashant Ramachandra from Toyota for their support and interest in my research and for their direction during my summer work at Toyota. I would also like to my labmates Sarah Thornton and Dan Wiese for all the late night homework and study times. Thank you also to Andy Wright for his awesome support and encouragement through all these semesters at MIT. Lastly, thank you to my mom, dad and brother for their understanding, support and unconditional love.

Support: This work was supported in part by the NSF Grant No. ECCS-1135815 via the CPS initiative.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This thesis focuses on the distributed implementation of multiple feedback controllers onto an automotive architecture with a network of electronic control units (ECUs). In particular, the scenario where the ECUs transmit signals over a shared hybrid communication bus such as FlexRay [1] and TTCAN [11] is of interest. Due to wide variety of functional and timing requirements in domains like automotive the hybrid protocols are considered to be an attractive option for today's in-vehicle communication network. In general, the hybrid bus protocols allow both time-triggered (e.g., static segment of FlexRay) and event-triggered (e.g., dynamic segment of FlexRay) communication. Both segments have their own advantages and disadvantages. By pure time-triggered (TT) communication with perfectly synchronized TT *slots* and ECUs, it is possible to achieve negligible communication delay. But pure TT communication is highly bandwidth consuming and hence expensive in terms of resource usage. On the other hand, a bandwidth efficient pure event-triggered (ET) implementation often results in a variable and large communication delay due to the arbitration with higher-priority traffic. Thus, a feedback controller provides (i) a good control performance with high resource usage (i.e., network bandwidth) in a pure TT implementation (ii) a poor control performance with low resource usage in a pure ET implementation. The goal is to obtain an improved control performance with a tight *resource dimensioning.* That is, what is aimed for is to achieve a control performance better than what one could have achieved using a pure ET implementation and at the same time, to reduce the usage of TT slots compared to a pure TT implementation. Towards achieving these two conflicting goals, another possible implementation is to map the feedback signals dynamically to the time-triggered and the event-triggered segments depending on the *state* of the plant being controlled [16]. That is, multiple controllers *share* a TT slot and each controller has its own ET priority: only one controller can use the TT slot at any given point in time and all other controllers use their ET priority for message communication. Essentially, a TT slot is *arbitrated* by the multiple controllers based on the plant's state which further depends on the underlying dynamics being controlled. The presented technique aims to achieve a tighter resource dimensioning using such mixed time-/event-triggered communication schemes. Although the applicability of the presented technique is illustrated in the context of resource dimensioning in hybrid protocols, it can be adapted to other settings such as control loops with variable sampling rates or task periods.

1.1 Contributions

In the above context, an external *disturbance* causes perturbation in the system dynamics which brings the system in *transient* state (see – Fig. 2-4). To reject such disturbance for bringing it back to *steady state* within a given time duration ξ_i^d, a control application C_i needs to use the shared TT slot for the transmission of its feedback signal and towards this, it needs to arbitrate with other control applications for accessing the TT slot. In this setting, as a slot-sharing policy, this work considers *fix-priority non-preemptive* scheduling where each control application has a predefined priority. Once an application gets chance to send its feedback signal via the TT slot, it cannot be preempted for $t_{dw,i}$ time units which is computed to be enough to completely reject all possible disturbances under consideration. While arbitrating for the access to the TT slot, a control application C_i might have to wait $t_{wait,i}$ time units since the TT slot might already being used by the higher-priority applications. Thus, the above arbitration for the TT slot gives rise to a classical schedulability problem which is what is attempted to be addressed in this work. Here, the relationship between $t_{dw,i}$ and $t_{wait,i}$ depends on the underlying system dynamics and plays a key role in achieving tighter resource dimensioning (see Figure 3-1 - parameters are explained later). To this end, all previous attempts made simplifying assumption [16, 17, 9] on monotonic system dynamics which resulted in a linear relationship between $t_{dw,i}$ and $t_{wait,i}$ – see Figure 3-1. The results presented here show that such monotonic assumption often results in conservative resource allocation. In this work, a schedulability analysis is presented which takes this accurate non-monotonic system dynamics into account. It is shown that there exists a stable *fixed point* solution of the above schedulability problem utilizing classical Lyapunov based approach from control theory. Further, the existence of the above non-monotonic property is verified using an automotive experimental setup. Finally, the effectiveness of this technique is shown in terms of savings in communication resource (i.e., the number of TT slots). Although the presented technique is applicable to many other domains such as avionics, the applicability of this analysis is more prominent in the cost-sensitive domains like automotive.

1.2 Related work

While control over wireless networks has been a focus of networked control system (NCS) literature [10, 6, 12], this work mainly targets fault-tolerant wired communication for the feedback signals. This work can be classified as "control/network" co-design [5] which is related to two broad areas: (i) schedulability/timing analysis (ii) control/schedule co-design. The timing/schedulability analysis for real-time systems mainly determines response time of a real-time task [23, 4]. In distributed settings, timing analysis methods exists for time-triggered [19, 13, 24], hybrid [21], and event-triggered [20] systems. Typical questions addressed in these works include computation of the worst-case end-to-end delays, optimal schedules synthesis, and partitioning of system functionality into time-triggered and event-triggered activities.

The schedulability of control tasks is studied in several recent works [15, 25, 7, 14, 17, 16]. In general, the question addressed in these works is to how to do schedulability analysis or schedule synthesis such that one or multiple control applications provide optimal performance. Analysis methods exist for schdulability/timing analysis for both single-processor [25] and distributed architectures [15, 7, 14] in this context. Further, in the case of distributed settings, the problem of network schedule synthesis/analysis taking control performance requirements into account is studied in [7, 17, 16]. The work presented in [17] formulated a schedulability analysis problem using a *limited-preemption* scheme with retransmissions which reduces the number of time-triggered slots that are necessary. In [16], a similar scheduling analysis was done with a *monotonic* assumption between dwell time $t_{dw,i}$, the time taken by the TT slot to result in a desired response, and wait time $t_{wait,i}$, the time that the application may lie in an ET implementation due to priority-based arbitration.

1.3 Organization

The rest of this thesis is organized as follows. The problem is formally presented in Chapter 2. Chapter 3 then provides a formal characterization of the non-monotonic system behavior the control applications may have. This is followed by a discussion of the schedulability analysis in Chapter 4. The applicability of this analysis is illustrated with a case study in Chapter 5.

Chapter 2

Problem Setting and Motivation

In this chapter, the distributed implementation of multiple control applications on a network of ECUs is described in detail and the problem to be addressed in this thesis is formally presented. The goal from the controls context and from the systems context is outlined and a co-design is proposed where the design of the distributed implementation uses information from the control application and the controller design uses information from the description of the distributed architecture.

2.1 Control Problem

For a control application C_i, we consider a standard continuous-time model, given by

$$\dot{x}_i(t) = A_i x_i(t) + B_i u_i(t - \tau_i) + D_i(t) \tag{2.1}$$

where x_i are the plant states, u_i is the control input, D_i is an impulse disturbance that occurs sporadically, and τ_i is the maximum communication delay between reading sensor data and the corresponding actuation (*sensor-to-actuator* delay) and can be arbitrary. In order to implement the requisite controller, the signals in (2.1) are sampled at a constant sampling period h_i. Since τ_i can be arbitrary, the following is defined:

$$\begin{aligned}\tau_i' &= \tau_i - \left\lfloor \frac{\tau_i}{h_i} \right\rfloor h_i \\ \tau_{1,i} &= \left\lfloor \frac{\tau_i}{h_i} \right\rfloor \\ \tau_{2,i} &= \left\lceil \frac{\tau_i}{h_i} \right\rceil\end{aligned} \tag{2.2}$$

Defining $x_i[k] := x_i(kh_i)$ and $u_i[k] := u_i(kh_i)$, the zero-order hold sampling of the continuous-time model in (2.1) gives the discrete plant-model [3]

$$x_i[k+1] = \Phi_i x_i[k] + \Gamma_{0,i} u_i[k - \tau_{1,i}] + \Gamma_{1,i} u_i[k - \tau_{2,i}] + \hat{D}_i[k] \tag{2.3}$$

where Φ_i, $\Gamma_{0,i}$ and $\Gamma_{1,i}$ are discrete-time equivalent system matrices and are given by

$$\begin{aligned}\Phi_i &= e^{A_i h_i} \\ \Gamma_{0,i} &= \int_0^{h_i - \tau_i'} e^{A_i s} ds\, B_i \\ \Gamma_{1,i} &= \int_{h_i - \tau_i'}^{h_i} e^{A_i s} ds\, B_i\end{aligned} \tag{2.4}$$

and $\hat{D}_i[k]$ is the discrete equivalent disturbance given by

$$\hat{D}_i[k] = \sum_{j=1}^{N_D} M_{D,j} e^{A_i h_i \left(k_{D,j} + 1 - \frac{t_{D,j}}{h_i}\right)} \delta[k - k_{D,j}]. \tag{2.5}$$

In the above equation, δ is a unit impulse function, N_D is the number of impulse disturbances that occur, M_D is the magnitude of the impulse disturbance, t_D is time it occurs and k_D is the corresponding sample in which it occurs and is given by $k_D = \lfloor \frac{t_D}{h_i} \rfloor$.

With the plant model as in (2.3), the goal of the control application is to choose $u_i[k]$ so that regulation is achieved in a stable manner, i.e., choose $u_i[k]$ so that $x_i[k]$ tends to zero. In addition, each application C_i must achieve the regulation within a specified desired response time or *deadline* ξ_i^d.

2.2 Distributed Implementation

Each control application C_i uses two ECUs and is divided into three tasks as shown in Figure 2-1: a sensor task $T_{s,i}$ measures the plant states x_i, a control task $T_{c,i}$ computes the control input u_i, and an actuator task $T_{a,i}$ applies the control input u_i to the actuator. The tasks $T_{s,i}$ and $T_{c,i}$ are mapped to one ECU which is attached to the corresponding sensors and the task $T_{a,i}$ is mapped to a second ECU which is attached to the corresponding actuators. Additionally, the control input u_i is sent from task $T_{c,i}$ on one ECU to task $T_{a,i}$ on the other ECU over a hybrid communication bus. Since the tasks $T_{s,i}$ and $T_{c,i}$ are mapped to the same ECU, they do not need to communicate over the bus.

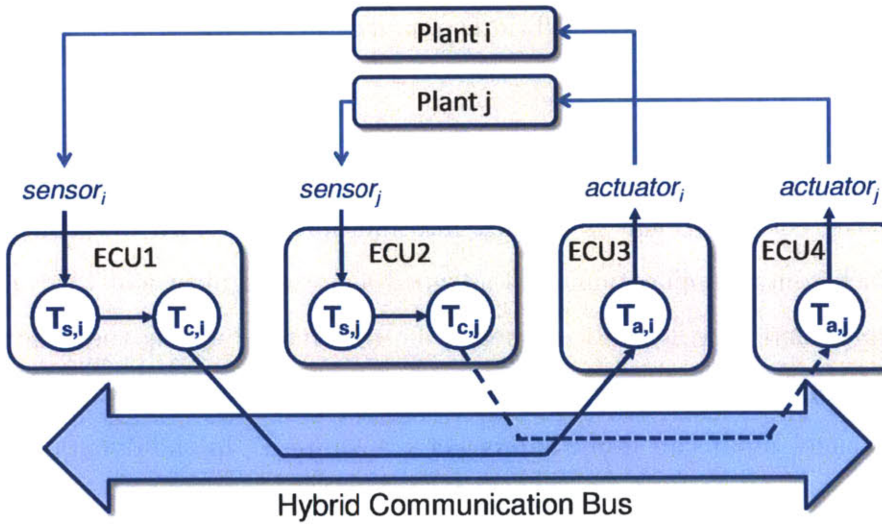


Figure 2-1: Distributed cyber-physical architecture

2.2.1 Task Triggering

The tasks $T_{s,i}$ and $T_{a,i}$ that belong to a particular control application are triggered periodically with the sampling period h_i. Task $T_{c,i}$ is triggered after the execution of task $T_{s,i}$ is finished. Therefore, each periodic triggering generates one control input u_i which is then transmitted over the communication bus and received by task $T_{a,i}$.

2.2.2 Hybrid Communication Bus Protocol

The communication bus follows FlexRay specification [1] where the communication bandwidth is divided into communication cycles of equal and predefined length. Each communication cycle on the bus is further divided into a time-triggered (or static) and an event-triggered (or dynamic) segment as shown in Figure 2-2.

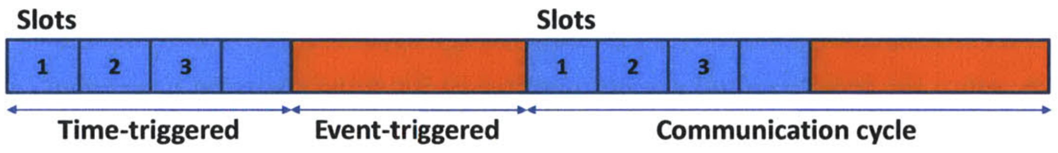


Figure 2-2: Hybrid communication protocol

The static segment follows time division multiple access (TDMA) scheduling policy where the entire segment is divided into multiple *slots* with predefined size or length. Each control application C_i is assigned a static segment slot to transmit its control inputs and they are only allowed to be transmitted during the assigned slot. If the generation of the control input u_i is synchronized with the starting time of its slot, the control input can immediately get transmitted. In the dynamic segment, scheduling is priority-based. Every control application C_i is assigned a priority. The control application with the highest priority gets to transmit its control input first while the lower priority control applications wait to transmit their control inputs.

Every controller task $T_{c,i}$ can send control input u_i to task $T_{a,i}$ over either the static or the dynamic segment of the bus. The transmission rate in FlexRay is 10 Mbit/s. As a result, the transmission times of control inputs over the bus are generally in the order of μs and therefore negligible compared to the sampling periods h_i of common control applications which are in the order of ms. The execution times of tasks $T_{s,i}$, $T_{c,i}$ and $T_{a,i}$ are on the order of a few μs and therefore also negligible compared to the sampling period h_i.

Static Segment Attributes

The triggering of tasks $T_{s,i}$ and $T_{a,i}$ are synchronized with a given slot on the static segment of the bus. It is assumed (which is common in real applications) that the slot length on the static segment has been chosen such that every possible control input fits entirely into one slot. The transmission of control inputs u_i, therefore, experience *zero* (or negligible) delay when being transmitted over the static or time-triggered (TT) segment.

Dynamic Segment Attributes

On the dynamic or event-triggered (ET) segment, the sending of control inputs may experience a maximum communication delay τ_i. This is due to a possible contention among control applications with different priorities trying to send their control inputs. The maximum delay the sending of control inputs experience can be computed by the traditional worst-case response time analysis. We assume that the priority assigned to every control application C_i for the dynamic segment is such that $0 < \tau_i \leq h_i$ holds for that C_i. This implies that

$$\begin{aligned} \tau_{1,i} &= 0 \\ \tau_{2,i} &= 1 \end{aligned} \tag{2.6}$$

in equation (2.3).

2.3 Performance Requirements

We consider n control applications C_i ($i \in \{1, 2 \ldots n\}$) with sampling period h_i that run on a distributed automotive architecture as in Figure 2-1. As already mentioned, the objective of each control application is to achieve the system states $x_i[k] \to 0$. In this context, a *disturbance* D_i in (2.1) moves $x_i[k]$ away from zero. Based on the error tolerance of the system, a value E_{th} is chosen by the designer such that the norm of the system states $\sqrt{x_i^T[k]x_i[k]} \leq E_{\text{th}}$ is tolerable and referred to as *steady-state.*

Denoting the norm of the vector $x_i[k]$ as $||x_i[k]||$ where

$$||x_i[k]|| := \sqrt{x_i^T[k]x_i[k]},$$

when a disturbance arrives, the state norm moves away from zero resulting in $||x_i[k]|| > E_{\text{th}}$. In that case, the performance requirement of each control application is to bring the state norm down to E_{th}, i.e., $||x_i[k]|| \leq E_{\text{th}}$ or steady-state, within a desired response time of ξ_i^d from when the disturbance occurs.

2.4 Resource Constraints

We assume that there are m TT slots S_j $(j \in \{1, 2 \dots m\})$ and $m < n$. That is, the number of available TT slots is less than the number of control applications. Therefore, each control application C_i cannot be assigned a dedicated TT slot for transmitting their computed control inputs u_i.

2.5 Design Challenges

As mentioned before, the control input u_i can either be transmitted via TT slots or the priority-based ET segment. Given this, there are two design possibilities to consider: (D1) Fully synchronized time-triggered implementation and communication via TT slots. With zero communication delay, such an implementation leads to fast response times ξ_i^{TT} that meet the performance requirements since $\xi_i^{TT} < \xi_i^d$ as shown in Figure 2-3. In this design, a dedicated TT slot is necessary for each control application. Therefore, n TT slots would be needed for such an implementation. (D2) Fully event-triggered implementation and communication via ET segment. In this case, the delay τ_i between sensor to actuator causes a significant deterioration in response times, where ξ_i^{ET} are the response times, since $\xi_i^{ET} > \xi_i^d$. Clearly, the design option (D1) is not implementable with the given resource constraints of having less than n TT slots and design option (D2) does not satisfy the performance requirements of

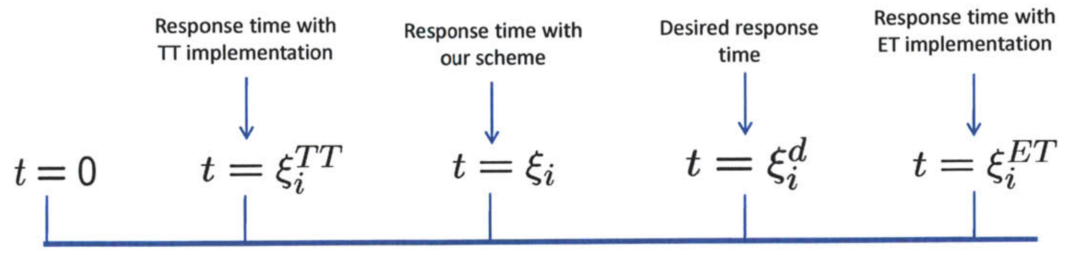

Response time with TT implementation
Response time with our scheme
Desired response time
Response time with ET implementation
$t = 0$
$t = \xi_i^{TT}$
$t = \xi_i$
$t = \xi_i^d$
$t = \xi_i^{ET}$

Figure 2-3: Relation among various response times

having response times within ξ_i^d. Another design option is therefore proposed in the following section to meet all the given constraints.

2.6 Basic Switching Scheme

The overall goal is to meet the performance requirements (by making sure $\xi_i < \xi_i^d$ for $i \in \{1, 2 \dots n\}$) in presence of resource constraints (using m TT slots with $m < n$). To achieve this, a basic switching scheme, shown in Figure 2-4, is proposed where TT slots are used to transmit control input whenever $\|x_i[k]\| > E_{\text{th}}$ (i.e., a disturbance arrives), but not otherwise (i.e., during steady-state). This switching scheme allows one to economize the number of TT slots (which is necessary as $m < n$). Therefore, more than one control application C_i are assigned to a particular TT slot S_j for $j \in \{1, 2 \dots m\}$. The control applications, that are *sharing* the same TT slot, send their control inputs u_i via the TT slot only in the case of a disturbance. When multiple such control applications (sharing the same TT slot) experience disturbances simultaneously, only the one with the highest priority can use the TT slot for message communication while all the rest with lower priorities must use ET communication and *wait* until the shared TT slot becomes available. In this setting, a lower priority control application has to wait t_{wait} time using ET communication to reject the current disturbance while their TT slot is not available. Once the TT slot is available to such an application, it takes another $t_{dw,i}$ or *dwell* time, with TT communication to fully reject the disturbance. The response time of such lower priority control application is then

$$\xi_i = t_{dw,i} + t_{wait,i}. \tag{2.7}$$

We need to achieve

$$t_{dw,i} + t_{wait,i} \leq \xi_i^d$$

for each control application C_i to meet our performance requirements. Evidently, the relation between $t_{wait,i}$ and $t_{dw,i}$ highly depends on the system dynamics and plays an important role in this scheme. An accurate and non-monotonic behavior of the system dynamics is exploited to determine which control applications are scheduled on each TT slot so that they meet their performance requirements and is shown in Chapter 4. This is unlike the previous efforts in this direction [16, 17] where a simplified/approximated system model was considered.

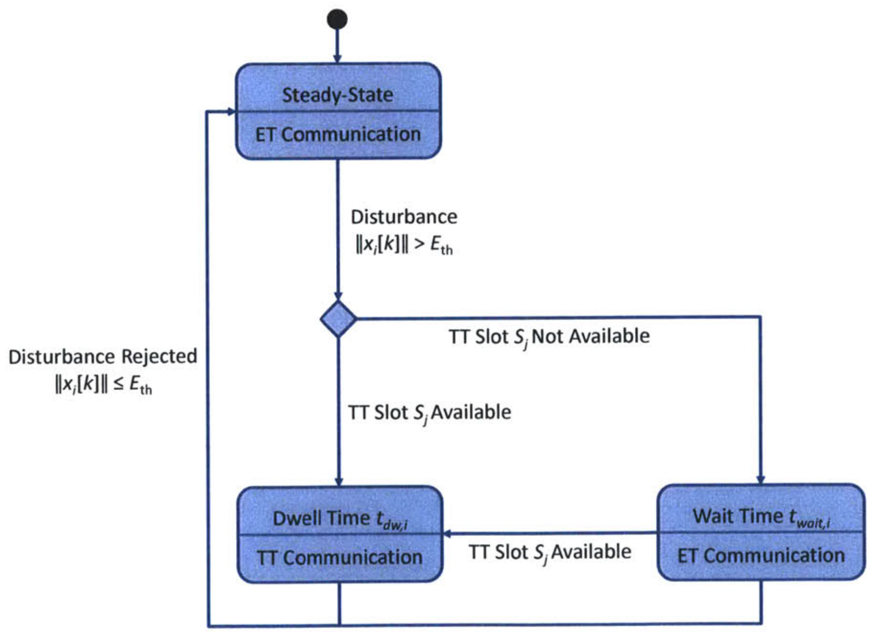


Figure 2-4: Basic switching scheme between time- and event-triggered communication depending on the system states.

2.7 Controller Design

The design of the controller is focused on one that switches between two cases, where in the first case the control input is sent through the TT segment and in the second case, the control input is sent through the ET segment. The discussions in the previous sections lead to the following control application models for transmitting using each respective segment in the communication cycle.

2.7.1 TT Controller Design

In the TT segment, the underlying discrete-time model experiences zero delay (i.e., $\tau_i = 0$), and hence is of the form

$$x_i[k+1] = \Phi_i x_i[k] + \Gamma_{0,i} u_i[k] + \hat{D}_i[k] \tag{2.8}$$

where

$$\begin{aligned} \Phi_i &= e^{A_i h_i} \\ \Gamma_{0,i} &= \int_0^{h_i} e^{A_i s} ds\, B_i. \end{aligned} \tag{2.9}$$

This can be seen directly from (2.3) where $\tau_i' = 0$, which follows from $\tau_i = 0$ in (2.2). The control input u_i to be sent over the TT segment is then designed using this model of the plant.

2.7.2 ET Controller Design

In the ET segment, the underlying discrete-time model has a delay τ_i. Since we assume, as mentioned in Section 2.2.2, that $\tau_i \leq h_i$[1], equation (2.6) follows, and

[1]It should be noted that if $\tau_i > h_i$, the inputs in (2.10) are altered as $u[k - \tau_{1,i}]$ and $u[k - \tau_{1,i} - 1]$ with $\tau_{1,i} \neq 0$. The control strategy that is discussed in the following section can be suitably altered to accommodate the resulting dynamics.

therefore equation (2.3) reduces to the form

$$x_i[k+1] = \Phi_i x_i[k] + \Gamma_0 u_i[k] + \Gamma_1 u_i[k-1] + \hat{D}_i[k] \tag{2.10}$$

where Φ_i and $\Gamma_{0,i}$ and $\Gamma_{1,i}$ are discrete-time equivalent system matrices and are given by

$$\begin{aligned} \Phi_i &= e^{A_i h_i} \\ \Gamma_{0,i} &= \int_0^{h_i-\tau_i'} e^{A_i s} ds\, B_i \\ \Gamma_{1,i} &= \int_{h_i-\tau_i'}^{h_i} e^{A_i s} ds\, B_i \end{aligned} \tag{2.11}$$

Equations (2.10) and (2.11) can be compactly represented in state-space form as

$$X_i[k+1] = \begin{bmatrix} \Phi_i & \Gamma_{1,i} \\ 0 & 0 \end{bmatrix} X_i[k] + \begin{bmatrix} \Gamma_{0,i} \\ I \end{bmatrix} u_i[k] + \begin{bmatrix} I \\ 0 \end{bmatrix} \hat{D}_i[k] \tag{2.12}$$

where $X_i[k] = \begin{bmatrix} x_i[k]^T & u_i[k-1]^T \end{bmatrix}^T$. In (2.8) and (2.12), the discrete equivalent disturbance $\hat{D}(k)$ is given by (2.5). The control input u_i to be sent over the ET segment is then designed using this plant model that takes into account the communication delay τ_i that may be experienced in transmission.

2.7.3 Control Strategy

The plant models for the ET and TT segments can be controlled using a variety of control strategies including state feedback and model predictive control (MPC). For example, one can use state feedback control of the form

$$u[k] = Gx[k] \tag{2.13}$$

where G is chosen using optimal control principles [8] or an extended state feedback with Linear Matrix Inequalities (LMI) methods [2] of the form

$$u[k] = KX[k]. \tag{2.14}$$

Using whichever control strategy desired, the controller is designed in particular for use in the TT and ET segments, respectively. The resulting control sequences u_i are then the control input into the plants to obtain the closed-loop responses. It is noted that because of the delay present in the ET segment, the response time of the system in the TT segment, ξ_i^{TT} is significantly less than ξ_i^{ET}, the response time of the system in the ET segment.

Chapter 3

Non-Monotonic System Behavior

In order to schedule n control applications C_i $(i \in \{1, 2 \dots n\})$ onto m TT slots S_j $(j \in \{1, 2 \dots m\})$ where $m < n$ such that each control application meets their desired response time or deadline ξ_i^d, the relation between dwell time $t_{dw,i}$ and wait time $t_{wait,i}$ must be known. This relation is dependent on the behavior of the system dynamics and is described in this chapter. Particularly, a non-monotonic system behavior is looked at closely as this has a significant impact on the $t_{dw,i}$ and $t_{wait,i}$ relation.

3.1 Impact of Non-Monotonic System Behavior

For illustration, consider the cases $\xi_i = t_{wait,i} + t_{dw,i}$ and $\xi_j = t_{wait,j} + t_{dw,j}$. When the system behaves in a monotonic manner,

$$t_{wait,i} > t_{wait,j} \Rightarrow t_{dw,i} < t_{dw,j}.$$

However, in reality, the system behavior often has *non-monotonic* phase, i.e.,

$$t_{wait,i} > t_{wait,j} \Rightarrow t_{dw,i} > t_{dw,j}.$$

Figure 3-1 shows a typical non-monotonic response (parameters will be explained in the later sections). In this figure, the non-monotonic behavior is shown between

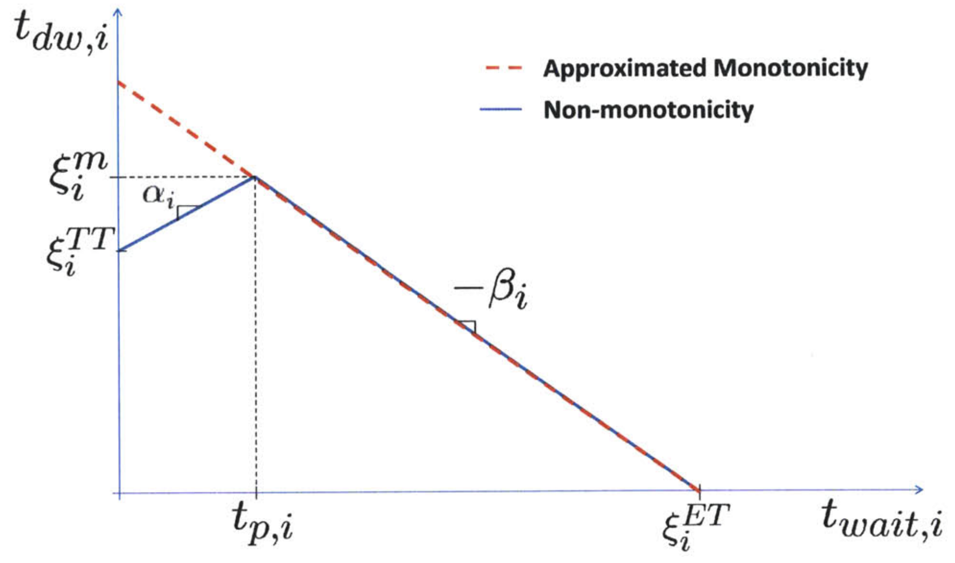


Figure 3-1: Relation between $t_{dw,i}$ and $t_{wait,i}$

$t_{wait,i} = 0$ to $t_{p,i}$ and the system behavior becomes monotonic with $t_{wait,i} > t_{p,i}$.

In general, for an arbitrary initial condition (i.e., $x(t)$ at $t = 0$ in (2.1) or $x(0)$) of the plant, it is quite possible for the system states, even with a well-designed controller, to decay to their steady-state value in a *non-monotonic manner*. For illustration, we consider a second order system with two states $x_1(t)$ and $x_2(t)$. Figure 3-2 shows non-monotonic behavior with $x_1(0) = -1$ and $x_2(0) = 5$ while the same system behaves monotonically with $x_1(0) = -1$ and $x_2(0) = -1$ as shown in Figure 3-3. It is noticeable from Figure 3-2 that both $x_1(t)$ and $x_2(t)$ initially increase before they delay to their steady state values. This particular nature in state response results in non-monotonicity in system behavior as shown in Figure 3-1 between $t_{wait,i} = 0$ to $t_{wait,i} = t_{p,i}$. On the other hand, Figure 3-3 shows that the states $x_1(t)$ and $x_2(t)$ increase monotonically and the resulting system behavior is shown in Figure 3-1 with $t_{wait,i} > t_{p,i}$. Hence, this property is dependent on the initial condition $x(0)$, as shown in Figure 3-2 and 3-3.

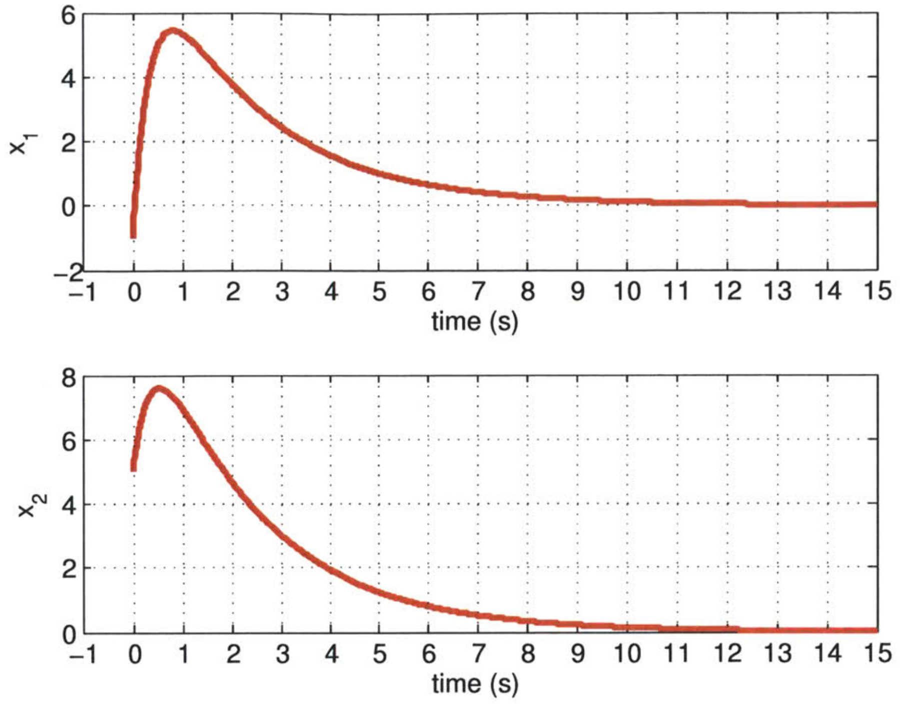


Figure 3-2: Non-monotonicity with $x_1(0) = -1$ and $x_2(0) = 5$.

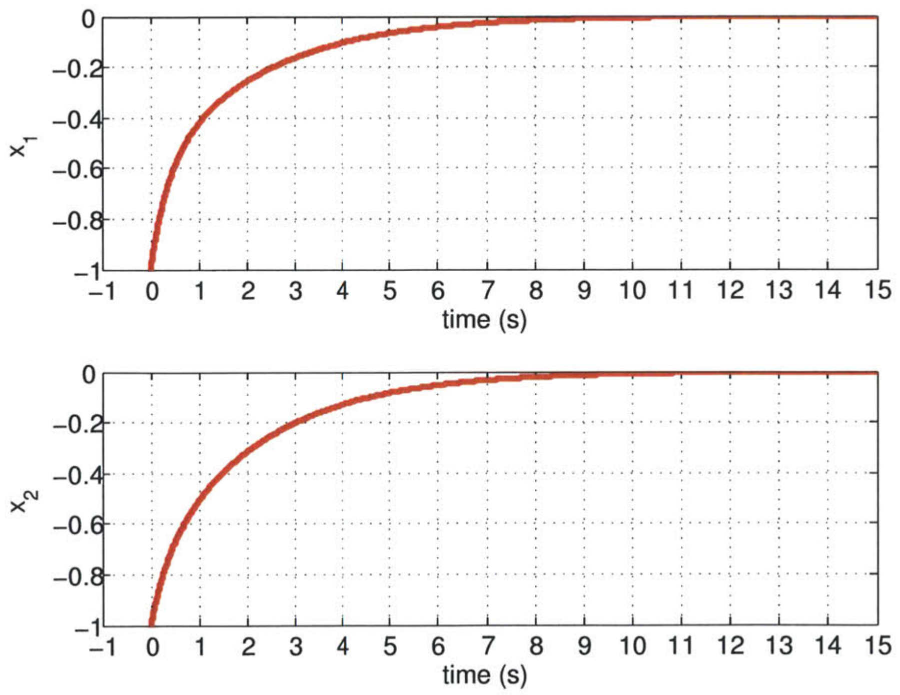


Figure 3-3: Monotonicity with $x_1(0) = -1$ and $x_2(0) = -1$

3.2 Analytical Description of Non-Monotonicity

The focus of these discussions is the manner in which the state x of the system

$$x[k+1] = Ax[k] \tag{3.1}$$

evolves as k increases, given that the system (3.1) is stable. Stability of (3.1) implies that [18] a symmetric positive-definite matrix P exists such that it solves the matrix equation

$$A^T PA - P = -I \tag{3.2}$$

where I is an identity matrix. This is because the positive definite function $V(x[k])$ defined as in (3.3) has the property that

$$\begin{aligned} V(x[k+1]) - V(x[k]) &= x^T[k](A^T PA - P)x[k] \\ &= -||x[k]||^2 < 0. \end{aligned}$$

This does not imply, however, that $||x[k]||$ will decrease to zero in a monotonic manner. Non-monotonicity occurs if there is any instant k_0 where

$$||x[k_0 + 1]|| > ||x[k_0]||.$$

This can only occur if in some regions of the state-space, the norm of the state x increases and in other regions it decreases. This behavior is found in the class of systems where the matrix A in (3.1) is indefinite. That is, given a stable system as in (3.1), non-monotonicity occurs if A is *indefinite.*

The following are theorems from [22] that define and provide necessary and sufficient conditions for an indefinite A.

Let m and n be nonzero vectors such that $m'Am > 0$ and $n'An < 0$.

Theorem 3.2.1. *A is indefinite if and only if the magnitude of the state vector increases with time for $x = m$ and decreases with time for $x = n$.*

A necessary and sufficient condition for A to be indefinite:

Theorem 3.2.2. *Let A be a real matrix. Then A is indefinite if and only if $A + A^T$ is indefinite.*

Another necessary and sufficient condition for A to be indefinite:

Theorem 3.2.3. *In order that a real matrix A be indefinite, it is necessary and sufficient that at least one of the following two conditions is met:*

1. *There exists a negative principle minor of $A + A^T$ of even order.*
2. *Not all principle minors of $A + A^T$ of odd order have the same algebraic sign.*

There are many different classes of matrices that will produce a non-monotonic behavior of the system, but the following is an important, general, class of matrices that are indefinite:

Theorem 3.2.4. *If A is in phase-variable form then A is indefinite whenever either of the following conditions are met:*

1. *A is a 2×2 matrix of nonunity determinant,*
2. *A is an $n \times n$ matrix where $n > 2$.*

3.2.1 Remark About Performance Measure

Instead of using $\|x_i[k]\|$ as specified in Section 2.3, one can use a more general positive definite function of the system states x_i in the form of

$$V(x_i) = x_i^T P x_i \tag{3.3}$$

where P is a symmetric positive definite matrix. If $V(x_i)$ is small, one can then infer that x_i is proportionately small with the proportionality constant determined by the eigenvalues of P. Thus, the performance measure that is monitored to switch between the static segment and the dynamic segment can also be a positive definite function $V(x_i[k])$ defined as in (3.3). In the case, the matrix P is chosen such that it coincides with the solution of (3.2), then it follows that $V(x_i)$ will always be monotonic. If on the other hand, the norm $\|x_i[k]\|$ is used as a performance measure, then it can, as outlined above, lead to a non-monotonic response for a general dynamic system (3.1). The advantage of choosing V, which is essentially a weighted norm of x_i, is the underlying monotonicity. The disadvantage is that its computation requires the determination of P which in turn, by virtue of (3.2), is dependent on the system model. The use of $\|x_i[k]\|$ avoids direct dependence on the system model, but suffers from a possible non-monotonic response which is the focus on this work.

3.3 Examples of Non-Monotonic Systems

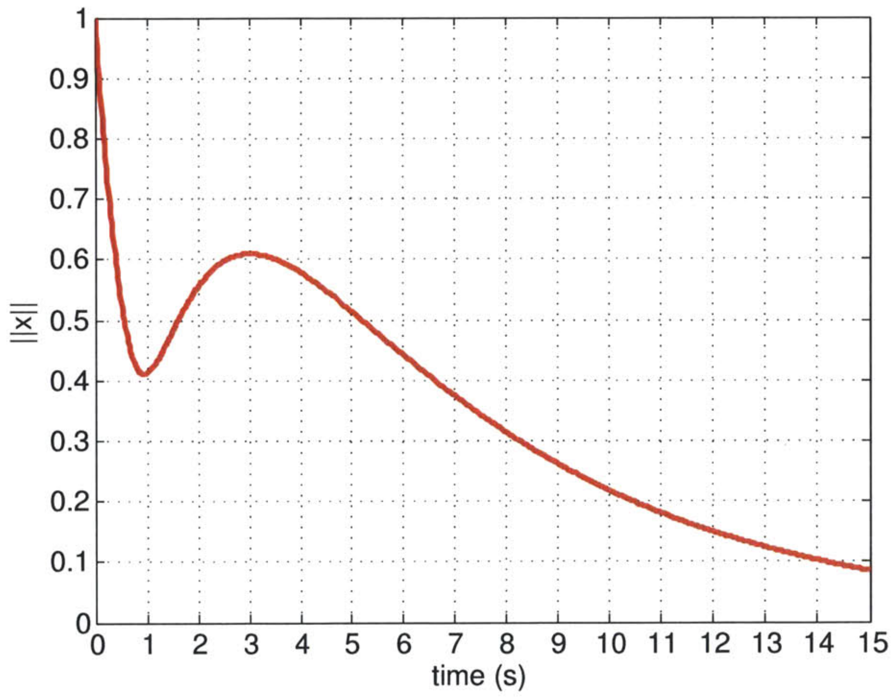


Figure 3-4: Example of non-monotonic behavior in third-order system

Consider the system in (3.1) where A is given as

$$A = \begin{bmatrix} -1.30 & -0.12 & 0.90 \\ -0.62 & -0.25 & 0.50 \\ -0.75 & -0.06 & 0.40 \end{bmatrix}.$$

The eigenvalues of this system are -0.7576, -0.1829 and -0.2095. Since they are all less than zero, this system is stable. Using Theorem 3.2.3, it can be determined whether A is indefinite and therefore whether non-monotonicity occurs in this system.

$$A + A^T = \begin{bmatrix} -2.60 & -0.74 & 0.15 \\ -0.74 & -0.50 & 0.44 \\ 0.15 & 0.44 & 0.80 \end{bmatrix}$$

The principal minors of $A + A^T$ of even order are 0.7524 and -0.5936. Since at least one of these is negative, condition 1 of Theorem 3.2.3 is satisfied and A is indefinite. Therefore, this system, although stable, has non-monotonic behavior. This system is simulated and the non-monotonic behavior can be seen for an initial condition $x_0 = [1 \quad 0 \quad 0]^T$ in Figure 3-4.

As another example, consider the system in (3.1) where A is in phase-variable form. The general form of A and $A + A^T$ are

$$A = \begin{bmatrix} 0 & 1 \\ a & b \end{bmatrix}$$

$$A + A^T = \begin{bmatrix} 0 & 1+a \\ 1+a & 2b \end{bmatrix}$$

The principle minor of $A + A^T$ of even order is $-(1+a)^2$. This is always negative if $a \neq -1$ so condition 1 of Theorem 3.2.3 is satisfied and A is indefinite if $a \neq -1$. Therefore, this system has non-monotonic behavior if $a \neq -1$. Since the determinant of A is $-a$, this also proves Theorem 3.2.4, condition 1.

3.4 Relation Between Dwell and Wait Times

Given that closed-loop system responses can have the above non-monotonic response, its effect on the total response time is examined when switching the transmission of control inputs between TT and ET segments. An accurate determination of this

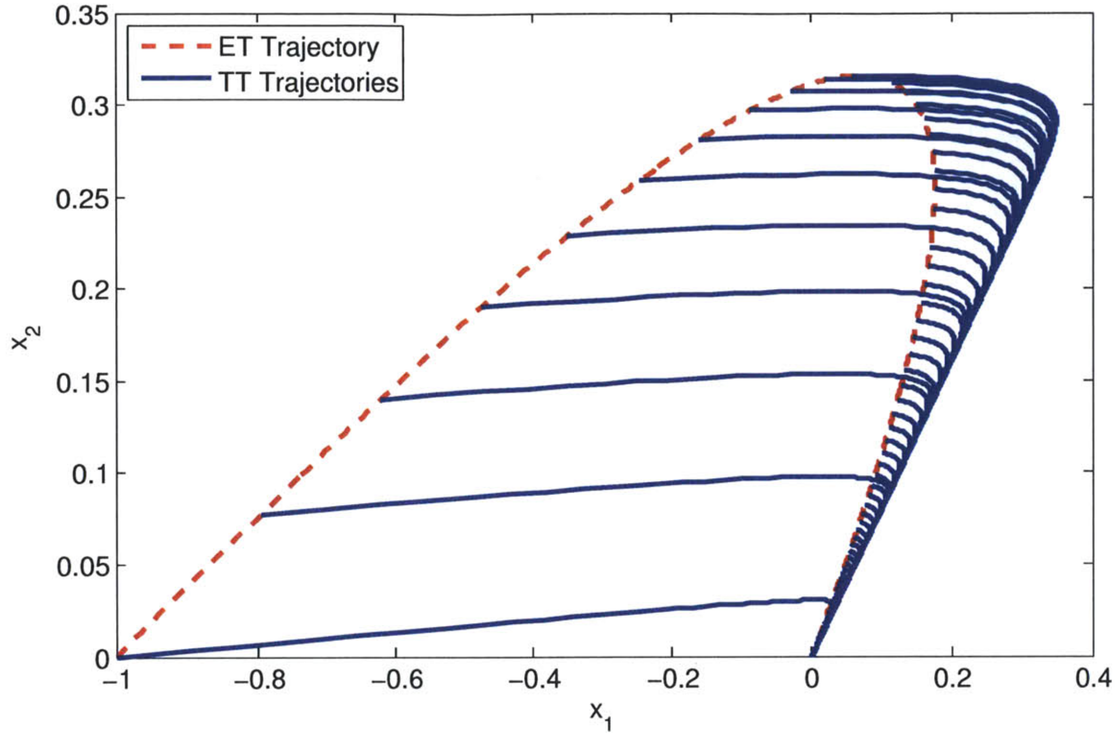


Figure 3-5: TT for different initial conditions from ET

effect will in turn affect the scheduling of control applications C_i to TT slots S_j in the schedulability analysis, discussed in Chapter 4.

Consider a second-order model for a control application as (2.1) where $x_i(t)$ is given as $\begin{bmatrix} x_1(t) & x_2(t) \end{bmatrix}^T$. In Figure 3-5, the set of state trajectories x_1 and x_2 of the corresponding closed-loop systems in the ET and TT segments are shown. We assume that the disturbance is such that the state starts at the point $(-1, 0)$ in the phase-space. The red curve represents the trajectory of the ET closed-loop system that starts at this point. Following this trajectory, the system is brought back to stability at $(0, 0)$. The blue curves represent the trajectories of the TT closed-loop system at different initial conditions of the ET-trajectory. Depending on at what point in the trajectory of the ET system does the switch to TT, the TT system takes a different (blue) path to return to the origin.

In the case of a non-monotonic closed-loop response of the ET and TT systems as in the above example, the dwell time $t_{dw,i}$ initially increases as the amount of wait time increases since the TT closed-loop system receives worse initial conditions resulting in longer trajectories to stability. After this initial increase and as the wait time continues to increase, $t_{dw,i}$ then decreases with $t_{wait,i}$ when better initial conditions are received resulting in shorter, monotonic trajectories that converge to zero. The maximum value $t_{dw,i}$ reaches before then decreasing is called the maximum response time ξ_i^m and occurs after $t_{p,i}$ time, or the time to peak, of wait time $t_{wait,i}$. This relationship between the dwell time and wait time can be depicted in Figure 3-1 using two piecewise linear curves. From this relation, the dwell time t_{dw} can be written as a piecewise function of the wait time t_{wait}.

$$t_{dw,i} = \begin{cases} \xi_i^{TT} + \alpha_i t_{wait,i} & t_{wait,i} < t_{p,i} \\ \beta_i \xi_i^{ET} - \beta_i t_{wait,i} & t_{wait,i} > t_{p,i} \end{cases} \tag{3.4}$$

where

$$\begin{aligned} \alpha_i &= \frac{\xi_i^m - \xi_i^{TT}}{t_{p,i}} \\ -\beta_i &= \frac{\xi_i^m}{\xi_i^{ET} - t_p, i} \end{aligned} \tag{3.5}$$

As stated earlier, since the response time of the system in the TT segment is significantly less than the response time of the system in the ET segment, $\beta_i < 1$. The total response time ξ_i is then

$$\xi_i = t_{wait,i} + t_{dw,i} \tag{3.6}$$

$$\xi_i = \begin{cases} t_{wait,i} + \xi_i^{TT} + \alpha_i t_{wait,i} & t_{wait,i} < t_{p,i} \\ t_{wait,i} + \beta_i \xi_i^{ET} - \beta_i t_{wait,i} & t_{wait,i} > t_{p,i} \end{cases} \tag{3.7}$$

$$\xi_i = \begin{cases} \xi_i^{TT} + (1 + \alpha_i) t_{wait,i} & t_{wait,i} < t_{p,i} \\ \beta_i \xi_i^{ET} + (1 - \beta_i) t_{wait,i} & t_{wait,i} > t_{p,i} \end{cases} \tag{3.8}$$

This model is general for both non-monotonic and monotonic closed-loop responses. For a monotonic closed-loop response, the time to peak t_p would be zero and the maximum response time $\xi_i^m = \xi_i^{TT}$. While this linear fit provides a good approximation of the actual relationship between dwell time and wait time, a good upper bound approximation, or monotonic approximation (MA), could be to extend the negative slope part of the curve, shown as a dashed line in Figure 3-1. This monotonic approximation will result in a more conservative scheduability analysis since in the region with the positive slope, the extension will predict a higher $t_{dw,i}$ than what it actually is and higher dwell times result in the use of more static slots.

Chapter 4

Schedulability Analysis

In this chapter, the scheduling of control applications C_i to TT slots S_j is described. It is shown that the non-monotonic system behavior described in the previous chapter affects this scheduling analysis. Inclusion of this effect into the analysis yields a more accurate and less conservative result than has been done previously.

4.1 Assumptions for Schedulability

The overall goal is to assign n control applications to m shared TT slots (with $m < n$) such that $\xi_i < \xi_i^d$ for $i \in \{1, 2 \ldots n\}$, i.e., the performance requirements are met.

4.1.1 Task-to-Slot Mapping

In order to meet the overall goal, there are two possible mappings from the tasks, i.e., the control applications C_i, to the TT slots S_j for $j \in \{1, 2 \ldots m\}$. Consider, for example, three control applications C_1, C_2, C_3 and two slots S_1, S_2. One option is *partitioned scheduling*: C_1 and C_2 are assigned to S_1 and C_3 to S_2. With this scheduling, if C_1 and C_2 experience disturbances at the same time and S_2 is free, C_1 or C_2 cannot use S_2. The task-to-slot mapping is fixed in this option. The other option is *global scheduling*: C_1, C_2, C_3 can all use S_1, S_2 based on their necessity. For this schedulability analysis, a partitioned scheme is used as shown in Figure 4-1. That is, each control application is assigned to a single TT slot such that it always uses the same slot when transmitting over the static segment. Each slot then contains one or more control applications C_i that request its access to transmit control input for $t_{dw,i}$ time after the occurrence of a disturbance. The slot must provide this amount of service to these applications within their desired response time ξ_i^d from when the disturbance occurs. From the schedulability perspective, the desired response time ξ_i^d of each control application then acts as a *deadline*.

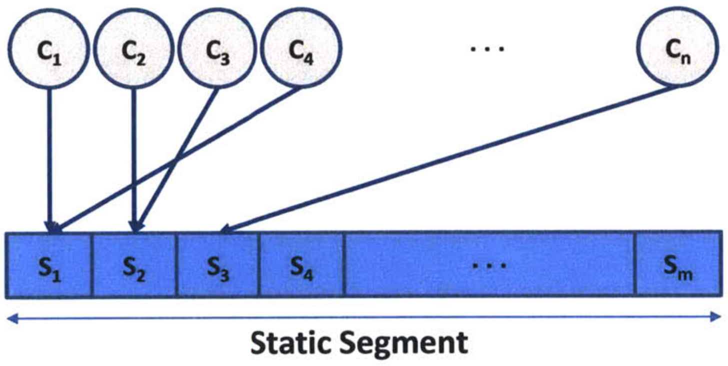


Figure 4-1: Example of partitioned scheduling on the static segment

4.1.2 Occurrence of Disturbances

For the schedulability analysis, the pattern of when the disturbances occur for each controller is assumed to be *sporadic* with a minimum inter-arrival time r_i, where $\xi_i^d \leq r_i$ for every C_i. Under this assumption, each control application should have enough time to reject a disturbance before another one occurs. The sources of the disturbances are also assumed to be independent of each other so that one disturbance will not cause another one to occur. The *worst-case disturbance pattern* is therefore when disturbances occur simultaneously with their respective minimum inter-arrival times r_i for all control applications C_i in one slot.

4.1.3 Schedule Within a Slot

The schedule between the control applications sharing a single TT slot is designed to be priority based. Every control application C_i is assigned a priority according to their deadline ξ_i^d. The shorter the deadline of a C_i, the higher priority it has in the shared slot. Therefore when multiple control applications C_i simultaneously request access to a shared slot to transmit control input, the control application with the highest priority is granted access to the slot. The other control applications must then wait until the higher-priority control application finishes using the TT slot. While they wait, these control applications will use their respective ET slots to transmit control input.

Since a control application switches from ET to TT only once in the process of a particular disturbance rejection, it blocks the TT slot for $t_{dw,i}$ time once it gets access to it. It will block the slot regardless of whether a higher-priority application requests access within the $t_{dw,i}$ time. In other words, the sending of control input over the static segment is non-preemptive. Therefore, if the slot is being used by one control application, another application C_i will have to wait for access to the slot regardless of its priority. This increases the wait time $t_{wait,i}$ of the blocked C_i which affects its dwell time t_{dw} as discussed earlier. Also, because $\beta_i > 1$, the increase in wait time also increases the total response time ξ_i. The schedulability of a control application C_i on a shared slot is then guaranteed if for every possible wait time $t_{wait,i}$, $\xi_i \leq \xi_i^d$.

4.2 Worst-Case Response Time

In order to determine the schedulability of C_i to a particular TT slot, the maximum possible wait time, $\hat{t}_{wait,i}$, that leads to the worst-case response time, $\hat{\xi}_i$, must be found. It is assumed that all higher-priority applications C_j interfering with C_i require their maximum possible dwell time on the slot, ξ_j^m. This assumption leads to a conservative schedulability analysis since dwell time of the higher-priority application can be less depending on the blocking time suffered by C_j. Using this assumption, the worst-case response time for control application C_i occurs when it needs to use the TT slot at the same time as all the higher-priority applications C_j and it also suffers blocking time due to a lower-priority application since the sending of messages is non-preemptive. This worst-case can occur when a lower-priority is using the slot and all higher-priority applications C_j and C_i have disturbances occur simultaneously.

To find the worst-case response time of a task under a fixed-priority non-preemptive scheduling, the response times of all jobs of that task must be computed within its *maximum busy period.* The task here is given by a control application C_i sending its control input over the shared static slot. The maximum busy period $t_{max,i}$ of C_i is then the longest time which the slot is constantly being used by higher-priority control applications and by C_i. It is assumed that $t_{max,i} \leq r_i$ for all C_i so there is only one transmission of messages of C_i over the TT segment within its busy period $t_{max,i}$. Using this assumption, only the response time ξ_i of the sole job of C_i within $t_{max,i}$ needs to be computed to obtain the worst-case response time $\hat{\xi}_i$. This can be done using (3.8) and the maximum possible wait time $\hat{t}_{wait,i}$ described above to yield the following iterative equation for ξ_i.

$$\xi_i = \begin{cases} \xi_i^{TT} + (1+\alpha_i)b_i + (1+\alpha_i)\sum_{j=1}^{i-1} \left\lceil \frac{\xi_i}{r_j} \right\rceil \xi_j^m & \text{Case 1} \\ \beta_i \xi_i^{ET} + (1-\beta_i)b_i + (1-\beta_i)\sum_{j=1}^{i-1} \left\lceil \frac{\xi_i}{r_j} \right\rceil \xi_j^m & \text{Case 2} \end{cases} \tag{4.1}$$

Cases 1 and 2 are defined depending on the value of ξ_i, as

$$\text{Case 1: } b_i + \sum_{j=1}^{i-1} \left\lceil \frac{\xi_i}{r_j} \right\rceil \xi_j^m < t_{p,i}$$

$$\text{Case 2: } b_i + \sum_{j=1}^{i-1} \left\lceil \frac{\xi_i}{r_j} \right\rceil \xi_j^m > t_{p,i}$$

where $b_i = \max_{k=i+1}^{n}(\xi_k^m)$ is the maximum possible blocking time due to lower-priority applications suffered by C_i and n is the number of control applications. For the rest of the paper, it is assumed that applications are sorted in order of decreasing priority so that C_j has a higher priority than C_i and C_i has a higher priority than C_k for

$1 \leq j < i < k \leq n$. Equation (4.1) can be solved iteratively starting from

$$\xi_i = \begin{cases} \xi_i^{TT} + (1+\alpha_i)b_i & b_i < t_{p,i} \\ \beta_i \xi_i^{ET} + (1-\beta_i)b_i & b_i > t_{p,i} \end{cases} \tag{4.2}$$

and proceeding until either (a) ξ_i becomes greater than ξ_i^d or (b) converges to a value $\hat{\xi}_i$. If ξ_i is greater than ξ_i^d, then it implies that C_i is not schedulable on the shared slot. If it converges, then C_i can meet its deadline and is schedulable on the slot.

For either case (a) or case (b) to occur, it is necessary that the implicit equations in (4.1) have a solution. In what follows, we discuss the analytical framework that ensures the existence of such a solution.

4.2.1 Convergence to Fixed Point

We note that equation (4.1) essentially is a difference equation. Its solutions converging to a fixed value implies that (4.1) has a fixed point. Expressing both cases 1 and 2 of (4.1) as

$$\xi_i(k+1) = f_p(\xi_i(k)) \tag{4.3}$$

where

$$f_p(\xi_i(k)) = a_p + c_p \sum_{j=1}^{i-1} \left\lceil \frac{\xi_i(k)}{r_j} \right\rceil \xi_j^m$$

$$a_1 = \xi_i^{TT} + (1+\alpha_i)b_i, \quad a_2 = \beta_i \xi_i^{ET} + (1-\beta_i)b_i$$

$$c_1 = 1+\alpha_i, \qquad c_2 = 1-\beta_i$$

we state conditions below in Theorem 4.2.1 under which (4.3) has a fixed point $\hat{\xi}_i$ to which all solutions converge.

Theorem 4.2.1. *Consider the nonlinear difference equation*

$$\begin{aligned} \xi_i(k+1) &= f(\xi_i(k)) \\ f(\xi_i(k)) &= a + c\sum_{j=1}^{i-1} \left\lceil \frac{\xi_i(k)}{r_j} \right\rceil \xi_j^m \end{aligned} \tag{4.4}$$

and let m be defined as

$$m = c\sum_{j=1}^{i-1} \frac{\xi_j^m}{r_j}.$$

If $a > 0$ and $m < 1$, then all solutions of (4.4)

1. *have a fixed point $\hat{\xi}_i$, and*

2. *converge to $\hat{\xi}_i$.*

Proof. The right hand side of (4.4) can be bounded with linear upper and lower bounds using the following property of the ceiling function:

$$x \leq \lceil x \rceil < x + 1 \tag{4.5}$$

Linear combinations of $\lceil x \rceil$ are used to get lower and upper bounds of $f(\xi_i(k))$ as

$$a + c\sum_{j=1}^{i-1} \frac{\xi_j^m}{r_j}\xi_i(k) \leq f(\xi_i(k)) < a + c\sum_{j=1}^{i-1} \frac{\xi_j^m}{r_j}\xi_i(k) + c\sum_{j=1}^{i-1} \xi_j^m.$$

The above inequality can be simplified as

$$L(\xi_i(k)) \leq f(\xi_i(k)) < L(\xi_i(k)) + c\sum_{j=1}^{i-1} \xi_j^m$$

where

$$L(\xi_i(k)) = a + m\xi_i(k)$$
$$m = c\sum_{j=1}^{i-1} \frac{\xi_j^m}{r_j}.$$

By definition, the fixed point $\hat{\xi}_i$ has the property

$$\xi_i(k) = \hat{\xi}_i \implies \xi_i(k+1) = \hat{\xi}_i.$$

That is, $\hat{\xi}_i$ lies on the line

$$\xi_i(k+1) = \xi_i(k).$$

It follows that $f(\xi_i(k)) = \hat{\xi}_i$ has a solution if $L(\xi_i(k)) = \hat{\xi}_i$ has a solution. The latter holds if $a > 0$ and $m < 1$, which proves the first part of Theorem 4.2.1.

To prove that $\xi_i(k)$ will converge to $\hat{\xi}_i$ for all initial conditions, we consider the Lyapunov function candidate

$$V(\xi_i(k)) = (\xi_i(k) - \hat{\xi}_i)^2.$$

If the fixed point $\hat{\xi}_i$ is stable, then the following inequality must be satisfied:

$$V(\xi_i(k+1)) - V(\xi_i(k)) < 0 \tag{4.6}$$

Substituting the Lyapunov function candidate into (4.6), we obtain that

$$\begin{aligned} V(\xi_i(k+1)) - V(\xi_i(k)) &= (\xi_i(k+1) - \hat{\xi}_i)^2 - (\xi_i(k) - \hat{\xi}_i)^2 \\ &= (2\hat{\xi}_i - \xi_i(k) - \xi_i(k+1))(\xi_i(k) - \xi_i(k+1)). \end{aligned}$$

In what follows, we show that $V(k)$ is a Lyapunov function by showing that (1) $2\hat{\xi}_i - \xi_i(k) - \xi_i(k+1) \geq 0$ and (2) $\xi_i(k) - \xi_i(k+1) < 0$.

(1) Let $\xi_i(k)$ be such that $\xi_i(k) < \hat{\xi}_i$. Another property of the ceiling function is

$$x_1 < x_2 \rightarrow \lceil x_1 \rceil \leq \lceil x_2 \rceil. \tag{4.7}$$

Since $\xi_i(k) < \hat{\xi}_i$, inequality (4.7) implies that

$$f(\xi_i(k)) \leq f(\hat{\xi}_i). \tag{4.8}$$

Since $\hat{\xi}_i$ is a fixed point,

$$f(\hat{\xi}_i) = \hat{\xi}_i,$$

and with equation (4.4) therefore implies that (4.8) can be rewritten as

$$\xi_i(k+1) \leq \hat{\xi}_i.$$

Then again since $\xi_i(k) < \hat{\xi}_i$, it follows that

$$(\hat{\xi}_i - \xi_i(k)) + (\hat{\xi}_i - \xi_i(k+1)) > 0.$$

(2) Given $\xi_i(0) = 0$, prove $\xi_i(k) < \xi_i(k+1)$ by induction. Base Case:

$$\xi_i(0) < \xi_i(1)$$

$$0 < a + c\sum_{j=1}^{i-1} \left\lceil \frac{\xi_i(0)}{r_j} \right\rceil \xi_j^m = a$$

$a > 0$ so the base case holds.

Inductive Step: If $\xi_i(k) < \xi_i(k+1)$ holds, then $\xi_i(k+1) < \xi_i(k+2)$ holds. Assume $\xi_i(k) < \xi_i(k+1)$ holds, then using

$$\xi_1 < \xi_2 \rightarrow f(\xi_1) \leq f(\xi_2)$$

from before gives,

$$f(\xi_i(k)) \leq f(\xi_i(k+1))$$

Then from (4.4), it follows that

$$\xi_i(k+1) \leq \xi_i(k+2)$$

Since both the basis and inductive step have been proved, then by induction $\xi_i(k) < \xi_i(k+1)$ for all k.

It further implies that (4.6) is satisfied and hence the fixed point $\hat{\xi}_i$ is stable. □

A direct application of Theorem 4.2.1 to equation (4.3) implies that in cases 1 and 2 of (4.1), ξ_i converges to a fixed point $\hat{\xi}_i$ if $a_p > 0$ and $m_p < 1$ for $p \in \{1, 2\}$.

Algorithm 4.1 Scheduling parameters algorithm

```
Require: Plant parameters A and B, delay d, sampling period h, and time t
     {Form equivalent discrete model for ET}
 1: Phi = expm(A*h)
 2: Gamma_1 = expm(A*(h-d))*integrate(expm(A*s)*B,'s',0,d)
 3: Gamma_0 = integrate(expm(A*s)*B,'s',0,h-d)
 4: AdET = [Phi Gamma_1 ; zeros(1,2) 0]
 5: BdET = [Gamma_0 ; 1]
 6: Compute control input $u$
 7: Form closed-loop $A_{cl,ET}$
 8: Define initial conditions x0ET for ET system
 9: xET = simulate($A_{cl,ET}$,t,x0ET)
10: for $i$ = 1 to length(xET) do
11:     xnormET(i) = norm(xET(i,1:2),2)
12: end for
     {Set inital conditions for TT controller as ET trajectory}
13: x0TT = xET(:,1:2)
     {Form equivalent discrete model for TT}
14: AdTT = Phi
15: BdTT = integrate(expm(A*s)*B,'s',0,h)
16: Compute control input $u$
17: Form closed-loop $A_{cl,TT}$
     {Compute norms for TT time responses using ET trajectory as initial conditions
     for TT controller}
18: for $i$ = 1 to length(x0TT) do
19:     xTT = simulate($A_{cl,TT}$,t,x0TT(i,:))
20:     for $j$ = 1 to length(xTT) do
21:         xnormTT(j) = norm(xTT(j,:),2)
22:     end for
23:     SettTimeTT(i) = settling_time(xnormTT,t)
24: end for
25: $t_{dw}$ = SettTimeTT
26: $t_{wait}$ = t
27: Calculate linear approximations of $t_{dw}$ vs $t_{wait}$
28: Obtain parameters from linear approximations
```

4.3 Scheduling Parameters

Some of the scheduling parameters needed for (4.1) are chosen, but most are determined by the system behavior. The minimum inter-arrival time for the disturbances, r_i, and the deadline, ξ_i^d, are chosen by the designer. The response time with a TT implementation, ξ_i^{TT}, response time with an ET implementation, ξ_i^{ET}, time to peak, $t_{p,i}$, and maximum response time, ξ_i^m, are parameters obtained for each control application as outlined below using equations from Section 2.7.
Procedure for Parameters:

1. Begin with a plant model given as in (2.1)

2. Using A and B in (2.1), compute Φ, Γ_0, Γ_1 given sampling time h and delay $\tau \neq 0$ for ET, $\tau = 0$ for TT using (2.11) and (2.9)

3. Compute $u_{ET}[k] = G_{ET}x[k]$ and $u_{TT}[k] = G_{TT}x[k]$ where G_{ET} and G_{TT} and chosen so that the closed-loop systems are stable

4. Form closed-loop systems for ET and TT using $u_{ET}[k]$ and $u_{TT}[k]$ as follows:

$$ET_{cl}\begin{cases} X[k+1] = \begin{bmatrix} \Phi & \Gamma_1 \\ 0 & 0 \end{bmatrix} X[k] + \begin{bmatrix} \Gamma_0 \\ I \end{bmatrix} u_{ET}[k] \\ y[k] = \begin{bmatrix} C & 0 \end{bmatrix} X[k] \\ u_{ET}[k] = G_{ET}x[k] \end{cases}$$

$$TT_{cl}\begin{cases} x[k+1] = \Phi x[k] + \Gamma_0 u_{TT}[k] \\ y[k] = Cx[k] \\ u_{TT}[k] = G_{TT}x[k] \end{cases}$$

5. Compute ET closed-loop response $x_{ET} = \text{simulate}(ET_{cl}, k, x_{ET0})$ using an initial condition x_{ET0}

6. For $i = 0$ to k_{final}

   (a) Compute TT closed-loop response x_{TT} = simulate($TT_{cl}, k, x_{ET}(i)$) using $x_{ET}(i)$ as the initial condition

   (b) For $j = 0$ to k_{final}

      i. Compute norm of $x_{TT}(j)$

   (c) Compute settling time $t_{dw}(i)$ of the norms

7. Set $t_{wait} = k$

8. Using Step 7, i.e., $\{t_{wait}, t_{dw}(t_{wait})\}$, determine a piecewise-linear function t_{dw} of t_{wait}, as in Figure 3-1

9. Obtain parameters ξ^{TT}, ξ^{ET}, ξ^{m} and t_p from linear approximations as in Figure 3-1

An algorithm of this procedure is shown in Algorithm 4.1. This algorithm is carried out in MATLAB and the full code can be seen in Appendix A.

4.4 Slot Scheduling Algorithm

Using the parameters obtained as specified in 4.3, Algorithm 4.2 schedules a given set of control applications C_i to a number of static segment slots using (4.1) to decide whether a control application C_i is schedulable on a particular slot S_j. The algorithm begins with one slot and inserts the control applications C_i as long as they are schedulable on the slot as per the schedulability analysis. From that analysis, a C_i is schedulable on a particular slot if it can meet its deadline ξ_i^d when assigned to the slot. The algorithm attempts to schedule all C_i to the existing slots. However, if a C_i cannot be scheduled on an existing slot, a new slot is added and the C_i is inserted there. Once all C_i have been scheduled, the algorithm returns the number of static segment slots that were necessary. This algorithm was implemented as a C++ program and the full program can be seen in Appendix B.

Algorithm 4.2 Slot scheduling algorithm

```
Require: $n$ control applications $C_i$ with parameters $r_i$, $\xi_i^d$, $\xi_i^{TT}$, $\xi_i^{ET}$, $\xi_i^m$ and $t_{p,i}$
1: Sort $C_i$ in order of decreasing priority
2: num_slots = 1
3: for $i = 1$ to $n$ do
4:   for $s = 1$ to num_slots do
5:     if Schedulable(slot($s$), $C_i$) then
6:       Insert $C_i$ into slot($s$)
7:       break
8:     else if $s ==$ num_slots then
9:       num_slots = num_slots + 1
10:      Insert $C_i$ into slot(num_slots)
11:      break
12:    end if
13:  end for
14: end for
15: return num_slots
```

4.5 Simulation of Algorithm

Consider six control applications distributed as shown in Figure 2-1. The parameters of each control application are shown in Table 4.1, columns 2 through 7. All six control applications were chosen to have a non-monotonic closed-loop response and therefore a non-monotonic relation between dwell time $t_{dw,i}$ and wait time $t_{wait,i}$ as shown in Figure 3-1.

Table 4.1: Example control applications (parameters in ms)

C_i	r_i	ξ_i^d	ξ_i^{TT}	ξ_i^{ET}	ξ_i^m	$t_{p,i}$	$\hat{\xi}_i$	MA $\hat{\xi}_i$
1	2000	85	36	200	46	16	84.5	36.0
2	2000	500	144	550	184	44	317.1	327.3
3	1500	85	36	200	46	16	84.5	36.0
4	2000	300	144	400	184	32	292.0	300.0
5	5000	1000	576	2000	736	160	576.0	576.0
6	600	600	216	700	276	56	216.0	216.0

4.5.1 Communication Details

The communication protocol is assumed to be FlexRay with a cycle length of 5 ms. The static segment has 2 ms length and it is divided into 10 slots. The rest of the cycle is assigned to the dynamic segment. The proposed scheduling analysis is applied to these control applications and the necessary number of slots that guarantee all necessary requirements is determined using Algorithm I.

4.5.2 Results

The schedulability analysis yields four slots: $S_1 = \{C_1, C_3\}$, $S_2 = \{C_4, C_2\}$, $S_3 = \{C_6\}$ and $S_4 = \{C_5\}$. The worst-case response times for each control application is shown in Table 4.1, column 8. This is compared to the same schedulability analysis using a monotonic approximation (MA) for the relation between dwell and wait times, as shown in Figure 3-1, which yields five slots: $S_1 = \{C_1\}$, $S_2 = \{C_3\}$, $S_3 = \{C_4, C_2\}$, $S_4 = \{C_6\}$ and $S_5 = \{C_5\}$. For this case, the worst-case response time for each control application is shown in Table 4.1, column 9. This result was expected since in the region with the positive slope in Figure 3-1, the monotonic approximation (the dashed line in the figure) will predict a higher $t_{dw,i}$ than what it actually is. The higher dwell times then result in the use of more TT slots which is shown with this example.

4.5.3 Discussion

The results show that the proposed schedulability analysis allows for a reduced number of TT slots with respect to a purely TT scheme, which would require six slots, or a schedulability analysis that does not take into consideration the non-monotonic system behavior. When non-monotonic system behavior is considered in the analysis, the requirement on the number of TT slots is four in order to guarantee the control performance which saves 33% TT communication usage. On the other hand, with a simplified monotonic assumption on system behavior, the system needs five TT slots to guarantee the desired control performance. In this case, the saving is only 17%.

Chapter 5

Simulation

The distributed control applications setup as detailed in Chapter 2 is simulated using MATLAB/Simulink in combination with Truetime. Truetime is a MATLAB/Simulink based simulator for real-time control systems developed by the Department of Automatic Control at Lund University. Simulink is used to simulate the plants and controllers, while Truetime is able to simulate the Flexray communication network and the timing of the transmission of control input. The simulation shows that given n control applications C_i $(i \in 1, 2 \ldots n)$ and m TT slots S_j $(j \in 1, 2 \ldots m)$ where $m < n$ and given by the schedulability analysis proposed in Chapter 4, all the control applications do actually meet their desired response time or deadline ξ_i^d.

5.1 Distributed Architecture

Using Simulink, each control application is distributed as specified in Section 2.2. A typical control application is shown in Figure 5-1. The controller gets the plant states from the plant and sends a computed control input through the communication bus back to the plant just as shown previously in Figure 2-1. Recall from Section 2.2 that the execution times of the tasks $T_{s,i}$, $T_{c,i}$ and $T_{a,i}$ are assumed negligible. Because of this, the plant states are fed directly to the controller and likewise, the control input is directly applied to the plant. Also, the control input is immediately available at each sampling period, meaning the simulation assumes zero execution time for the

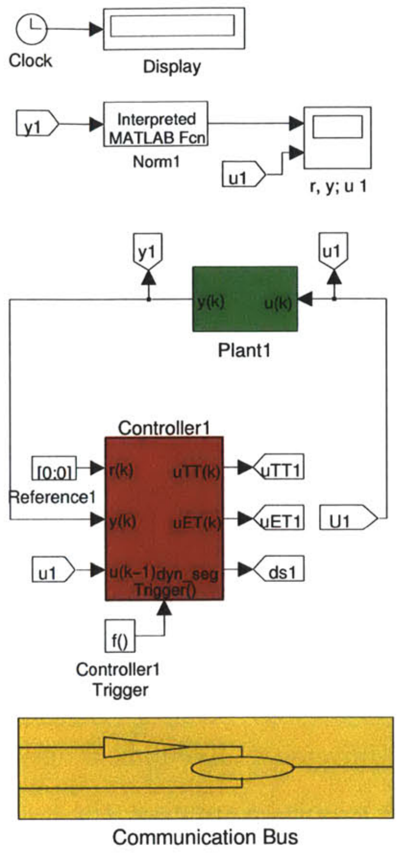


Figure 5-1: Simulink model of typical control application of distributed system

computation of the control input.

5.2 Hybrid Communication Bus

Using the Truetime Network, Send and Receive blocks, as well as logic to arbitrate and route all the signals correctly, the hybrid communication bus is simulated as specified in Section 2.2.2. The typical setup of the communication bus for two control applications sharing a single TT slot is shown in Figure 5-2. The blue colored Truetime Send blocks send control inputs through the static segment of the communication cycle, while the orange colored Truetime Send blocks send control inputs through the dynamic segment of the communication cycle.

The Truetime Network block simulates the Flexray specification where the communication bandwidth is divided into communication cycles of equal length. The

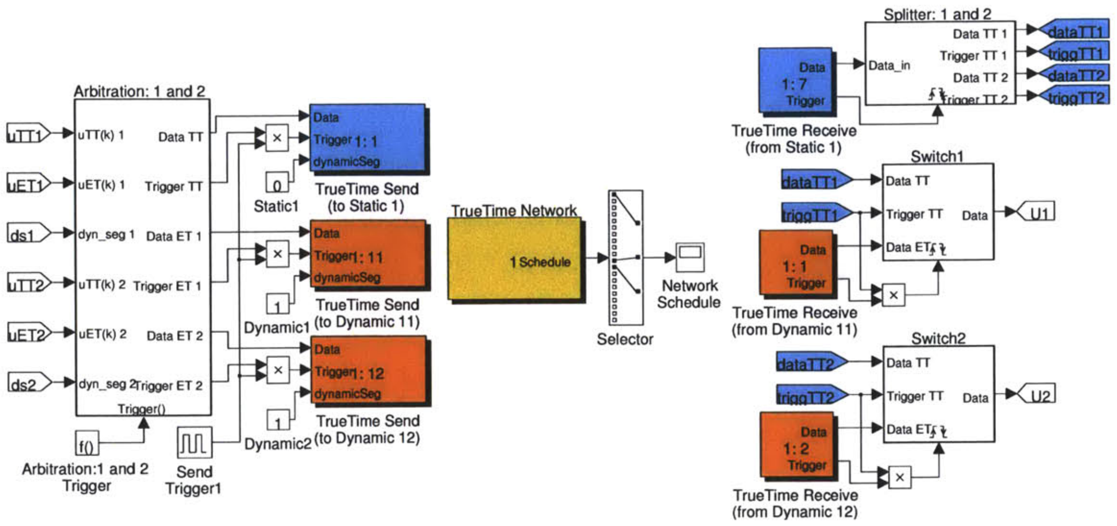


Figure 5-2: Simulink model of communication bus for sharing a single TT slot

length of the communication cycle is predetermined by the length of the static and dynamic segments, which is specified in this block.

5.2.1 Static Segment

The static segment is specified in the Truetime Network block by its slot length and schedule. As stated previously, the slot length is chosen to be large enough to fit every possible control input. For the schedule, every slot is given a number and the order they are placed in the schedule determines its order on the segment. Therefore, if the slots are given numbers in increasing numerical order and placed in that same order, then the segment will be identical to the static segment shown in Figure 2-2. Using the slot length and the length of the schedule, or number of slots, the total length of the static segment can be determined.

The Truetime Send blocks are assigned over which segment to send a particular control input and when being sent through the static segment, they are also assigned a specific slot in the segment. These blocks, therefore, determine which control applications are assigned to each TT slot.

5.2.2 Dynamic Segment

The dynamic segment is specified in the Truetime Network block by its mini-slot length and schedule. In Truetime, a mini-slot works much like a slot in the static segment where each one is given a number and the order they are placed in the schedule determines its order on the segment. The key difference, however, is that the mini-slot length is typically chosen to be smaller than the length necessary to send a control input, meaning that it takes several mini-slots to send a message over this segment. Whereas in the static segment, a transmission of a control input is assigned to a particular slot for the length of time of that slot, in the dynamic segment, a transmission is a assigned to a particular mini-slot to begin its sending, but it continues to transmit using the length of time of other mini-slots until it is done sending.

As stated in Section 2.2.2, the dynamic segment is priority-based. To simulate this, the schedule must be chosen in such a way there are no mini-slots that will never be able to begin a transmission, meaning there is a stream of control inputs (assigned to that mini-slot) that are never able to be send. One such schedule is $[1, 2, 3, 1, 2, 3, 1, 2, 3, 1]$, where a complete transmission of a control input takes four mini-slots. Using this schedule, any two mini-slots could send their assigned control inputs over the dynamic segment within one communication cycle with the higher priority one (i.e., the smaller numbered one) transmitting first.

5.3 Switching Scheme

The switching scheme detailed in Section 2.6 and Figure 2-4 is simulated using Simulink and Truetime. The majority of the work occurs within the Arbitration block shown in Figure 5-2. The internal logic of this block is shown in Figure 5-3 in the case that two control applications are sharing a single TT slot. Each controller sends this block their computed control inputs to be sent either over the static or dynamic segment as well as a signal indicating which segment is requested. This signal is determined by checking whether the system states are in steady-state or not

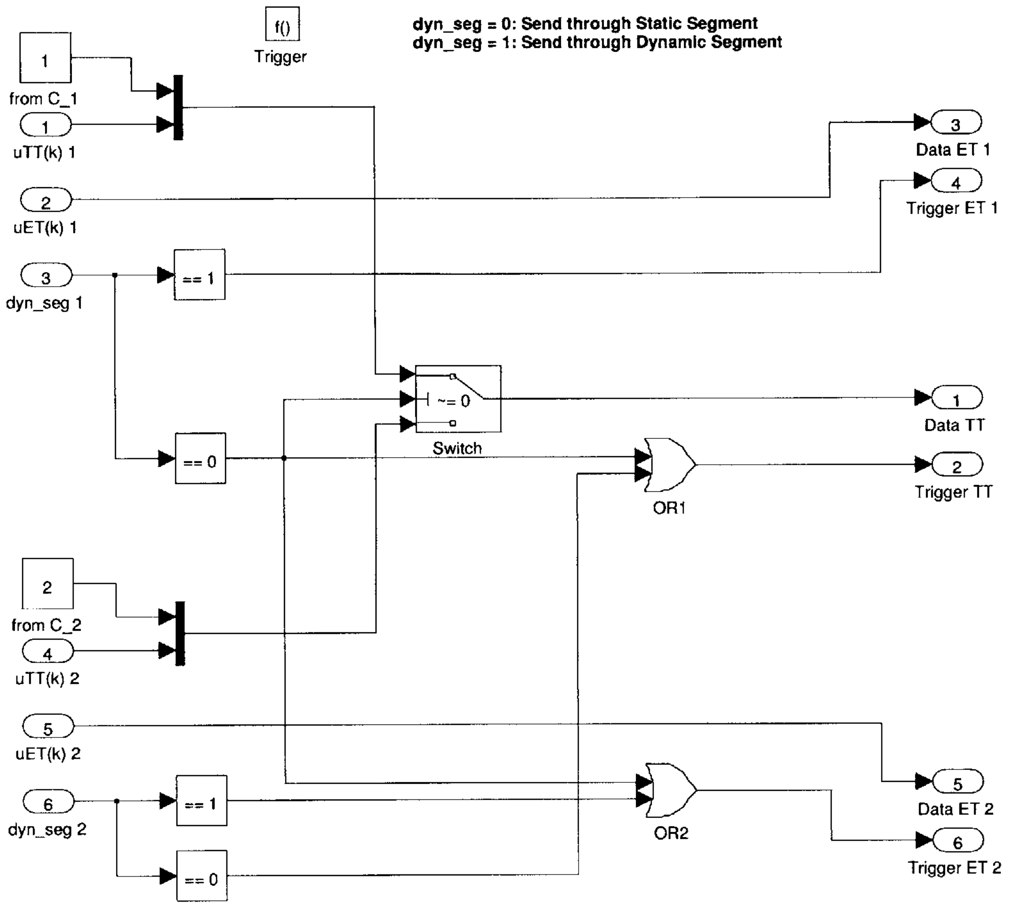


Figure 5-3: Simulink model of switching scheme

as indicated in an earlier section. If the system is not in steady-state (i.e., a disturbance arrives), the static segment is requested. Otherwise, the dynamic segment is requested.

The block checks whether the higher-priority control application C_1 requests the static segment. If it does, then it gets to send its particular control input through the shared TT slot and the other control application C_2 sends its control input that goes through the dynamic segment. If the higher-priority control application does not request the static segment, then it sends its particular control input through the dynamic segment and the block checks whether the other control application requests the static segment. If it does, then it gets to send its particular control input through the shared TT slot. The logic in this block therefore ensures that if both control applications assigned to a single TT slot request access to it, only the one

Table 5.1: Example plants

C_i	A_i	B_i
1	$\begin{bmatrix} -0.76 & 0.43 \\ -0.34 & 0.07 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
2	$\begin{bmatrix} -1.50 & 1.39 \\ -0.75 & 0.58 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
3	$\begin{bmatrix} 0.57 & -1.43 \\ 1.93 & -3.07 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
4	$\begin{bmatrix} -1.00 & 0.66 \\ -0.60 & 0.30 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
5	$\begin{bmatrix} -0.92 & 0.56 \\ -0.50 & 0.17 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
6	$\begin{bmatrix} -5.00 & 3.50 \\ -3.00 & 2.00 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$

with a higher priority will receive access and the other will have to wait and use the dynamic segment in the meantime as specified in Section 2.6.

5.4 Simulation of Example System

Consider six continuous-time plants of the form in (2.1) with system parameters as shown in Table 5.1. By applying Theorem 3.2.3 directly to all six A_i it can be seen that all six plants have non-monotonic behavior.

These plants are distributed and communicate over a hybrid communication bus as discussed in Section 5.1. Figure 5-4 shows the Simulink model showing the six control applications with the plants from Table 5.1 and the communication bus.

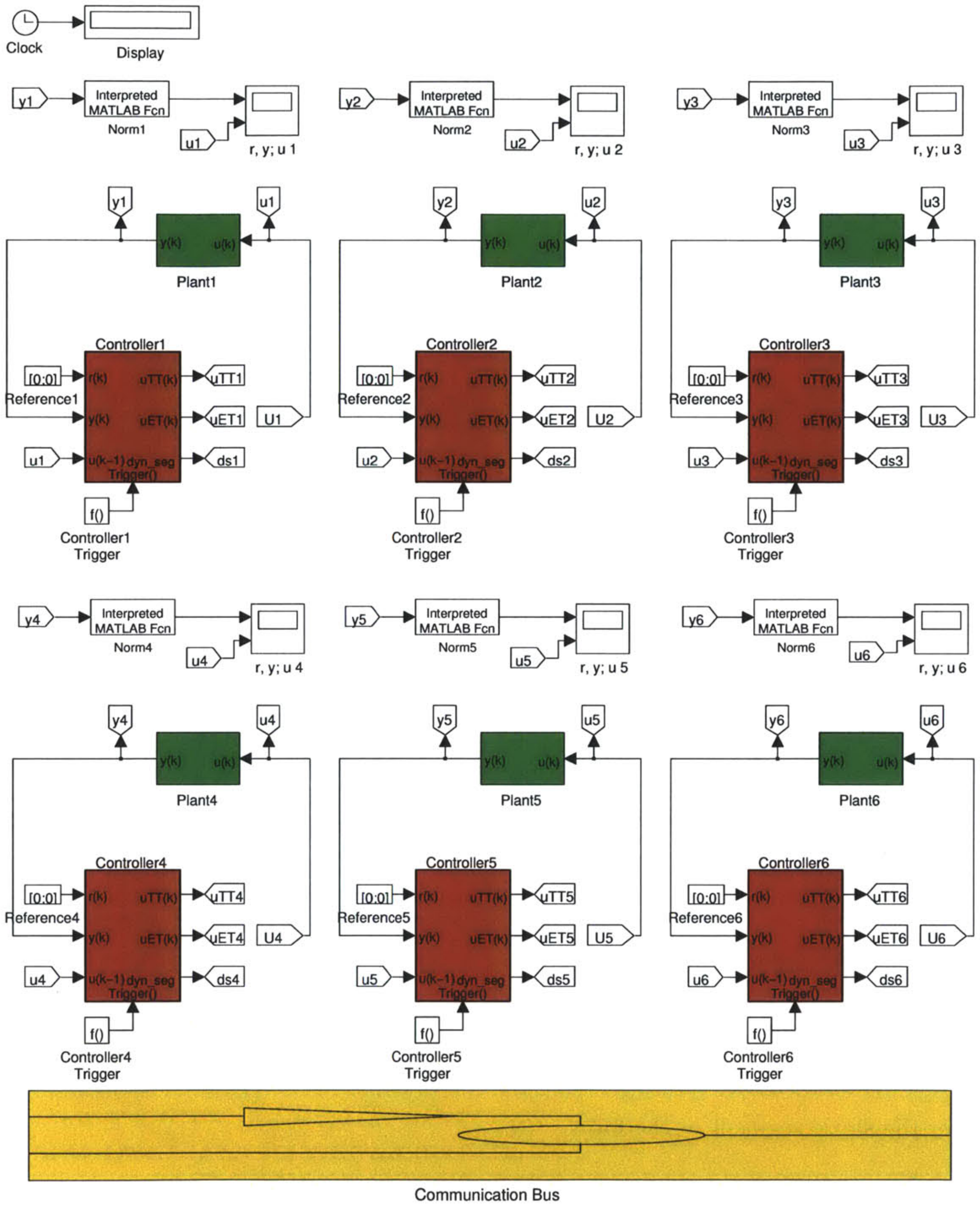


Figure 5-4: Simulink model of distributed system

5.4.1 Communication Details

The communication protocol is assumed to be FlexRay with a cycle length of 5 ms. The static segment has 2 ms length and it is divided into 10 slots. Therefore,

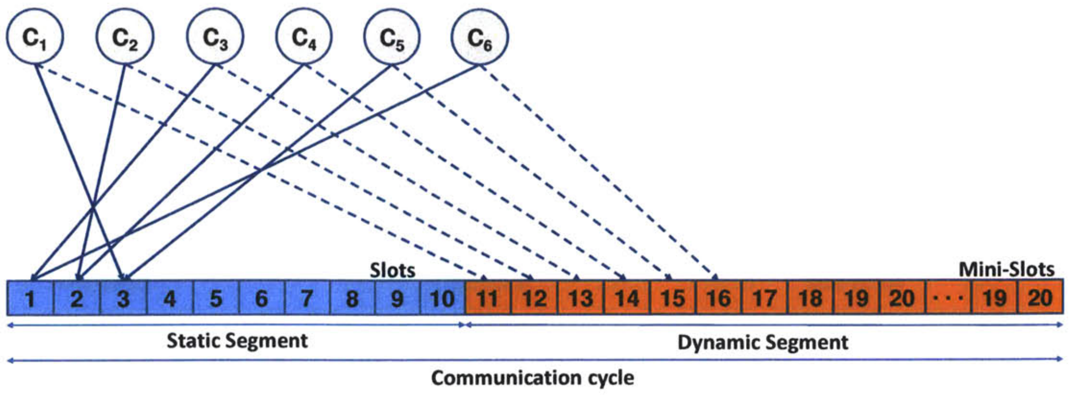


Figure 5-5: Communication protocol used in simulation

each slot can transmit control inputs of size 250 bytes. The slots are numbered 1 to 10 and are placed in that order on the static segment. The rest of the cycle is assigned to the dynamic segment. The mini-slots on the dynamic segment are numbered 11 to 20 and are smaller in size compared to the static slots. It takes 11 mini-slots in order to transmit a control input in the dynamic segment. Figure 5-5 shows one communication cycle of the communication protocol used. It also shows the scheduling of the control applications to the TT slots, determined by the schedulability analysis below, and how each control application is assigned their own mini-slot in the dynamic segment.

5.4.2 Controller Design

A variety of control strategies can be used to control the plant models for the static (2.8) and dynamic (2.12) segments. In this simulation, state feedback control is used for the static segment of the form

$$u_i[k] = G_{TT,i}x_i[k] \tag{5.1}$$

where $G_{TT,i}$ is chosen using optimal control principles. For the dynamic segment, an extended state feedback is used of the form

$$u_i[k] = G_{ET,i}X_i[k]. \tag{5.2}$$

where $G_{ET,i}$ is chosen using Linear Matrix Inequalities (LMI) methods.

5.4.3 Schedule of TT Slots

Using the procedure outlined in Section 4.3, we can obtain the parameters for the slot scheduling algorithm in Algorithm 4.2 to determine the number of TT slots necessary so that all the control applications meet their desired response time ξ_i^d. These parameters are shown in columns 2 through 7 of Table 5.2.

Table 5.2: Example control applications (parameters in s)

C_i	r_i	ξ_i^d	ξ_i^{TT}	ξ_i^{ET}	ξ_i^m	$t_{p,i}$	$\hat{\xi}_i$	MA $\hat{\xi}_i$
1	200	9	1.6809	11.6243	5.3027	2.2675	8.57086	6.5877
2	20	6.25	2.578	8.5865	2.9487	1.342	5.88212	3.495
3	15	2	0.38562	3.9724	0.64081	0.68966	1.51785	1.58611
4	200	7.5	2.495	10.3982	4.0258	1.9215	6.48666	4.9384
5	20	8.5	2.7534	10.633	4.577	1.9714	8.11936	5.6187
6	6	6	0.71207	7.94	0.92249	0.66886	1.55448	1.68436

Using these parameters in the slot scheduling algorithm yields three TT slots: $S_1 = \{C_3, C_6\}$, $S_2 = \{C_2, C_4\}$, $S_3 = \{C_5, C_1\}$. Figure 5-6 shows this assignment of the control applications to TT slots in the communication bus. The worst-case response times for each control application is shown in Table 5.2, column 8.

This is compared to the same schedulability analysis using a monotonic approximation (MA) for the relation between dwell and wait times, as shown in Figure 3-1, which yields five slots: $S_1 = \{C_3, C_6\}$, $S_2 = \{C_2\}$, $S_3 = \{C_4\}$, $S_4 = \{C_5\}$ and $S_5 = \{C_1\}$. For this case, the worst-case response time for each control application is shown in Table 5.2, column 9. This result was expected since in the region with the positive slope in Figure 3-1, the monotonic approximation (the dashed line in the figure) will predict a higher $t_{dw,i}$ than what it actually is. The higher dwell times then result in the use of more static slots which is shown with this example.

5.4.4 Results

The response of each control application is shown in Figures 5-7. The blue region denotes the time that control inputs to the plant were sent through the static segment, while the orange region denotes the time that control inputs were sent through the dynamic segment. The dashed red lines indicated the region of steady-state. Recall that the control application should send the corresponding control input through the static segment when, in this case, $||x|| > 0.1$ and through the dynamic segment otherwise.

The schedule of the communication bus is shown in Figure 5-8. It can be seen that the control applications use the static (TT) and dynamic (ET) segments as the switching scheme indicates from earlier.

5.4.5 Discussion

In cost-sensitive domains like automotive, an efficient resource is one of the most important design consideration. These results clearly show that a tighter resource usage (i.e., using fewer TT slots) is achievable using the presented scheme and analysis. In general, a large number of feedback loops run on in-vehicle Electrical/Electronic (E/E) architecture incorporating various functionalities such as adaptive cruise control, idle speed control, active suspension control, engine control etc. In the most cases, the control loops are implemented in a distributed fashion leading to the need to access the shared communication network. In such scenarios, the application of the presented analysis can potentially achieve significant resource saving.

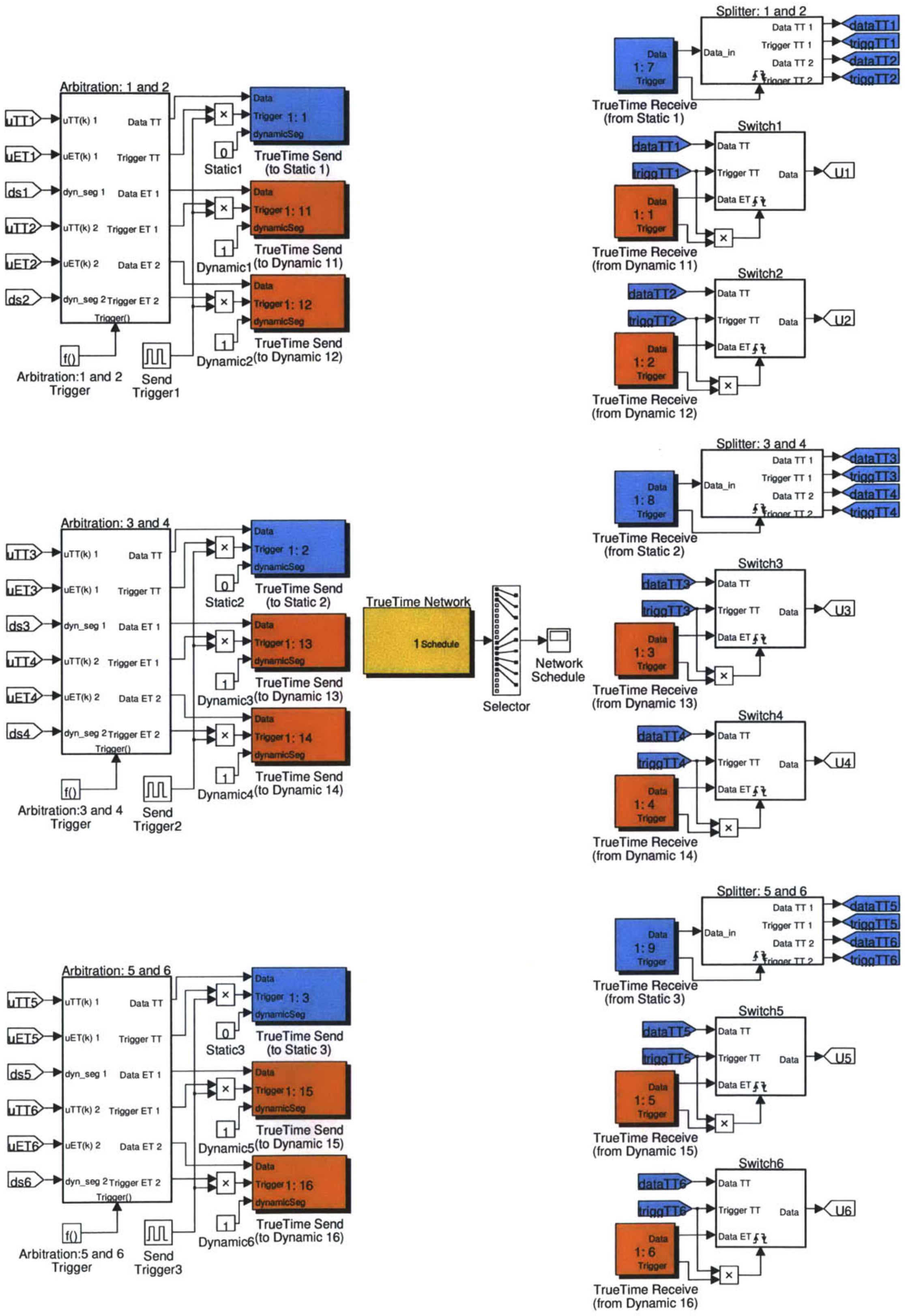
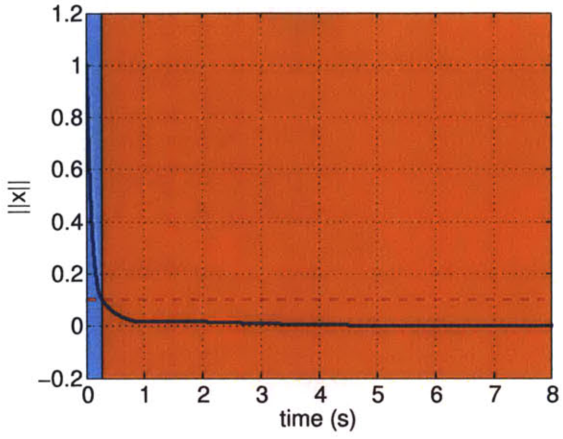
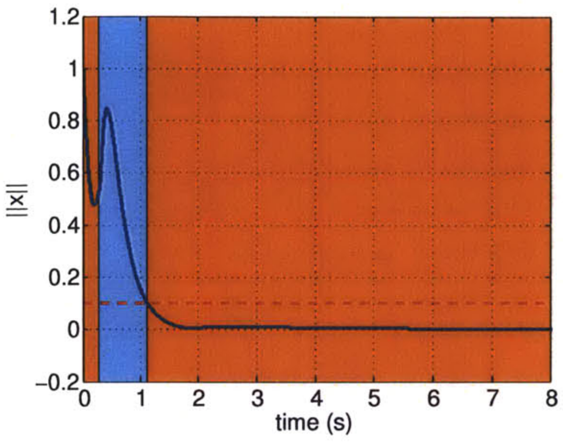

Arbitration: 1 and 2
Arbitration: 3 and 4
Arbitration: 5 and 6
TrueTime Send (to Static 1)
TrueTime Send (to Dynamic 11)
TrueTime Send (to Dynamic 12)
TrueTime Network
Selector
Network Schedule
Splitter: 1 and 2
Switch1
Switch2
TrueTime Receive (from Static 1)
TrueTime Receive (from Dynamic 11)
TrueTime Receive (from Dynamic 12)

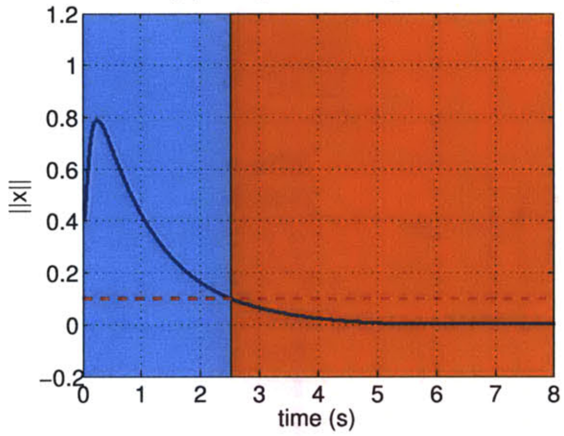
Figure 5-6: Simulink model of communication bus



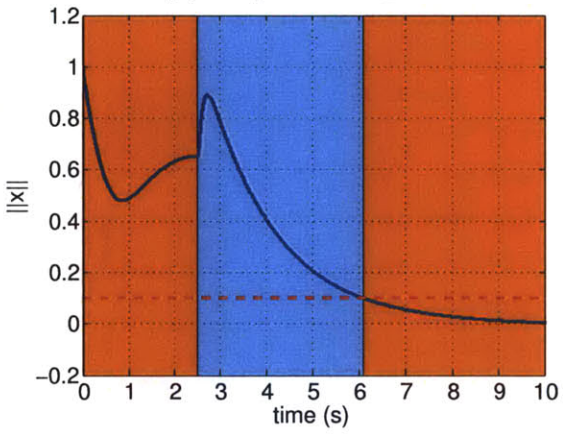
(a) Response of C_1



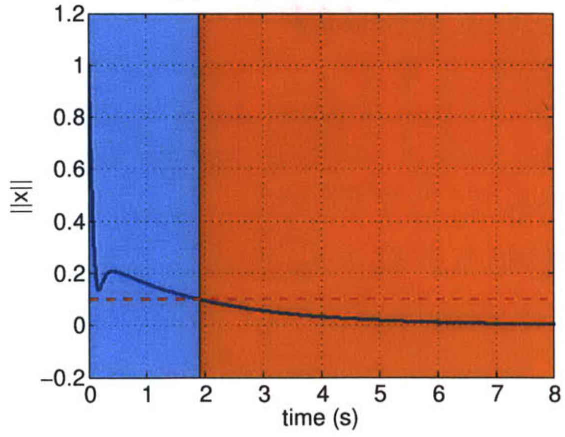
(b) Response of C_2



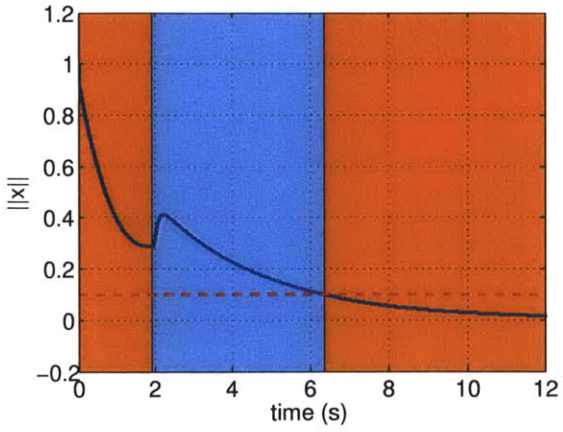
(c) Response of C_3



(d) Response of C_4



(e) Response of C_5



(f) Response of C_6

Figure 5-7: Responses of the six control applications

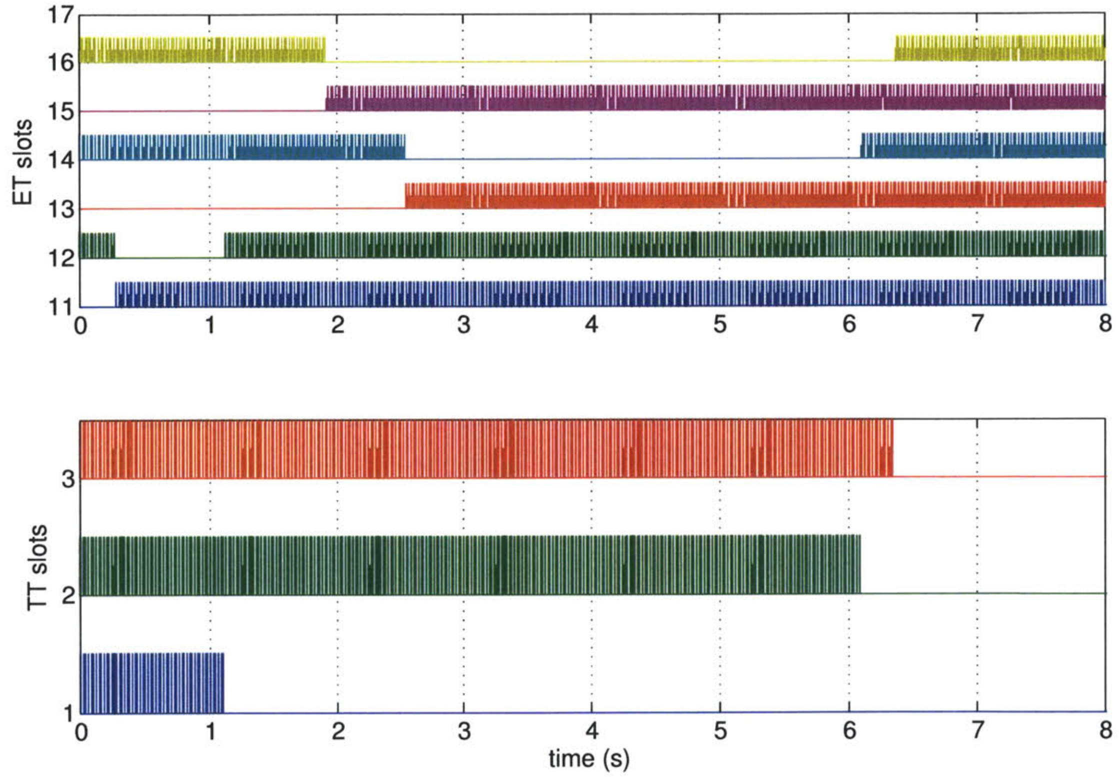


Figure 5-8: Schedule of the communication bus

Chapter 6

Conclusion

This thesis deals with a resource efficient distributed implementation of control applications on embedded platforms. In general, an aggressive control algorithm with good control performance, e.g., shorter settling time requires higher resource on an embedded platform. On the other hand, a relatively less aggressive control algorithm with poor control performance requires lower resource on an embedded platform. The presented work aims to achieve an implementation which ensures that the resource allocated to the control applications is as close as possible to what resource is necessary to achieve a desired control performance. The idea is to switch between these two possible control algorithms and the corresponding resource allocations such that the "average" resource usage is close to what is necessary for the control applications. Towards this, the work presented here used the non-monotonic system dynamics of some systems and accommodated it in the presented schedulability analysis to achieve a tighter resource dimensioning. The technique is illustrated considering a hybrid communication bus as a shared resource and reduced the usage of the time-triggered communication on that bus. Further, it is possible to adapt the presented analysis for a wider class of combinations of embedded resource and control algorithm. For example, an aggressive control algorithm can be realized with a higher sampling rate and hence, higher resource usage on the processor. In such cases, the presented technique can also be applied to vary the sampling rate of the control applications to achieve a tight resource dimensioning in terms of processor utilization.

Appendix A

Scheduling Parameters MATLAB Code

```
function distributed_w_schedan_init
clc, clear all
global b sampleTime
%% Fixed for all cases
b = [1;0];
sampleTime = 0.02; % Sampling interval = 20 ms

assignin('base', 'B', b);
assignin('base', 'sampleTime', sampleTime);
%% CASE 1
d1_1 = -5/27;
d2_1 = -1/2;
v1_1 = [3/5;4/5];
v2_1 = [1;3/5];
A1 = [v1_1,v2_1] * diag([d1_1 d2_1]) * [v1_1,v2_1]^(-1);

d1 = 0.007; % Delay in ms

[KET1,KTT1] = ETandTTControl(A1,d1);

x0_1 = [1;0];
```

```

assignin('base', 'A1', A1);
assignin('base', 'KET1', KET1);
assignin('base', 'KTT1', KTT1);
assignin('base', 'x0_1', x0_1);
%% CASE 2
d1_2 = -1/4;
d2_2 = -2/3;
v1_2 = [2/3;3/5];
v2_2 = [1;3/5];
A2 = [v1_2,v2_2] * diag([d1_2 d2_2]) * [v1_2,v2_2]^(-1);

d2 = 0.017; % Delay in ms

[KET2,KTT2] = ETandTTControl(A2,d2);

x0_2 = [0.2; -0.4];

assignin('base', 'A2', A2);
assignin('base', 'KET2', KET2);
assignin('base', 'KTT2', KTT2);
assignin('base', 'x0_2', x0_2);
%% CASE 3
d1_3 = -2;
d2_3 = -1/2;
v1_3 = [1/2;9/10];
v2_3 = [4/5;3/5];
A3 = [v1_3,v2_3] * diag([d1_3 d2_3]) * [v1_3,v2_3]^(-1);

d3 = 0.012; % Delay in ms

[KET3,KTT3] = ETandTTControl(A3,d3);

x0_3 = [1;0];

assignin('base', 'A3', A3);
assignin('base', 'KET3', KET3);
assignin('base', 'KTT3', KTT3);
assignin('base', 'x0_3', x0_3);
%% CASE 4
d1_4 = -1/5;
d2_4 = -1/2;
v1_4 = [2/3;4/5];
v2_4 = [4/5;3/5];
A4 = [v1_4,v2_4] * diag([d1_4 d2_4]) * [v1_4,v2_4]^(-1);

d4 = 0.012; % Delay in ms

[KET4,KTT4] = ETandTTControl(A4,d4);

x0_4 = [1;-0.1];

assignin('base', 'A4', A4);
assignin('base', 'KET4', KET4);
assignin('base', 'KTT4', KTT4);
assignin('base', 'x0_4', x0_4);
%% CASE 5
d1_5 = -1/4;
d2_5 = -1/2;
v1_5 = [1/2;3/5];
v2_5 = [4/5;3/5];
A5 = [v1_5,v2_5] * diag([d1_5 d2_5]) * [v1_5,v2_5]^(-1);

d5 = 0.017; % Delay in ms

[KET5,KTT5] = ETandTTControl(A5,d5);

x0_5 = [1;-0.1];

assignin('base', 'A5', A5);
assignin('base', 'KET5', KET5);
assignin('base', 'KTT5', KTT5);
assignin('base', 'x0_5', x0_5);
%% CASE 6
A6 = [-5,3.5;-3,2];

d6 = 0.007; % Delay in ms

[KET6,KTT6] = ETandTTControl(A6,d6);

x0_6 = [1;0];

assignin('base', 'A6', A6);
assignin('base', 'KET6', KET6);
assignin('base', 'KTT6', KTT6);
assignin('base', 'x0_6', x0_6);
%% Simulation and Plots of Responses
sim('distributed_w_schedanNM_pretty');
assignin('base', 'tout', tout);
assignin('base', 'y1', y1);
assignin('base', 'y2', y2);
assignin('base', 'y3', y3);
assignin('base', 'y4', y4);
assignin('base', 'y5', y5);
assignin('base', 'y6', y6);
assignin('base', 'nschedule1', nschedule1);

% Response Times
xi_1 = ts( y1.signals(1,1).values, tout );
xi_2 = ts( y2.signals(1,1).values, tout );
xi_3 = ts( y3.signals(1,1).values, tout );
xi_4 = ts( y4.signals(1,1).values, tout );
xi_5 = ts( y5.signals(1,1).values, tout );
xi_6 = ts( y6.signals(1,1).values, tout );

% Color Definitions for Shadings
blue = [97/255,189/255,252/255];  % Static segment
orange = [255/255,128/255,0/255]; % Dynamic segment
```

```

figure(1);
% Shading static and dynamic segements
area([0 xi_1],[1.2 1.2],-0.2,'FaceColor',blue); hold on;
area([xi_1 8],[1.2 1.2],-0.2,'FaceColor',orange);
% Plot response
plot(tout,y1.signals(1,1).values,'LineWidth',2); hold off;
xlabel('time (s)','FontSize',12,'FontWeight','bold');
ylabel('||x||','FontSize',12,'FontWeight','bold');
% Plot lines showing steady-state region
line([0 8],[0.1 0.1],'Color','r','LineStyle','--','LineWidth',2);
line([0 8],[-0.1 -0.1],'Color','r','LineStyle','--','LineWidth',2);
% Set other figure properties
grid on; xlim([0 8]); ylim([-0.2 1.2]);
set(gca,'FontSize',12,'FontWeight','bold','Layer','top');

figure(2);
% Shading static and dynamic segements
area([0 xi_1],[1.2 1.2],-0.2,'FaceColor',orange); hold on;
area([xi_1 xi_2],[1.2 1.2],-0.2,'FaceColor',blue);
area([xi_2 8],[1.2 1.2],-0.2,'FaceColor',orange);
% Plot response
plot(tout,y2.signals(1,1).values,'LineWidth',2); hold off;
xlabel('time (s)','FontSize',12,'FontWeight','bold');
ylabel('||x||','FontSize',12,'FontWeight','bold');
% Plot lines showing steady-state region
line([0 8],[0.1 0.1],'Color','r','LineStyle','--','LineWidth',2);
line([0 8],[-0.1 -0.1],'Color','r','LineStyle','--','LineWidth',2);
% Set other figure properties
grid on; xlim([0 8]); ylim([-0.2 1.2]);
set(gca,'FontSize',12,'FontWeight','bold','Layer','top');

figure(3);
% Shading static and dynamic segements
area([0 xi_3],[1.2 1.2],-0.2,'FaceColor',blue); hold on;
area([xi_3 8],[1.2 1.2],-0.2,'FaceColor',orange);
```

```
% Plot response
plot(tout,y3.signals(1,1).values,'LineWidth',2); hold off;
xlabel('time (s)','FontSize',12,'FontWeight','bold');
ylabel('||x||','FontSize',12,'FontWeight','bold');
% Plot lines showing steady-state region
line([0 8],[0.1 0.1],'Color','r','LineStyle','--','LineWidth',2);
line([0 8],[-0.1 -0.1],'Color','r','LineStyle','--','LineWidth',2);
% Set other figure properties
grid on; xlim([0 8]); ylim([-0.2 1.2]);
set(gca,'FontSize',12,'FontWeight','bold','Layer','top');

figure(4);
% Shading static and dynamic segements
area([0 xi_3],[1.2 1.2],-0.2,'FaceColor',orange); hold on;
area([xi_3 xi_4],[1.2 1.2],-0.2,'FaceColor',blue);
area([xi_4 10],[1.2 1.2],-0.2,'FaceColor',orange);
% Plot response
plot(tout,y4.signals(1,1).values,'LineWidth',2); hold off;
xlabel('time (s)','FontSize',12,'FontWeight','bold');
ylabel('||x||','FontSize',12,'FontWeight','bold');
% Plot lines showing steady-state region
line([0 10],[0.1 0.1],'Color','r','LineStyle','--','LineWidth',2);
line([0 10],[-0.1 -0.1],'Color','r','LineStyle','--','LineWidth',2);
% Set other figure properties
grid on; xlim([0 10]); ylim([-0.2 1.2]);
set(gca,'FontSize',12,'FontWeight','bold','Layer','top');

figure(5);
% Shading static and dynamic segements
area([0 xi_5],[1.2 1.2],-0.2,'FaceColor',blue); hold on;
area([xi_5 8],[1.2 1.2],-0.2,'FaceColor',orange);
% Plot response
plot(tout,y5.signals(1,1).values,'LineWidth',2); hold off;
xlabel('time (s)','FontSize',12,'FontWeight','bold');
ylabel('||x||','FontSize',12,'FontWeight','bold');
% Plot lines showing steady-state region
```

```
line([0 8],[0.1 0.1],'Color','r','LineStyle','--','LineWidth',2);
line([0 8],[-0.1 -0.1],'Color','r','LineStyle','--','LineWidth',2);
% Set other figure properties
grid on; xlim([0 8]); ylim([-0.2 1.2]);
set(gca,'FontSize',12,'FontWeight','bold','Layer','top');

figure(6);
% Shading static and dynamic segements
area([0 xi_5],[1.2 1.2],-0.2,'FaceColor',orange); hold on;
area([xi_5 xi_6],[1.2 1.2],-0.2,'FaceColor',blue);
area([xi_6 12],[1.2 1.2],-0.2,'FaceColor',orange);
% Plot response
plot(tout,y6.signals(1,1).values,'LineWidth',2); hold off;
xlabel('time (s)','FontSize',12,'FontWeight','bold');
ylabel('||x||','FontSize',12,'FontWeight','bold');
% Plot lines showing steady-state region
line([0 12],[0.1 0.1],'Color','r','LineStyle','--','LineWidth',2);
line([0 12],[-0.1 -0.1],'Color','r','LineStyle','--','LineWidth',2);
% Set other figure properties
grid on; xlim([0 12]); ylim([-0.2 1.2]);
set(gca,'FontSize',12,'FontWeight','bold','Layer','top');

figure(7)
subplot(2,1,2);
plot(tout,nschedule1.signals.values(:,1)); hold all;
plot(tout,nschedule1.signals.values(:,2));
plot(tout,nschedule1.signals.values(:,3)); hold off
grid on; xlim([0 8]);
xlabel('time (s)','FontSize',12,'FontWeight','bold');
ylabel('TT slots','FontSize',12,'FontWeight','bold');
set(gca,'FontSize',12,'FontWeight','bold');
subplot(2,1,1);
plot(tout,nschedule1.signals.values(:,4)); hold all;
plot(tout,nschedule1.signals.values(:,5));
plot(tout,nschedule1.signals.values(:,6));
plot(tout,nschedule1.signals.values(:,7));
plot(tout,nschedule1.signals.values(:,8));
plot(tout,nschedule1.signals.values(:,9)); hold off
grid on; xlim([0 8]);
ylabel('ET slots','FontSize',12,'FontWeight','bold');
set(gca,'FontSize',12,'FontWeight','bold');

function [KET,KTT] = ETandTTControl(a,delay)
% TT and ET systems using LQR
global b sampleTime

% Continuous system model
A = a;
B = b;

% Time Parameters
d = delay; % Delay in ms
h = sampleTime;
%% ET response
% Form equivalent discrete model
syms s;
e_As = expm(A*s);
Phi = subs(e_As,s,h);
Gamma_1 = double(subs(e_As,s,h-d)*int(e_As*B,'s',0,d));
Gamma_0 = double(int(e_As*B,'s',0,h-d));
AdET = [Phi Gamma_1
     zeros(1,2) 0];
BdET = [Gamma_0
     1];
sysET = ss(AdET,BdET,zeros(1,3),0,h);

% Compute LQR gain K
QET = [eye(2) zeros(2,1)
     zeros(1,2) 10];
R = 0.01;
KET = lqr(sysET,QET,R);
%% TT responses
```

```
% Form equivalent discrete model
AdTT = Phi;
BdTT = double(int(e_As*B,'s',0,h));
sysTT = ss(AdTT,BdTT,zeros(1,2),0,h);

% Compute LQR gain K
QTT = eye(2);
KTT = lqr(sysTT,QTT,R);
```

src/distributed_w_schedan_init.m

Appendix B

Slot Scheduling C++ Program

B.1 Number_of_slots.cpp

```
#include <iostream>
#include <iomanip>
#include <list>
#include <vector>
#include "C_app.h"
#include "Slot.h"
using namespace std;

int Number_of_slots(list<C_app> &Ci);
void print(vector<Slot> &slotArray);

int main()
{
    // For TrueTime Simulation
    // Non-Monotonic
    C_app *C1 = new C_app(1,200,9   ,1.6809  ,11.6243,5.3027  ,2.2675  );
    C_app *C2 = new C_app(2,20  ,6.25,2.578   ,8.5865 ,2.9487  ,1.342   );
    C_app *C3 = new C_app(3,15  ,2   ,0.38562,3.9724 ,0.64081,0.68966);
    C_app *C4 = new C_app(4,200,7.5 ,2.495   ,10.3982,4.0258  ,1.9215  );
    C_app *C5 = new C_app(5,20  ,8.5 ,2.7534  ,10.633 ,4.577   ,1.9714  );
    C_app *C6 = new C_app(6,6   ,6   ,0.71207,7.94   ,0.92249,0.66886);

    list<C_app> Sorted_Capps;

    // For TrueTime Simulation
    Sorted_Capps.push_back(*C3);
    Sorted_Capps.push_back(*C6);
    Sorted_Capps.push_back(*C2);
    Sorted_Capps.push_back(*C4);
    Sorted_Capps.push_back(*C5);
    Sorted_Capps.push_back(*C1);

    cout << "Number of Slots: " << Number_of_slots(Sorted_Capps) << endl
        ;
    return 0;
}

int Number_of_slots(list<C_app> &Ci)
{
    list<C_app>::iterator it;
    int number_slots = 1;
    Slot *first = new Slot();
    vector<Slot> slotArray;
    slotArray.push_back(*first);
    for(it=Ci.begin();it!=Ci.end();it++)
    {
        cout << "Scheduling C" << it->getID() << endl;
        for(int s=0;s<slotArray.size();s++)
        {
            cout << "Trying Slot " << s+1 << endl;
            if(slotArray[s].Scheduable(*it))
            {
                slotArray[s].insertCapp(*it);
                break;
            }
            else
            {
                    if(s==slotArray.size()-1)
                    {
                        number_slots = number_slots + 1;
                        cout << "NEW SLOT #" << number_slots << endl;
                        Slot *next = new Slot();
                        slotArray.push_back(*next);
                        it->setXI(it->getXI_TT());
                        slotArray[s+1].insertCapp(*it);
                        break;
                    }
                }
            }
        }
    print(slotArray);
//      return number_slots;
    return slotArray.size();
}

void print(vector<Slot> &slotArray)
{
    cout << "$C_i$ & $r_i$ & $\\xi^d_i$ & $\\xi^{TT}_i$ & $\\xi^{ET}_i$
       & $\\xi^m_i$ & $t_{p,i}$ & $\\hat{\\xi}_i$ & MA $\\hat{\\xi}_i$
       \\\\ \\hline" << endl;
    cout << left;
    for(int i=0;i<slotArray.size();i++)
    {
        for(int j=0;j<slotArray[i].getSize();j++)
        {
            cout << setw(1) << slotArray[i].Sch_Capps[j].getID() << " &
                " << setw(4) << slotArray[i].Sch_Capps[j].getR() << " &
                " <<
                setw(4) << slotArray[i].Sch_Capps[j].getXI_D() << "
                    & " << setw(7) << slotArray[i].Sch_Capps[j].
                    getXI_TT() << " & " <<
                setw(7) << slotArray[i].Sch_Capps[j].getXI_ET() << "
                    & " << setw(7) << slotArray[i].Sch_Capps[j].
```

```
                                getXI_M() << " & " <<
                                setw(7) << slotArray[i].Sch_Capps[j].getT_P() << " &
                                    " << setw(7) << slotArray[i].Sch_Capps[j].getXI
                                    () << " \\\\ \\hline" << endl;
            }
        }
        cout << endl;
        for(int i=0;i<slotArray.size();i++)
        {
            cout << "| ";
            for(int j=0;j<slotArray[i].getSize();j++)
            {
                cout << slotArray[i].Sch_Capps[j].getID() << " ";
            }
        }
        cout << "|" << endl;
}
```

src/Number_of_slots.cpp

B.2 C_app.h

```
#ifndef CAPP_H
#define CAPP_H

#include <iostream>
using namespace std;

class C_app
{
    protected:
        int id;
        double r;
        double Xi_d;
        double Xi_TT;
        double Xi_ET;
        double Xi_m;
        double t_p;
        double Alpha;
        double Beta;
        double Xi_new;
        double Xi;
    public:
        // Constructor
      C_app(int ID, double R, double XI_D, double XI_TT, double XI_ET,
          double XI_M, double T_P);
      // Get Methods
        int getID() {return id;}
      double getR() {return r;}
      double getXI_D() {return Xi_d;}
      double getXI_TT() {return Xi_TT;}
        double getXI_ET() {return Xi_ET;}
        double getXI_M() {return Xi_m;}
        double getT_P() {return t_p;}
        double getALPHA() {return Alpha;}
      double getBETA() {return Beta;}
```

```
        double getXI_NEW() {return Xi_new;}
        double getXI() {return Xi;}
      // Set Methods
      void setXI_NEW(double XI_NEW) {Xi_new = XI_NEW;}
      void setXI(double XI) {Xi = XI;}
};

C_app::C_app(int ID, double R, double XI_D, double XI_TT, double XI_ET,
    double XI_M, double T_P)
{
    id = ID;
    r = R;
    Xi_d = XI_D;
    Xi_TT = XI_TT;
    Xi_ET = XI_ET;
    Xi_m = XI_M;
    t_p = T_P;
    Alpha = (Xi_m - Xi_TT) / t_p;
    Beta = Xi_m / (Xi_ET - t_p);
    Xi_new = 0;
    Xi = 0;
}
#endif
```

src/C_app.h

B.3 Slot.h

```
#ifndef SLOT_H
#define SLOT_H

#include <iostream>
#include <vector>
#include "C_app.h"
#include <cmath>
using namespace std;

class Slot
{
    protected:
        vector<C_app> Sch_Capps;
    public:
        // Constructor
        Slot() {Sch_Capps.clear();}
        // Get Method
        int getSize() {return Sch_Capps.size();}
        // Other Methods
        void insertCapp(C_app &C);// {Sch_Capps.push_back(C);}
        double maxb(int minIndex, C_app &C);
        bool Scheduable(C_app &C);
        friend void print(vector<Slot> &slotArray);
};

void Slot::insertCapp(C_app &C)
{
    cout << "INSERTED C" << C.getID() << "!!!" << endl;
    cout << endl;
    Sch_Capps.push_back(C);
}

double Slot::maxb(int minIndex, C_app &C)
{
    double bMin = 0;
    for(int i=minIndex;i<Sch_Capps.size();i++)
    {
        if(bMin<Sch_Capps[i].getXI_M()) bMin = Sch_Capps[i].getXI_M();
    }
    if(bMin<C.getXI_M()) bMin = C.getXI_M();
    return bMin;
}

bool Slot::Scheduable(C_app &C)
{
    bool TF;
    if(Sch_Capps.empty())
    {
        C.setXI(C.getXI_TT());
        TF = true;
    }
    else
    {
        cout << "Already " << Sch_Capps.size() << " here" << endl;
        for(int i=0;i<Sch_Capps.size();i++)
        {
            cout << "LOOP #" << i << endl;
            double b = 0;
            if(i==Sch_Capps.size()-1) b = C.getXI_M();
            else b = maxb(i+1,C);
            cout << "maxb = " << b << endl;
            double Xi_old = 0;
            double Xi = 0;
            if(b<Sch_Capps[i].getT_P()) Xi = Sch_Capps[i].getXI_TT() +
                (1 + Sch_Capps[i].getALPHA()) * b;
            else Xi = Sch_Capps[i].getBETA() * Sch_Capps[i].getXI_ET() +
                 (1 - Sch_Capps[i].getBETA()) * b;
            cout << "Xi = " << Xi << endl;
            while(Xi<=Sch_Capps[i].getXI_D() && Xi!=Xi_old)
            {
                double sum = 0;
                Xi_old = Xi;
                for(int j=0;j<i;j++)
                {
                    sum = sum + ceil(Xi_old/Sch_Capps[j].getR()) *
                        Sch_Capps[j].getXI_M();
                }
                if((b+sum)<Sch_Capps[i].getT_P())
                {
                    Xi = Sch_Capps[i].getXI_TT() + (1 + Sch_Capps[i].
                        getALPHA()) * b + (1 + Sch_Capps[i].getALPHA())
                        * sum;
                }
                else
                {
                    Xi = Sch_Capps[i].getBETA() * Sch_Capps[i].getXI_ET
                        () + (1 - Sch_Capps[i].getBETA()) * b + (1 -
                        Sch_Capps[i].getBETA()) * sum;
                }
            }
            cout << "Slot Loop " << i << ": Xi     =" << Xi << endl;
            cout << "                 Xi_old=" << Xi_old << endl;
            if(Xi==Xi_old) Sch_Capps[i].setXI_NEW(Xi);
            else
            {
                cout << "Slot Loop " << i << " exits early" << endl;
                TF = false;
                return TF;
            }
        }
        double Xi_old = 0;
        double Xi = 0;
        Xi = C.getXI_TT();
        while(Xi<=C.getXI_D() && Xi!=Xi_old)
        {
            double sum = 0;
            Xi_old = Xi;
            for(int j=0;j<Sch_Capps.size();j++)
            {
                sum = sum + ceil(Xi_old/Sch_Capps[j].getR()) * Sch_Capps
                    [j].getXI_M();
            }
            if(sum<C.getT_P()) Xi = C.getXI_TT() + (1 + C.getALPHA()) *
                sum;
            else Xi = C.getBETA() * C.getXI_ET() + (1 - C.getBETA()) *
                sum;
        }
        cout << "Sched Contl : Xi     =" << Xi << endl;
        cout << "                     Xi_old=" << Xi_old << endl;
        if(Xi==Xi_old)
        {
            C.setXI(Xi);
            for(int i=0;i<Sch_Capps.size();i++)
            {
                Sch_Capps[i].setXI(Sch_Capps[i].getXI_NEW());
            }
            TF = true;
        }
        else TF = false;
    }
    cout << "Scheduable at end? " << TF << endl;
    return TF;
}
#endif
```

src/Slot.h

Bibliography

[1] *The FlexRay Communications System Specifications*, December 2005. Ver. 2.1.

[2] A. Annaswamy, D. Soudbakhsh, R. Schneider, D. Goswami, and S. Chakraborty. Arbitrated network control systems: A co-design of control and platform for cyber-physical systems. In *the Workshop on the Control of Cyber-Physical Systems*, Baltimore, MD, 2013.

[3] Karl J. Åström and Björn Wittenmark. *Computer-Controlled Systems: Theory and Design*. Prentice Hall, 1997.

[4] Sanjoy Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[5] X. Cao, P. Cheng, J. Chen, and Y. Sun. An Online Optimization Approach for Control and Communication Co-Design in Networked Cyber-Physical Systems. *IEEE Transactions on Industrial Informatics*, 2012.

[6] M. Cea Garrido and G. Goodwin. Stabilization of systems over bit rate constrained networked control architecture. *IEEE Transactions on Industrial Informatics*, 2012.

[7] L. Feng-Li, J.K. Yook, D.M. Tilbury, and J. Moyne. Network architecture and communication modules for guaranteeing acceptable control and communication performance for networked multi-agent systems. *IEEE Transactions on Industrial Informatics*, 2(3), 2006.

[8] Dip Goswami, Martin Lukasiewycz, Reinhard Schneider, and Samarjit Chakraborty. Time-triggered implementations of mixed-criticality automotive software. In *Proceedings of the 15th Conference for Design, Automation and Test in Europe (DATE)*, Dresden, Germany, 2012.

[9] Dip Goswami, Reinhard Schneider, and Samarjit Chakraborty. Re-engineering cyber-physical control applications for hybrid communication protocols. In *DATE*, pages 914–919, 2011.

[10] C. Huang, Y. Bai, and X. Liu. H-Infinity State Feedback Control for a Class of Networked Cascade Control Systems With Uncertain Delay. *IEEE Transactions on Industrial Informatics*, 6(1):62–72, 2010.

[11] *Time-Triggered Communication*, 2000. Part 4.

[12] A. Jestratjew and A. Kwiecien. Performance of HTTP Protocol in Networked Control Systems. *IEEE Transactions on Industrial Informatics*, 2012.

[13] Martin Lukasiewycz, Michael Glaß, Jürgen Teich, and Paul Milbredt. Flexray schedule optimization of the static segment. In *CODES+ISSS*, pages 363–372, 2009.

[14] P. Marti, A. Camacho, M. Velasco, and M. E. M. Ben Gaid. Runtime allocation of optional control jobs to a set of can-based networked control systems. *IEEE Transactions on Industrial Informatics*, 2(4), 2010.

[15] Adolfo Martinez and Paulo Tabuada. On the benefits of relaxing the periodicity assumption for networked control systems over CAN. In *RTSS*, pages 3–12, 2009.

[16] Alejandro Masrur, Dip Goswami, Samarjit Chakraborty, Jian-Jia Chen, Anuradha Annaswamy, and Ansuman Banerjee. Timing analysis of cyber-physical applications for hybrid communication protocols. In *DATE*, pages 1233–1238, March 2012.

[17] Alejandro Masrur, Dip Goswami, Reinhard Schneider, Harald Voit, Anuradha Annaswamy, and Samarjit Chakraborty. Schedulability analysis of distributed cyber-physical applications on mixed time-/event-triggered bus architectures with retransmissions. In *SIES*, pages 266–273, June 2011.

[18] K.S. Narendra and A.M. Annaswamy. *Stable Adaptive Systems*. Dover Books on Electrical Engineering Series. Dover, 2005.

[19] Paul Pop, Petru Eles, and Zebo Peng. Schedulability-driven communication synthesis for time triggered embedded systems. *Real-Time Systems*, 26(3):297–325, 2004.

[20] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the FlexRay communication protocol. *Real-Time Systems*, 39:205–235, 2008.

[21] Traian Pop, Petru Eles, and Zebo Peng. Design optimization of mixed time/event-triggered distributed embedded systems. In *CODES+ISSS*, pages 83–89, 2003.

[22] George C. Reis. Some properties of indefinite matrices related to control theory. *IEEE Transactions on Automatic Control*, 12(6):789–790, December 1967.

[23] Ken Tindell, Alan Burns, and Andy J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[24] H. Zeng, M. D. Natale, A. Ghosal, and A. Sangiovanni-Vincentelli. Schedule optimization of time-triggered systems communicating over the FlexRay Static Segment. *IEEE Transactions on Industrial Informatics*, 7(1), 2011.

[25] Fumin Zhang, Klementyna Szwaykowska, Wayne Wolf, and Vincent John Mooney III. Task scheduling for control oriented requirements for cyber-physical systems. In *RTSS*, pages 47–56, 2008.