Mini-project: NGW100 PI-controller

The final part of the exercises in TTK4147 is a mini-project. There are two major differences between this mini-project and the earlier exercises. First of all the results should be delivered on blackboard and the results will be graded. Secondly you will only be given some tools and a problem to solve, and except for some requirements to the program, it is up to you to find the best solution.

Before submitting, make sure your group on blackboard contain all the group members. If this is not the case, you must send me an e-mail with the corrections when submitting.

The project is divided into two parts.

1. Part 1

The program `miniproject-server` is representing a physical system that must be controlled. It is running a simulation of a dynamical system in real-time that must be controlled. Your task is to create a program that runs on NGW100 that will control the dynamical system as well as responding to the signals. The interface between the two programs will be UDP packets over Ethernet.

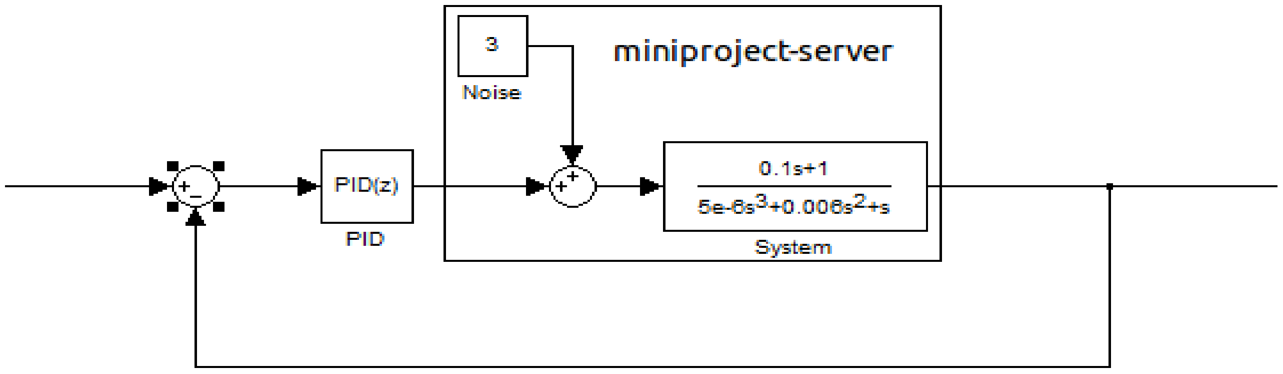
As the program runs, it logs the current state of the simulation and some other useful stats. When the program is completed the results can easily be displayed in graphs using the gnuplot tool.

1.1. The dynamic system

The dynamic system being simulated by the mini-project program has the following transfer function. Where y is the system output and u is the system input.

$$h(s) = \frac{y(s)}{u(s)} = \frac{0.1s + 1}{5 \cdot 10^5 s^3 + 0.006s^2 + s}$$

This system should be controlled with a PI-controller, with a Kp of 10 and Ki of 800. This controller should stabilize this unstable system and make it able to follow a set point. The system has a constant input noise of 3, which makes the integration part of the controller important to avoid the steady state error.



These controller parameters should give decent performance, but are probably not optimal. If you want to try other parameters or types of controllers you can do this, but it is not the focus of this project or course.

1.2. Miniproject.tar.bz2

Download this file from its learning. Unpack it and you will find the following four files:

- miniproject-server – The program that you should run on your Linux Mint computer, that act as a UDP server and simulates the dynamic system. You do not have the source code for this program, and you have to figure out how to communicate with it based on the information given in this exercise text.
- test-server – A program you can use to test your UDP packets.
- miniproject.h/c – Header and source files with useful functions for this project.
- plot4 and plot5 – Files used to plot the results.

1.3. Communication Interface

In addition to running a simulation of the dynamic system, the `miniproject-server` program is an UDP server that listens to port 9999. UDP packets are used to start and stop the simulation as well as for getting the current states and setting the inputs of the simulation. To make the communication with the server as easy as possible, a header file called `miniproject.h` has been created with some functions to help you. To see how to use these functions, look at the comments in the file.

The commands to the server are sent as strings, defined in the table below. Since creating the whole program at once is difficult, a `test-server` program have been created that will print any message you send to it. When the strings contain a value, you must get a double variable with the function `atof()`.

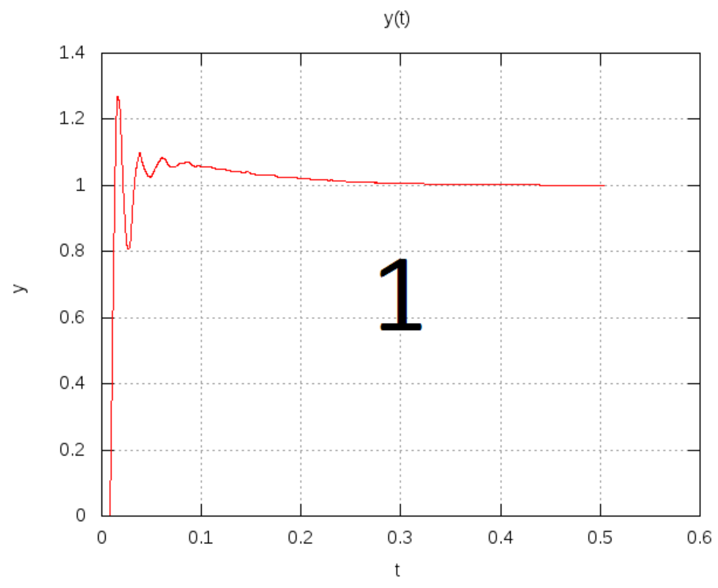
Command	String	Direction	Comments
START	“START”	To server	This packet starts the simulation
GET	“GET”	To server	This packet ask for the server to return a packet with y
GET_ACK	“GET_ACK:123.456”	From server	This packet is a reply from the server containing the y as a double.
SET	“SET:123.456”	To server	This packet contains the new u value as a double for the simulation
STOP	“STOP”	To server	This packet ends the simulation

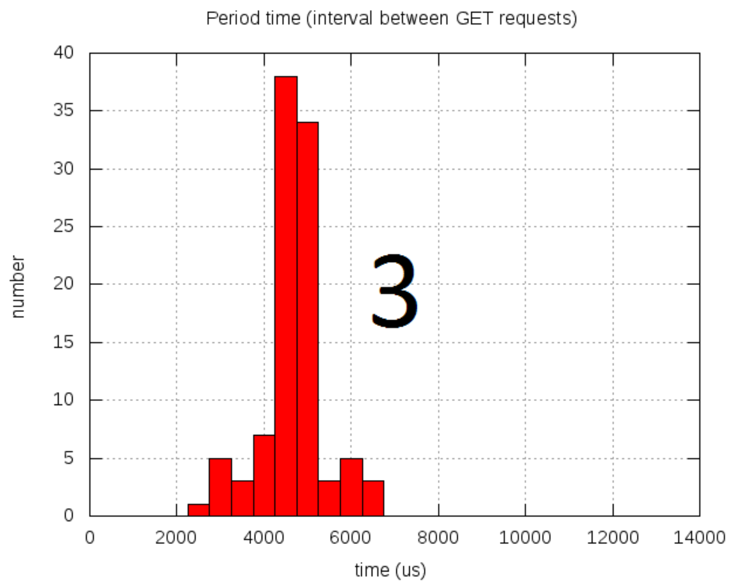
1.4. Gnuplot

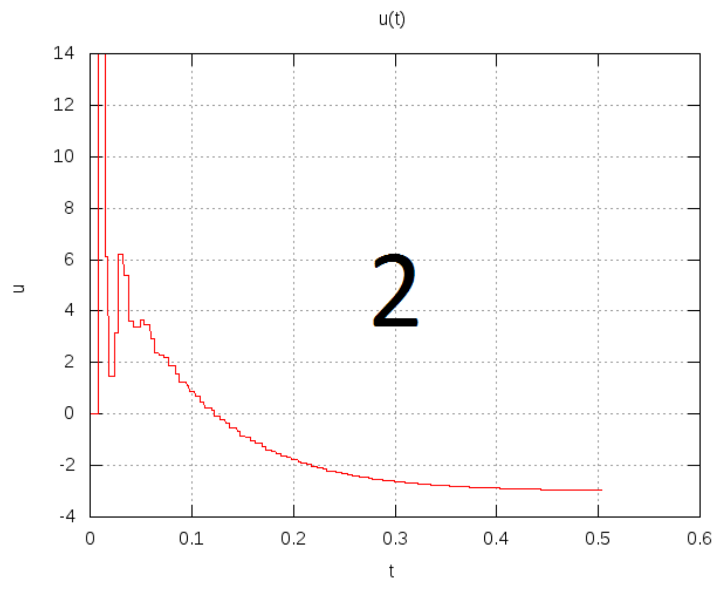
When `miniproject` completes two `.dat` files have been created in the same folder. Run the following command to create a PNG image containing graphs of the simulation results:

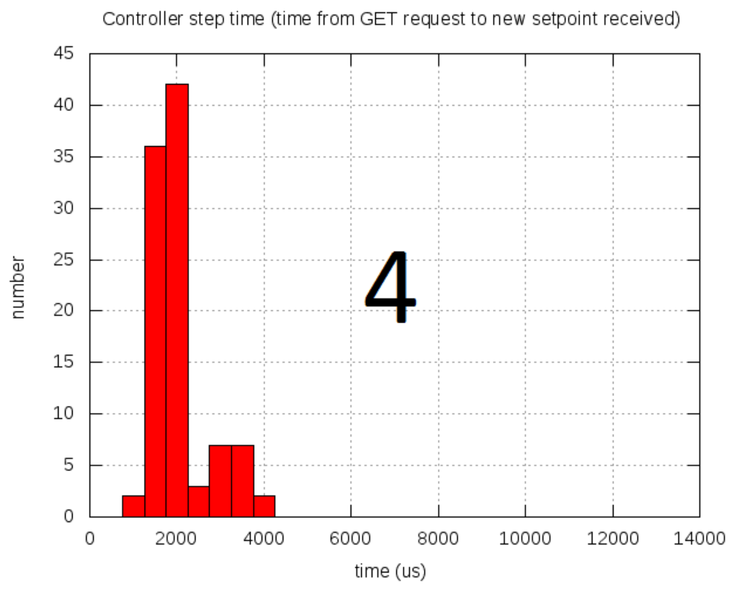
```
gnuplot plot4
```

An example of a plot is given below:









1. The output from the simulation of the dynamic system (y-value). It should ideally approach “1” quickly without any oscillations.
2. The input values (u-value) given to the simulation of the dynamical system.
3. A distribution/histogram of the actual periods of your PID controller. The period time is the time between the current GET request sent to the server and the previous one. Your periods should be very close to the period you have specified in your code, but some variance is expected from the `clock_nanosleep()` function and similar.
4. A distribution/histogram of the controller step time, i.e. the time the controller uses to get the system output, run controller algorithm and send the system input. It is calculated as the time from a y-value is requested, to a new u-value is received by the miniproject server. All this should be done within the period, so if any values are higher than the period time, it means that the controller missed one or more of its deadlines.

1.5. Programming in AVR32-Linux

Almost everything you have used in the previous exercises using Linux can be used on AVR32, including pthreads. Unfortunately the `clock_nanosleep()` function does not work. There are other ways to implement periodic execution of threads that works on the NGW100, but to keep things as simple as possible, I have implemented a simple `clock_nanosleep()` clone that you can find in supplied files together with the UDP functions. This function takes only one parameter, a `struct timespec` which specifies the time it should sleep until.

1.6. Configuring buildroot

Last time you created a standard buildroot for AVR32. You can use this, but to ensure the best possible results, you should recompile the kernel to get better real-time performance.

Move to the root folder of the buildroot. Type the following command:

```
make linux26-menuconfig
```

This will open the menuconfig for the linux kernel. You should look in the folder named “System Type and features”. Here you should select:

- Preemtion Model = preemtible kernel
- Timer frequeney = 1000Hz

You can also do other changes, but be careful as you might break the system.

Exit menuconfig, save the configuration and run “make” to recompile buildroot. It should be faster than last time.

When the compilation is done, you need to copy the new `uImage` to the TFTP server, and unpack the new `rootfs.tar.gz` to the `/export/nfs` folder as explained in the previous exercise.

1.7. Exercise

You should create a program that runs on the NGW100 that controls the simulation with a PI-controller. The program should read a y-value from the simulation and return a calculated u-value. The program should run for about 0.5 seconds before ending the simulation.

A pseudo code for a PI-controller is given below, and it must be executed periodically.

```
error = reference – y
integral = integral + (error * period)
u = Kp * error + Ki * integral
```

The period time is important for the control of the simulation. Too long period will give oscillation and might be unstable, while a too short period means that the algorithm will not complete in time and the controller period will be inaccurate. A too short period might be difficult (but not impossible) to see on the simulation output plot, but it will be very visible on the plot showing the measured period times of the controller.

You should find the step response to the controlled system, i.e. setting the reference to 1. The response should show that the y value is stable after about 0.2 seconds with a zero (or falling) steady state error and not too much overshoot in the start of the response.

1.8. Submitting

You should submit:

1. Your code.
2. PNG from gnuplot.
3. Explanation of how you found the controllers period and any other design choices you made. Keep your answer short and consise.

Be aware that deadline misses due to a too short period time will not be acceptable, even if the step response looks good.

1.9. Tips

- Remember to send STOP command to the server to stop the simulation. The simulation should run for about 0.5 seconds.
- Look at all the plots created by gnuplot, not just the step response.
- Don’t spend forever trying to get a perfect result, settle with “good enough” and explain why you think it is so. A perfect result might not be possible in this project.

2. Part 2

In this part we will introduce a second part of the `miniproject` program. In addition to simulating a dynamic system it also sends out signals that must be responded to, much like the response test of the BRTT.

2.1. Communication Interface

The signals are also sent as UDP packets, as described below.

Command	String	Direction	Comments
SIGNAL	"SIGNAL"	From server	This packet is a signal that should be responded with a SIGNAL_ACK
SIGNAL ACK	"SIGNAL_ACK"	To server	This is the response to the SIGNAL sent back to the server.

The server will start sending signals when the simulation is started, and run while the simulation lasts.

2.2. Gnuplot

When running the signal test an additional file with results is created. To plot the content of this file together with the ones from last part use the following command:

```
gnuplot plot5
```

In addition to the previous plot, an additional fifth plot has been created showing the distribution of the signal response time.

If the gnuplot command gives errors, it might be that the `signal_time.dat` file is empty. This is caused by the server not getting any responses to the signals.

2.3. Exercise

Extend your program from part 1 to also answer the signals. In the first part you could have implemented everything in one thread, but this will not be a good solution here (neither is it allowed). Your solution is required to (at least) have the following threads:

1. A thread that constantly listen to the UDP port. When it receives a packet, it will check it and "forward" it to another thread.
2. A thread that implements the PI-controller.
3. A thread that responds to received signals.

Thread 1 is the only one that should receive UDP packets. Multiple threads are allowed to send UDP packets, but you must make sure that two threads are not trying to send at the same time.

Some communication between your threads will be necessary, how you do solve this is up to you.

2.4. Submitting

You should submit the same as in part 1. In addition, you should describe the effect the signaling task had on the PID controller and why. And also describe the design choices you made. Keep your answer short and consise.