EL2450: Hybrid and Embedded Control Systems: Homework 2

[To be handed in **February 18**]

Introduction

This homework consists of three parts, the first part is about scheduling, the second part is about networked control systems and in the third part we will be studying concepts of Product Transition Systems. In the first part, control performance for three control tasks executed on the same CPU will be evaluated for different scheduling algorithms. In the second part, networked control systems with delay and packet losses will be modeled and analyzed. In the third part, different definitions of Product Transition Systems are implemented.

Part One: Scheduling [25p]

This part of the homework is on analysis, simulation and implementation of a real time control system. The main focus is on scheduling of the processor executing the control algorithms. The processes that should be controlled are three inverted pendulums and the control algorithms are all running on the same processor. To be able to simulate such a system, a Simulink toolbox called Truetime is used.

Exercises

Download the Matlab files package from the homepage. The processes to be controlled are illustrated in Figure 1.

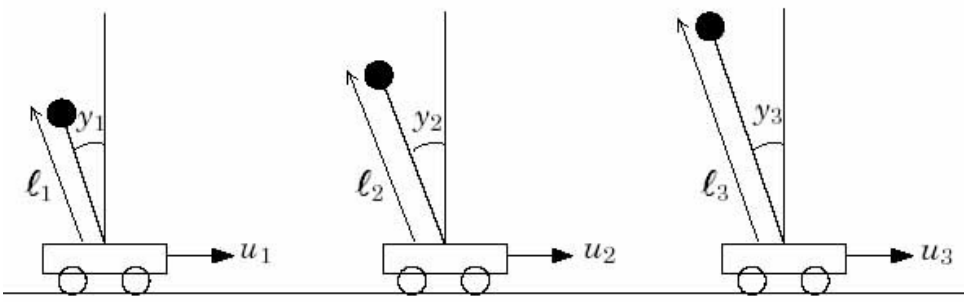


Figure 1: Three pendulums.

A linearized model of a pendulum is given by the transfer function:

$$G_i(s) = \frac{\omega_i^2}{s^2 - \omega_i^2} \ , \quad i = 1, 2, 3$$

where ω_i is the natural frequency of the pendulum. In this example the pendulums are of different length, $\ell_i = \{0.1, 0.2, 0.3\}$ m, which corresponds to the frequencies $\omega_i = \{9.9, 7.0, 5.7\}$ rad/s. Each pendulum is controlled by a controller derived with discrete LQG. Because of the different lengths the sampling times for the controllers are not equal. The sampling times T_i are chosen to be 20, 29 and 35 ms (a shorter pendulum is harder to stabilize and needs a shorter sampling period). The discrete controller for the shortest pendulum is:

$$C(z) = \frac{-9.243z^2 + 7.278z}{z^2 - 0.3768z + 0.1392}.$$

Rate Monotonic scheduling

[Task 1][1p] Explain what Rate Monotonic scheduling means.

[Task 2][2p] Assume the execution time for all three tasks are equally $6ms$. Are the tasks schedulable with this policy? Please motive your answer and construct part of the corresponding schedule manually.

[Task 3][2p] Install Truetime and Jitterbug by following the Appendix. Once it is done, open the simulink model by typing

```
> inv_pend_three
```

Simulate the system. Plot and save the pendulum angles. Are the pendulums stabilized? What are the differences in control performance between the different pendulums?

[Task 4][2p] Compare the schedule plot with yours. Does the result agree with the analysis, *i.e.*, does the task get time to execute before their sampling period is over?

[Task 5][3p] Now assume the execution time for all three tasks are equally $10ms$. Then resolve Question 2, 3 and 4, with the new execution time. What are the differences with respect to the control performance and the schedule? (*Hint*: Open `init_pend_three.m` and change the execution time to $10ms$.)

Earliest Deadline First (EDF)

Next we will examine how EDF scheduling affects the performance of the system.

[Task 6][1p] Explain how Earliest Deadline First scheduling works. What are the advantages and the disadvantages of using the EDF compared to RM?

[Task 7][2p] Assume the execution time is changed back to $6ms$. Are the tasks schedulable with the EDF? Please motive your answer and construct part of the corresponding schedule manually.

[Task 8][2p] Open `init_pend_three.m` and change the scheduling policy to 'prioEDF'. Save the script. Rerun `inv_pend_three.mdl`. Plot and save the angles of three pendulums. Are the pendulums stabilized? What are the differences in control performance between different pendulums?

[Task 9][2p] Compare the schedule plot with yours. Does the result agree with the analysis, *i.e.*, does the task get time to execute before its sampling periods is over?

[Task 10][4p] Resolve Question 7, 8 and 9 with the new execution time $T = 10ms$. What are the differences with respect to the control performance and the schedule?

[Task 11][4p] Do the controllers perform better than under RM-scheduling? Please motive your answer using the simulation results.

Part Two: Networked Control Systems [10p]

In the second part of the homework, you will analyze networked control systems (NCS).

Stability of NCS with Network-induced Delay

Consider the Networked Control Systems in Figure 2. The system consists of a continuous plant

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t)$$

and a discrete controller

$$u(kh) = -Lx(kh), \qquad k = 0, 1, 2, \ldots$$

where $A \in \mathbb{R}$, $B \in \mathbb{R}$, $C \in \mathbb{R}$.

Consider a simple integrator model, *i.e.*, $A = 0$, $B = 1$. Further assume that the delay is less than one sampling period, *i.e.*, $\tau = \tau_{sc} + \tau_{ca} \leq h$.

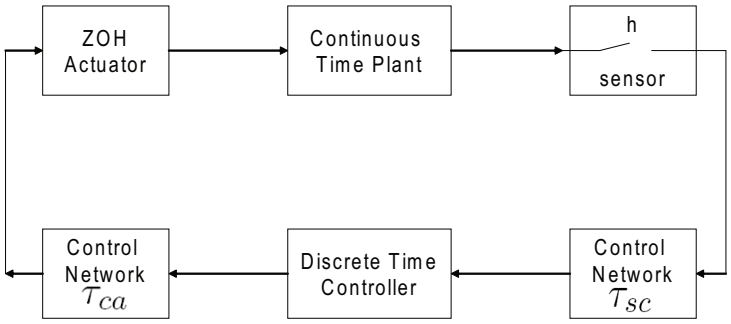


Figure 2: Networked Control System with communication delay.

[Task 12][2p] Calculate analytically the closed-loop equations of the system.

[Task 13][4p] Calculate the characteristic polynomial of the system and find a condition of the form:

$$f(L,h) < \frac{\tau}{h} < g(L,h),$$

which guarantees the stability of the system. (Hint: Use stability criteria for 2nd order polynomials in discrete-time).

Now we will simulate an example of NCS with network-induced delay to show how it affects the control performance. Make sure you have the file `inv_pend_delay.mdl` and open the simulink model by typing

```
> inv_pend_delay
```

[Task 14][4p] Simulate the system. Check by simulations that for which value of the communication delay (in the *Transport Delay* block) the system becomes unstable. Please support your answer by simulation results.

Part Three: Product Transition Systems [15p]

As it has been presented in the lectures, Transition Systems form a compact and convenient way of modeling the behavior of a system. A Transition System $\mathcal{T}$ is a tuple $(S, S^0, \Sigma, \longrightarrow, AP, L)$ where:

- S is a finite set of states,
- $S^0 \subseteq S$ is a set of initial states,
- Σ is a set of actions,
- $\longrightarrow \subseteq S \times \Sigma \times S$ is a transition relation ($\times$ is the Cartesian product operator),
- AP is a finite set of atomic propositions, and
- $L : S \to 2^{AP}$ is a labeling function.

In many research fields such as computer science, control theory and robotics, we usually have dynamical systems that can be modeled as Transition Systems and they can interact with each other. In order to model the interaction, we use Product Transition Systems. Two of the most popular Product Transition Systems techniques are:

1. Product Interleaving Transition System
2. Product Handshaking Transition System

We define hereafter formally the aforementioned terms. Given two transition systems $\mathcal{T}_i = (S_i, S_i^0, \Sigma_i, \longrightarrow_i, AP_i, L_i), i \in \{1, 2\}$ we define the Product Interleaving Transition System

$$\mathcal{T}_{\text{int}} = (S_{\text{int}}, S_{\text{int}}^0, \Sigma_{\text{int}}, \longrightarrow_{\text{int}}, AP_{\text{int}}, L_{\text{int}}),$$

where:

- $S_{\text{int}} = S_1 \times S_2$ is a set of states,
- $S_{\text{int}}^0 = S_1^0 \times S_2^0 \subseteq S_{\text{int}}$ is a set of initial states,
- $\Sigma_{\text{int}} = \Sigma_1 \cup \Sigma_2$ is a set of actions,
- $\longrightarrow_{\text{int}} \subseteq S_{\text{int}} \times \Sigma_{\text{int}} \times S_{\text{int}}$ is a transition relation and is defined according to the following rules:
  - If $(s_1, \sigma, s_1') \in \longrightarrow_1$ then $((s_1, s_2), \sigma, (s_1', s_2)) \in \longrightarrow_{\text{int}}$.
  - If $(s_2, \sigma, s_2') \in \longrightarrow_2$ then $((s_1, s_2), \sigma, (s_1, s_2')) \in \longrightarrow_{\text{int}}$.
- $AP_{\text{int}} = AP_1 \cup AP_2$ is a finite set of atomic propositions, and
- L_{int} is a labeling function which is defined as: $L_{\text{int}}((s, s')) = L_1(s) \cup L_2(s'), \forall (s, s') \in S_{\text{int}}$.

Given two transition systems $\mathcal{T}_i = (S_i, S_i^0, \Sigma_i, \longrightarrow_i, AP_i, L_i), i \in \{1, 2\}$ and the set $H \subseteq \Sigma_1 \cap \Sigma_2$, we define the Product Handshaking Transition System

$$\mathcal{T}_{\text{hand}} = (S_{\text{hand}}, S_{\text{hand}}^0, \Sigma_{\text{hand}}, \longrightarrow_{\text{hand}}, AP_{\text{hand}}, L_{\text{hand}}),$$

where:

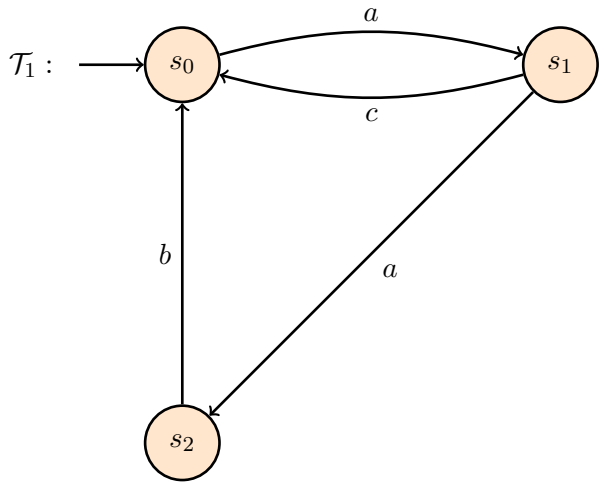
- $S_{\text{hand}} = S_1 \times S_2$ is a set of states,
- $S_{\text{hand}}^0 = S_1^0 \times S_2^0 \subseteq S_{\text{int}}$ is a set of initial states,
- $\Sigma_{\text{hand}} = \Sigma_1 \cup \Sigma_2$ is a set of actions,
- $\longrightarrow_{\text{hand}} \subseteq S_{\text{hand}} \times \Sigma_{\text{hand}} \times S_{\text{hand}}$ is a transition relation and is defined according to the following rules:
  - ☐ For an action $\sigma \notin H$ we have:
    - If $(s_1, \sigma, s_1') \in \longrightarrow_1$ then $((s_1, s_2), \sigma, (s_1', s_2)) \in \longrightarrow_{\text{hand}}$.
    - If $(s_2, \sigma, s_2') \in \longrightarrow_2$ then $((s_1, s_2), \sigma, (s_1, s_2')) \in \longrightarrow_{\text{hand}}$.
  - ☐ For an action $\sigma \in H$ we have:
    - If $(s_1, \sigma, s_1') \in \longrightarrow_1$ and $(s_2, \sigma, s_2') \in \longrightarrow_2$ then $((s_1, s_2), \sigma, (s_1', s_2')) \in \longrightarrow_{\text{hand}}$.
- $AP_{\text{hand}} = AP_1 \cup AP_2$ is a finite set of atomic propositions, and
- L_{hand} is a labeling function which is defined as: $L_{\text{hand}}((s, s')) = L_1(s) \cup L_2(s'), \forall (s, s') \in S_{\text{hand}}$.

Consider two transition systems $\mathcal{T}_i$ as depicted in Fig. 3, where $AP_i = S_i$, and L_i is the identity map ($L_i(x) = \{x\}$), $i \in \{1, 2\}$. Answer the following questions:

[Task 15][2p] Define the Transitions Systems $\mathcal{T}_1, \mathcal{T}_2$ formally (write down all the sets of the tuple).

[Task 16][6p] Define (write down all the sets of the tuple) and draw the Product Interleaving Transition System $\mathcal{T}_{\text{int}}$.

[Task 17][7p] Define (write down all the sets of the tuple) and draw the Product Handshaking Transition System $\mathcal{T}_{\text{hand}}$ for $H = \{c\}$.



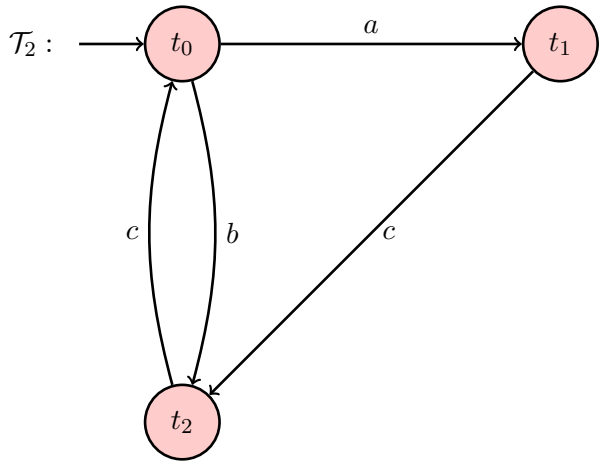


Figure 3: The Transition Systems $\mathcal{T}_1, \mathcal{T}_2$.

Appendix: Installation of Truetime and Jitterbug

To be able to simulate the system in Part one, a Simulink toolbox called Truetime is used. In Truetime, the control algorithms are executed on a simulated real-time operating system. By doing this, the performance of the controller with respect to latency in the system can be evaluated.

Note: If you have a 64bit system, simply follow the corresponding setup instructions further below to install Truetime and Jitterbug together, (i.e, you can skip the following two subsections)

Note: In order to see the plots in the Simulink scopes, you may need to enter the scope block parameters and set the time offset and axis scaling parameters to automatic.

Truetime Installation

32-bit Windows OS and Matlab 2011 (or higher) are recommended for this homework. Both softwares are free of charge from KTH program distributions like `MSDN` and `KTH ProgDist`. Otherwise please follow Section 2 of this Truetime Manual carefully before you continue.

Follow the instructions below to install Truetime within Matlab:

1. Unzip Truetime from the file `truetime-2.0.zip` and save it to any folder such as "`C:\TEMP\truetime-2.0`".
2. Execute the following commands at the Matlab command prompt:
   ```
   >setenv('TTKERNEL','C:\TEMP\truetime-2.0\kernel')
   >addpath([getenv('TTKERNEL')])
   >init_truetime
   >truetime
   ```
3. After executing the last line, check that you get the same window as in Figure 4.

Jitterbug installation

In order to run this simulation, you also need to install Jitterbug.

1. Unzip Jitterbug from the file `jitterbug-1.23.zip` and save it to any folder such as "`C:\TEMP\jitterbug-1.23`".
2. Add path (with subfolders) to the matlab path.
   ```
   >addpath(genpath('C:\TEMP\jitterbug-1.23'))
   ```

Now, you should be able to simulate the system in Part one by yourself!

Windows 64bit Installation

1. Download the Homework 2 Source (Windows64bit) from the course website.
2. Extract it.
3. Open Matlab and change directory to `HW2_sources_Windows64bit/truetime` (`HW2_sources_Windows64bit` is the one just extracted).
4. Run `init_truetime`.
5. Change directory back to `HW2_sources_Windows64bit`.

6. Run `addpath(genpath('.\jitterbug'))` (or add the folder jitterbug to the path manually).
7. Run `inv_pend_three`.
8. Run the Simulink Simulation (ctrl+T).

Ubuntu 64bit Installation

1. Download the Homework 2 Source (Ubuntu64bit) from the course website.
2. Extract it.
3. Open Matlab and change directory to `HW2_sources_Ubuntu64bit/truetime` (`HW2_sources_Ubuntu64bit` is the one just extracted).
4. Run `init_truetime`.
5. Change directory back to `HW2_sources_Ubuntu64bit`.
6. Run `addpath(genpath('.\jitterbug'))` (or add the folder jitterbug to the path manually).
7. Run `inv_pend_three`.
8. Run the Simulink Simulation.

MAC OS Installation

The same with **Ubuntu 64bit Installation**.

Truetime Introduction

TrueTime is a Matlab/Simulink-based simulator for real-time control systems. It consists of seven blocks:

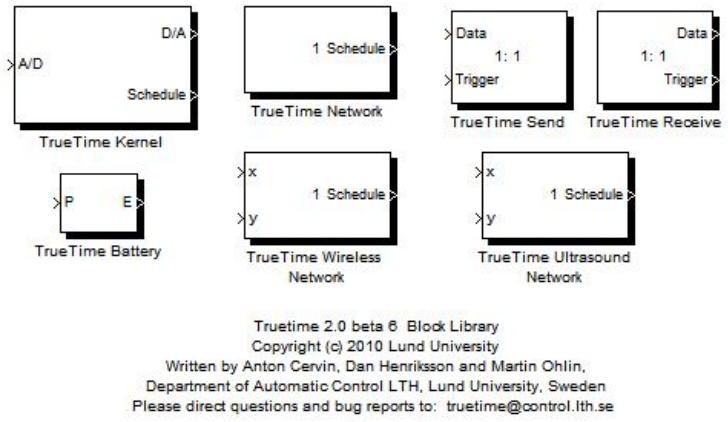


Figure 4: Truetime's Simulink blocks.

The first one, the one that this homework focuses on, is a computer block that simulates a computer running a real time operating system. The others are network blocks for simulation of networked systems, and will not be used in this homework.

The computer block executes user-defined task, i.e. control algorithms, and it is able to simulate the system using arbitrary scheduling policies. In Figure 4 the computer block (`TrueTime kernel`) is shown.

The ports used throughout this homework are the A/D, D/A and the schedule port. The input to the computer is connected to the A/D port of the block and the output to the D/A port. The schedule port shows how the different tasks are scheduled in the processor throughout the simulation.

The initialization script (`init_pend_three.m`) sets up the scheduling algorithm and creates the controller tasks. The data structure `data` contains measurement and control signals and values for the control algorithm. Open `init_pend_three.m` and try to understand the basics of the code. The scheduling policy and execution time can be modified there. Remember to save the script once you have changed something.

(*Note*: Deadline-monotonic scheduling (`'prioDM'`) is used in the code, which is the same as the Rate-monotonic scheduling in this example. Earliest Deadline First scheduling can be chosen by setting the `schedulingPolicy` to `'prioEDF'`).

The function (`pend_reg_three_code.m`) contains the code executed by different tasks. A code function is built up by a number of code segments executed in order by the kernel. Each segment has an execution time and next segment will not start until the time associated with the previous segment has elapsed in the simulation. Open `pend_reg_three_code.m` and try to understand the basics of the code.

Responsible TAs:

- **Part One: Fei Chen (fchen@kth.se))**
- **Parts Two and Three: Peter Varnai (varnai@kth.se)**