Lab Tutorial and Exercise 4

Real-Time Scheduling of Independent Periodic Tasks

In the last lab, we have learned how to use TrueTime to simulate control implementation on microprocessor and RTOS. We have seen the example of two tasks sharing the CPU with the fixed priority (FP) schedule. During the lecture, we have introduced several scheduling strategies for multiple tasks executing on one processor. This lab gives you the practice of choosing the right one for the satisfactory control implementation.

The practice is based on the TrueTime simulator, in which the TrueTime kernel block provides three built-in scheduling strategies that we discussed in the lecture. In this lab, we focus on the *preemptive* scheduling of *periodic* tasks. Other types of scheduling problems can also be modeled and simulated with TrueTime with extra efforts, but they are not discussed in this lab. If interested, you may develop other scheduling methods by referring to the TrueTime Reference Manual. You are welcome to discuss your model with the teaching team.

The objectives of this lab tutorial are:

- to implement fixed priority (FP), rate monotonic (RM), and earliest-deadline first (EDF) scheduling strategies for multiple preemptive periodic tasks executing on one processor,
- to choose the right scheduling strategy for satisfactory control implementation,
- to catch the deadline overrun interrupt and identify the delayed task.

The control system considered in this tutorial is a digital PID controller for a DC motor, whose transfer function is

$$G(s) = \frac{1000}{s(s+1)}$$

The PID controller is implemented according to the following equations[1, 2].

$$u(k) = P(k) + I(k) + D(k)$$

where

[1] TrueTime Reference Manual, pp. 28

[2] B. Wittenmark, K.J. Åström, and K.E. Årzén, *Computer Control: An Overview*, IFAC Professional Brief, 2002, pp.45-46, http://www.ifac-control.org/publications/list-of-professional-briefs/pb_wittenmark_etal_final.pdf/view

$$P(k) = K\big(\beta r(k) - y(k)\big)$$

$$I(k+1) = I(k) + \frac{Kh}{T_i}\big(r(k) - y(k)\big)$$

$$D(k) = \frac{T_d}{T_d + Nh} D(k-1) + \frac{KT_dN}{T_d + Nh}(y(k-1) - y(k))$$

In the equations, u is the control command to the DC motor, r the reference position for the DC motor, and y the position measurement of the motor. Furthermore, the parameter K is the proportional gain, T_i the integration time, T_d the derivative time, β a fraction of the control reference, N a parameter to limit the high-frequency gain of the derivative function, and h the sampling period.

When the sampling period is h = 5 ms, we take the following control parameters. The controller is not optimized for the best control performance, and this is not the objective of this lab.

Table 1: The Parameters of the PID Controller

h	K	T_i	T_d	β	N
0.005	1	0.12	0.049	0.5	10

1 TrueTime Implementation of the PID Control

Create a TrueTime simulation model to implement the PID control of the DC motor. Figure 1 illustrates the connection of the TrueTime kernel block and other Simulink blocks. Evidently the TrueTime kernel block has two analogous inputs and one analogous output.

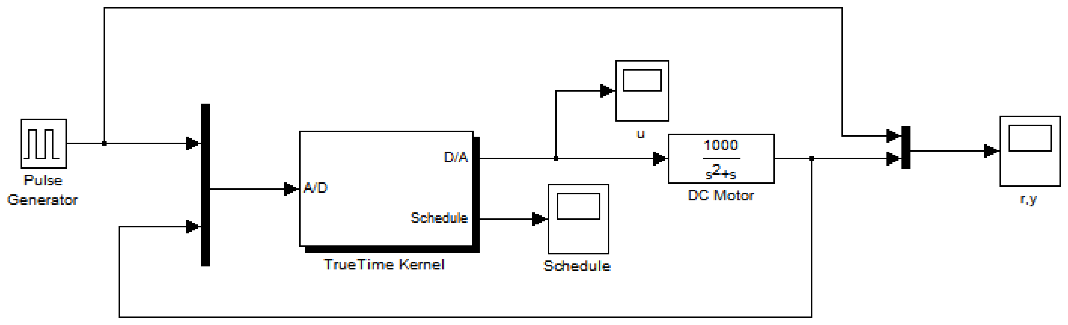


Figure 1: The TrueTime Model of One Motor Control

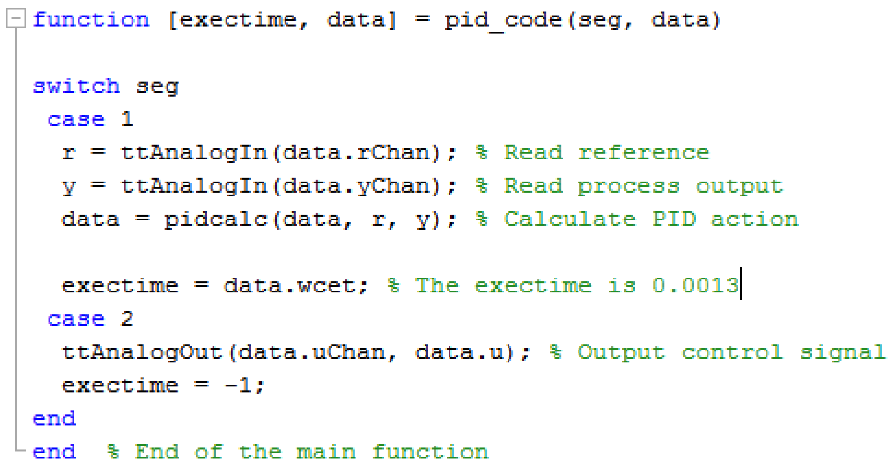
The reference signal *r* is a square wave produced by a Pulse Generator, whose parameters are given in the following table.

Table 2: The Parameters of the Pulse Generator

Amplitude	Period	Pulse Width(% of period)	Phase delay
2	1	50	0.1

Exercises: Please write the initialization function for the TrueTime kernel and the task function of the PID control task. *The execution time of the PID task is 1.3 ms*.

Recall the special code structure of the task function in TrueTime. Assuming the task function is named after `pid_code`, Fig provides hints to your design. Since there is just one periodic task, any scheduling method may be used.



```
function [exectime, data] = pid_code(seg, data)

switch seg
 case 1
  r = ttAnalogIn(data.rChan); % Read reference
  y = ttAnalogIn(data.yChan); % Read process output
  data = pidcalc(data, r, y); % Calculate PID action

  exectime = data.wcet; % The exectime is 0.0013
 case 2
  ttAnalogOut(data.uChan, data.u); % Output control signal
  exectime = -1;
end
end  % End of the main function
```

Figure 2: The Code Structure of the PID Task

Run simulation with your model and check if the motor position follows the reference position.

1. Which scheduling method do you use? How to select the rate-monotonic (RM) scheduling method?

2. How much are the *response time* and *latency* of the PID control task? How much is the *jitter* of this task? (Check the definitions in the lecture notes.)

3. How much is the CPU utilization?

2 Three PID Tasks on One Kernel

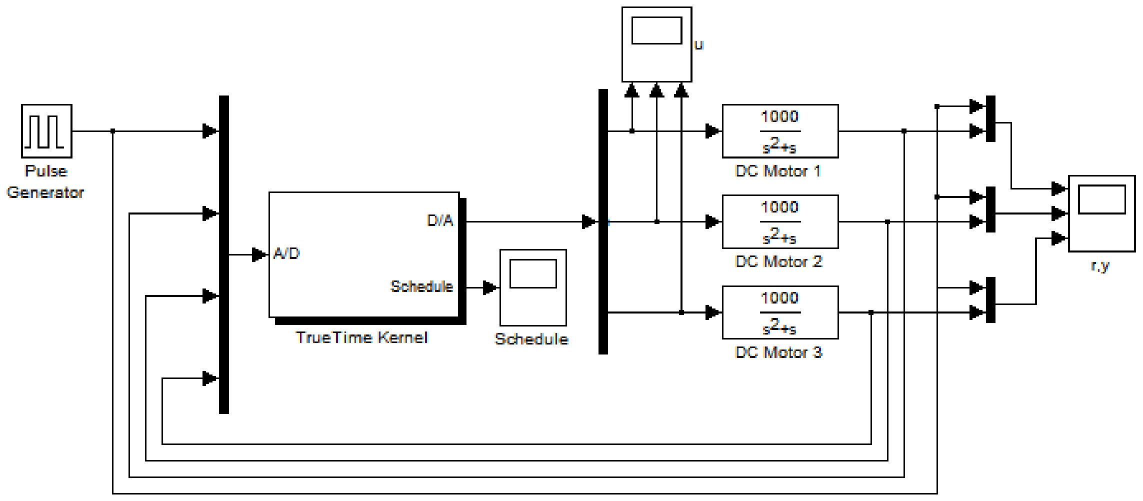
The CPU utilization of the microprocessor is low. More tasks may still be allocated to the processor. We consider the scenario that three identical DC motors are controlled by three independent PID controllers with the same parameters in Table 1. The only difference with these control tasks is the sampling period, as listed in Table 3.

Table 3: Periods of the Three PID Tasks

Task 1	Task 2	Task 3
6 ms	5 ms	4 ms

Task *i* controls DC motor *i*, (*i* = 1, 2, 3) and they all have the same execution time 1.3 ms. The TrueTime model of the system is given in Figure 3. Please enter the correct dialog parameters of the TrueTime kernel block and write the Matlab code to fulfil this control task. To simplify the code structure, please consider the following advices.

- The "Init function argument" parameter of the kernel block is an integer from 1 to 3. The three numbers indicate three different scheduling strategies, e.g., 1 for RM, 2 for FP, and 3 for EDF.
- The three PID control tasks can use the same code `pid_code` as illustrated in Figure 2. Different initial parameters and states of the control tasks are preserved in the `data` structure.



The TrueTime Kernel init function argument determines the scheduling policy used:
1: Deadline-monotonic (DM) / Rate-monotonic (RM) scheduling
2: Fixed-priority (FP) scheduling
3: Earliest-deadline-first (EDF) scheduling

Figure 3: The TrueTime Model of Three DC Motors

After you finish the necessary code functions, simulate the model and answer the following questions. Note that in the scheduling plot, the tasks are plotted bottom-up: the bottom plot is for task 1 and the top one for task 3.

Experiment

Use the RM scheduling method. Run simulation. Check if the controllers work properly.

1. Please list the tasks from low priority to high priority. Does the schedule plot confirm your answer?

2. Are these tasks schedulable under RM? What is the CPU utilization of the three tasks?

3. In the lecture, we have seen a formula to calculate the upper-bound of CPU utilization to successfully schedule multiple tasks under RM. Please write down the formula and calculate the upper bound for 3 tasks. Is the actual utilization smaller than the upper bound? If not, do you observe a violation of the theory?

4. What are the maximal response time and maximal latency of task 3?

Use the FP scheduling method and assign priority numbers 2, 3, and 4 to tasks 1, 2, and 3, respectively. Run simulation and check if the controllers work properly.

5. Please list the tasks from low priority to high priority.

6. Are these tasks schedulable under the given priority numbers? What are the maximal response time and maximal latency of task 3?

Use the EDF scheduling method. Run simulation and check if the controllers work properly.

7. What is the CPU utilization? Are these tasks schedulable?

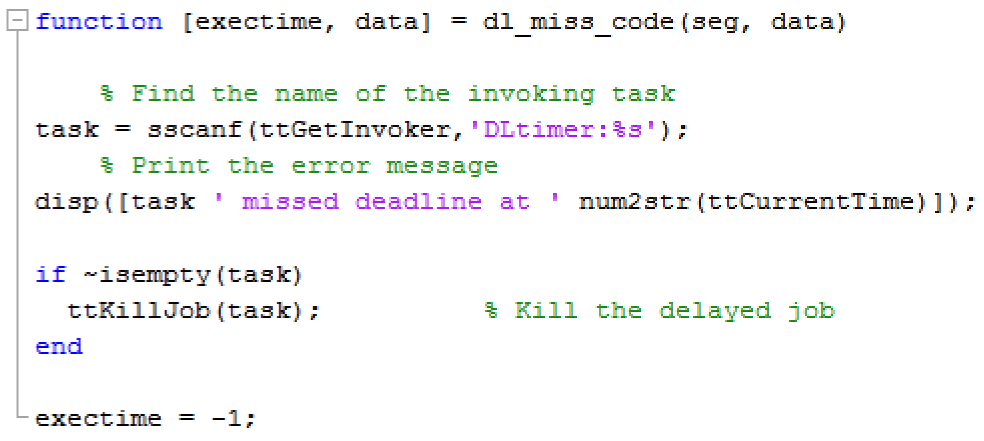
8. What are the maximal response time and maximal latency of task 3?

3 Deadline Overrun Handler

We have learned the deadline overrun handler in the last lab. When a real-time task fails to complete the computation before the deadline, a deadline overrun interrupt is then invoked by the real-time kernel to trigger the interrupt handler. In the last lab, the handler only prints a warning alert on the Matlab command window. We shall learn more functions of the deadline overrun handler.

1. We can attach one handler function to multiple real-time tasks so as to reduce the number of Matlab functions.
2. In the handler function, we can identify the task that invokes the deadline overrun handler when it misses the deadline.
3. In the handler function, we can kill the job that missed the deadline. This is a common operation performed by RTOS when a task misses its deadline. The reason is to confine the negative effect of the deadline overrun within the current failed job, so that it does not postpone the next job.

Create an interrupt handler in the initialization function of the TrueTime kernel and attach it to the three PID tasks as the deadline overrun handler. For the relevant TrueTime commands, refer to the last lab tutorial or the reference. The code of the handler function is given in Figure 4. The first line gets the name of the invoking task. Note that the invoker of the deadline overrun handler is the timer of the task. Thus if `pid_task1` misses the deadline, the return value of the command `ttGetInvoker` is `DLtimer:pid_task1`. The Matlab function `sscanf` then removes the prefix "`DLtimer:`" and assigns only string "`pid_task1`" to the variable `task`. The `disp` function prints the error message about when the task missed the deadline. The time is obtained by the TrueTime command `ttCurrentTime`. After the invoking task is identified, the TrueTime command `ttKillJob` kills the current delayed job.



```matlab
function [exectime, data] = dl_miss_code(seg, data)

    % Find the name of the invoking task
task = sscanf(ttGetInvoker,'DLtimer:%s');
    % Print the error message
disp([task ' missed deadline at ' num2str(ttCurrentTime)]);

if ~isempty(task)
  ttKillJob(task);          % Kill the delayed job
end

exectime = -1;
```

Figure 4: The Code of the Handler Function

Experiment

Change the execution time of all PID control tasks to 1.5 ms.

1. What is the CPU utilization of the three PID control tasks?

2. Choose either RM or FP in the kernel block. Do you notice any difference in the control performance? Please explain the reason.

3. Do you see any error message about deadline overrun on the Matlab command window? Please confirm these messages on the scheduling plot.

4. Choose EDF in the kernel block. Run simulation. Do you notice any improvement on the control performance? Why?

5. Do you see any error message about deadline overrun on the Matlab command window?

6. What are the maximal response time and latency of task3?

7. Change the execution time of all PID control tasks to 1.6 ms. Answer these questions again. What are your opinions on RM, FP, and EDF?

4 Summary

This lab reinforces the lecture on real-time scheduling of independent tasks on one processor. If all tasks are periodic, three preemptive scheduling strategies, i.e., RM, FP, and EDF, are often supported by many RTOS. Given the assumptions that these tasks start at the beginning of every period, have the deadline identical to the period, and do not communicate, the properties of the 3 scheduling strategies are well-known. RM and FP are both static scheduling methods and RM is the optimal one for all FP methods. They are easy to implement and supported by virtually all RTOS Generally, RM cannot fully utilize the CPU in order to keep all deadlines.

EDF, on the other hand, is a dynamic scheduling method and can meet the deadline requirements if and only if the CPU utilization is not greater than 1. It is slightly more difficult to implement and hence not supported by many RTOS.

After the lab, you should master the TrueTime skills for

- controlling multiple plants with one kernel block;
- writing the TrueTime code function to implement a control design;
- creating multiple tasks and handlers on one kernel block;
- selecting the suitable scheduling strategy to reach the satisfactory control performance;
- identifying the invoker of the interrupt handler.

In future labs, we shall study the real-time properties of dependent tasks on one microprocessor, which share memory, hardware resource, communicate, or have logical dependencies.