TTK4147 miniproject

Albert Danielsen
Bendik Standal

October 2018

1 Introduction

The goal of this project is to demonstrate the use of synchronization mechanisms and real time programming by controlling a simulation and responding to signals to and from a server. This report will explain our usage of threads, how they communicate and analysis for the results.

2 Threads

2.1 Number of threads

We have chosen to utilize three threads to make the program run and respond to the signals as smooth as possible. The threads include

- `periodic_timer_func()` Which sleeps for `period` seconds and then releases the `controller` semaphore the `controller_func()`-thread is waiting for.
- `controller_func()` Waits for the `periodic_timer_func()` to release the `controller` semaphore, then makes the call to request a new value from the server, before it finally calculates the input `u` to send back.
- `receiver_func()` Receives both the process variable `y` and `SIGNAL` from the server, then either responds to the `SIGNAL` or stores the received `y` in a global variable for the `controller_func()` to read. This is okay since these two threads will not try to read/write to this variable at the same time.

These threads represent the main processes that needs to run concurrently for the program to behave optimally, and meet the requirements for a responsive system.

2.2 Thread synchronization

The controller thread and the timer thread are synchronized through the use of semaphores. By making the controller thread wait for the timer to release its semaphore (and vice versa for the timer), we effectively create a period for the controller. Thus the deviations in period time (the interval between GET requests) is reduced from what it would have been had we not synchronized the threads. By introducing the `receiver_func()`-thread in task B, we also had to include a semaphore to wait for the a new process value and clearance to read the shared memory mentioned in 2.1.

3 Period and controller parameters

The controller parameters ended up being left unchanged from what was proposed in the assignment. Tuning them one way or the other did not lead to any significant improvement over the values we ended up with $K_p = 10$ and $K_i = 800$.

We experienced some trouble with using the `clock_nanosleep` function, it did not respond as expected when we attempted to set the period higher or lower. We therefore ended up writing our own function, `nanosleep_modified`, which is based on the `nanosleep` function. This modified function simply subtracts the wanted wakeup-time from the current time and sleeps for the resulting time.

The periodic time of the controller is important for the integral term and was found by simply timing the loop. This resulted in a period of $1500000ns = 1.5ms$.

4 Analysis of plots

To control the system a proportional and an integral term is needed as the system is exposed to some noise and we ended up with a significant steady-state error without the integral term. Our focus when designing a controller for the simulated system was for the period time to have as little deviation as possible. This was important because larger deviations would reduce the value of the output from the integral part of the controller. The response times in figure 1 shows quite fast and stable response time for the period between GET-requests and the subsequent SET-response.

In the second task we were tasked to respond to a signal from the server as fast as possible in addition to the tasks we were already doing. The addition of this task led to negative effects in the timing of the controller. To mitigate this we created a separate thread for receiving a signal, discerning whether its a process value or a signal that needed to be acknowledged. If it is the first case the controller's semaphore is released, if it is the latter case the signal is immediately acknowledged.

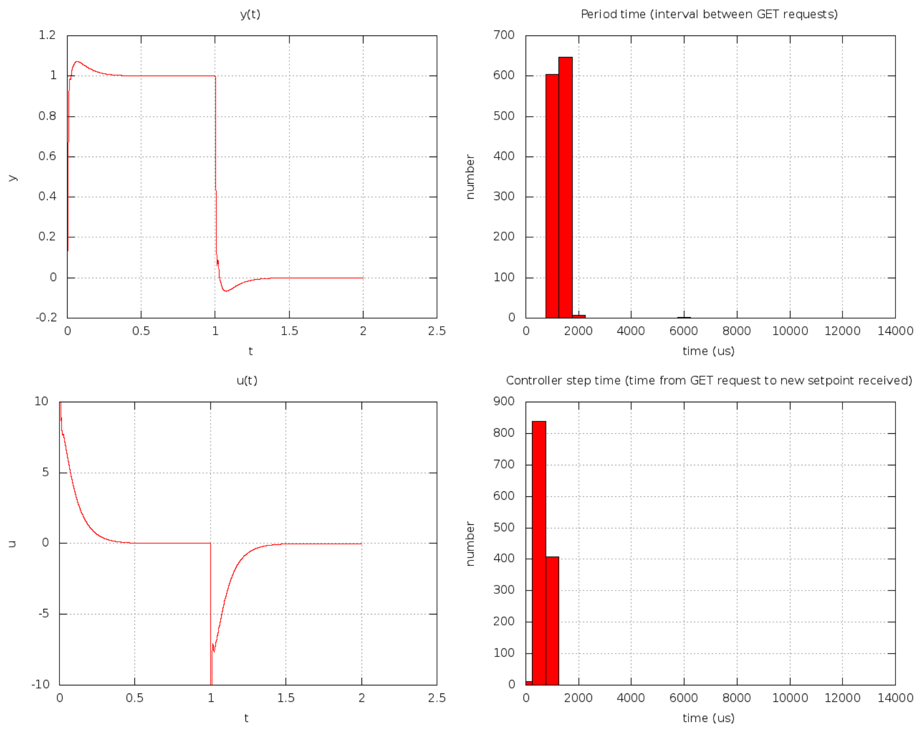

Figure 1: The results from task A. A small amount of overshoot when the setpoint changes is accepted in order to achieve a fast response. As there are no explicit requirements to the amount of overshoot, we see these results as satisfactory.

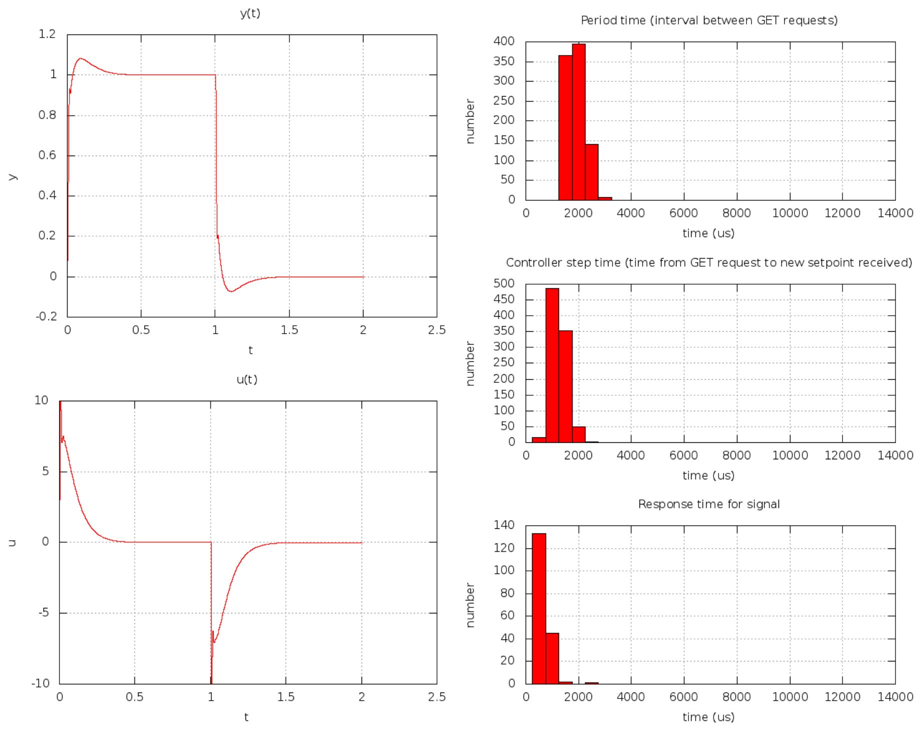

Figure 2: The results from task B. After trying different methods of synchronization we ended up going for an instant acknowledge and semaphore synchronization.