Module 8 - Implement your PI controller

Control Systems + TP, December 2024 v3

1 Introduction

This document is here to provide you with information and instructions about the practical. In this module you will implement your own PID controller. Please note that the software, this document and the whole infrastructure is still in beta stage.

1.1 Interface

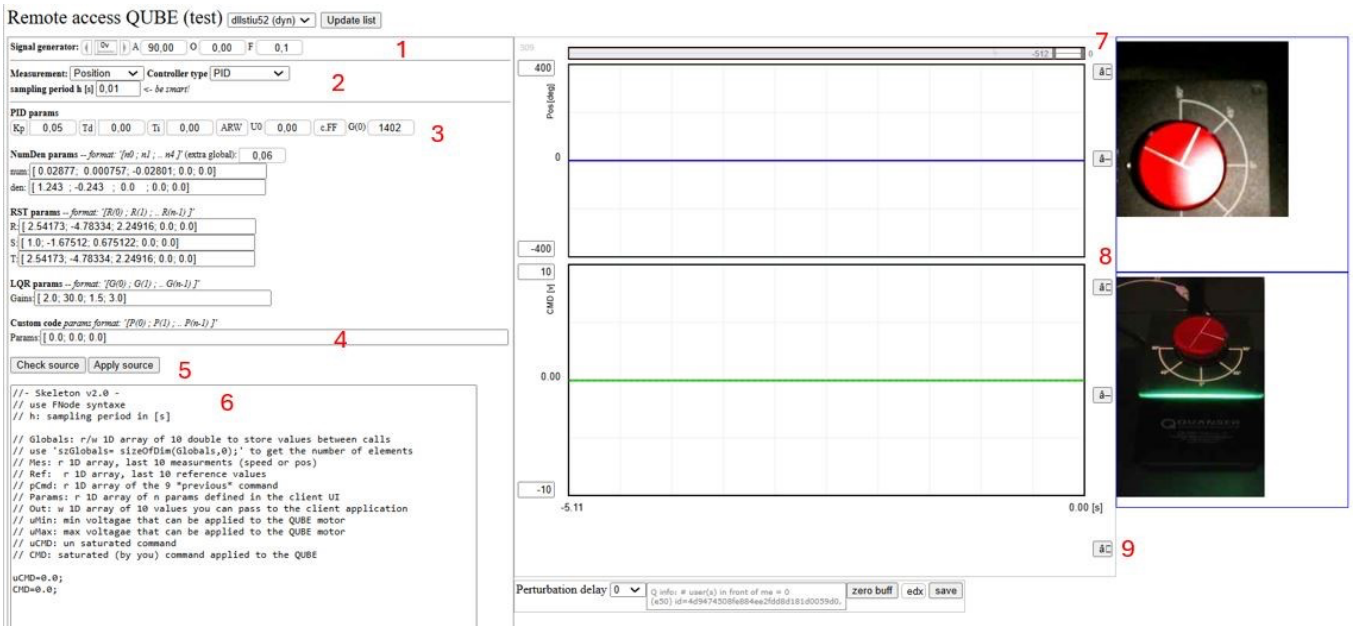


Figure 1: Interface used for the practical.

On Figure 1 you can find the different functionalities of the new web interface.

- 1: Similar to the old modules, you can control the **Signal generator**, i.e. the reference signal. Signal shape, amplitude, offset and frequency.
- 2: **Measurement** let you choose the input between position and speed, other options are not yet active. Similarly, the **Controller type** can be Open Loop, PID and Custom, and other options. Note that compared to old interface, the open loop mode is selected via the **Controller type**. *Custom* **Controller type** allows you to write your own C code (C subset), see point 6.
- 3: The **PID params** work the same as in the old interface, remember to click on the variable names will activate them. Note that Kp is always active in PID mode.
- 4: **Params** enables you to pass parameter to your custom code. The size limit is 10.
- 5: Each time you want to push new C code written in 6. First **Check Source**, then if no error click on **Apply source** to push the new code to the server. In case of error a message will be displayed. You can't push a bugus code. Remember that you need to be in **Custom** mode. The supported syntax is defined here : https://www.ni.com/docs/en-US/bundle/labview/page/formula-node-syntax.html

- 6: This block enables you to write your worn controller in C, a subset of C actually. Read the description of the various variable already at your disposal, which can help you a lot. Due to a typo **pCmd** can be accessed with **pCMD**. Furthermore, if you need for loops you have to initialize the loop variable outside of the loop.
- 7 & 8: The button enables to rescale the measurement window.
- 9: This button enables you to pause the measurement windows. You will be able to take different values.

1.2 Motivation

During this semester, through the various hand on sessions, you were able to apply some of the knowledge gained during the course: parameter identification, model-matching, Ziegler Nichols method on a basic mechanism. However, you never had to implement by yourself the various steps of such an algorithm. Such a task as an engineer can become tricky, and although in the future you will probably take a code found online, it is good to have done it at least once in its life to grasp the small details of such a control algorithm.

1.3 Usefull slides

Some code implementation examples can be found on the Moodle page of Week 14 in "PID implementation". The initial steps are very much like Module 1, 2 and 3.

2 Experiment

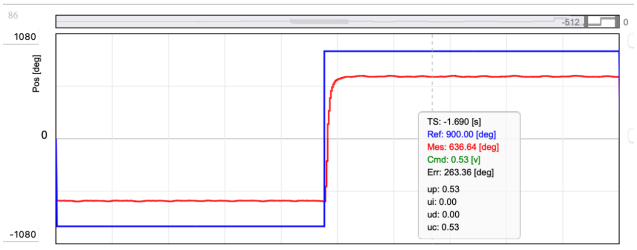
2.1 Parameter identification

The system at our disposal can be identified as a first order-system in speed mode:

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

Identify the two constants by making the measurements in speed and open loop mode. You will need to apply a step and stop the measurement's scrolling to compute the different values. What is the value after which the system is not affected by the friction anymore?
Hint: Module 1. As we are in open loop the input range can vary between 0 and 5V. The value you get are VERY different that what you get with in module 1.

You can use the new **Pause** button (9) to stop the graphs update. If you place the mouse over the 2 graphs a popup window will show the measurements under the mouse cursor. And computer both values without the need for the extra temporal fit tool. *Please note that the units for the measurement does not (yet) update, it remains deg when in speed it should be read as deg/sec. Same for the Y axis, it should be speed.*



2.2 Model Matching

We want to improve the system's response. This is done by imposing the closed-loop response, i.e. model matching method. Your goal is to identify the parameter Kp and Ti. The closed loop transfer function you want to match is:

$$T(s) = \frac{1}{1+\tau_m s} = \frac{K(s)G(s)}{1+K(s)G(s)}$$

Similarly to module 2, we want τ_m = 0.1sec. After computing the values change into PID mode and stay in speed mode. Apply a step that varies with time, does it behave as expected? Try different input values.
What is the new input range? Why?

Hint: Module 2. Your controller should look like $K(s) = K_p(1 + \frac{1}{T_i s})$.

2.3 Feed Forward

During the course we saw the feedforward control. Compute the compute the feedforward constant U_0, for a certain speed that you choose. Add it to your PI controller. Do you notice any difference? The new interface allows you to set a FF commande that is function of the reference signal. Set **G(0)** to the value you identify previously and click on **c.FF** button. Try various references values and check that the FF command works as expected.

2.4 Code implementation

The next step is to implement your own PID controller in C. The code is written in area (6), please note that the page does **NOT** save your code. If you reload the page the code is back to its default value. You need to keep track of your code in another docuement.

You have various variables that you can read/write

// **h**: (r) sampling period in [s]
// **Globals**: (r/w) 1D array of 10 double to store values between calls
// **Mes**: (r) 1D array, last 10 measurements (speed or pos)
// **Ref**: (r) 1D array, last 10 reference values
// **pCMD**: (r) 1D array of the 9 **previous** command values
// **Params**: (r) 1D array of **n** params defined in the client UI **(4)**
// **Out**: (w) 1D array of 10 values you can pass to the client application
// **uMin**: (r) min voltage that can be applied to the QUBE motor
// **uMax**: (r) max voltage that can be applied to the QUBE motor
// **uCMD**: (w) unsaturated command
// **CMD**: (w) saturated (by you) command applied to the QUBE

uCMD=0.0; // command before any saturation, will be displayed with dots in the graph
CMD=0.0; // the actual command applied to the QUBE, if greater than uMin/uMax will be saturated

// the current measurement is **Mes[0]**
// the current reference is **Ref[0]**
// the first parameter in the client UI **(4)** is **Params[0]**

// the current error is **float err = Ref[0] - Mes[0];**

2.4.1 Implementation of the PI controller

Implement your code in the code block. You should pass the parameters of your controller with the **Params** variable (4). You can also store local values in the **Globals** variable. The output command is **CMD**. You should start with a simple P controller (1-2 lines) then add UI computation using the trapezoidal approximation for the integral. Between calls the UI value should be save in **Globals[0]**.

2.4.2 Implementation of the ARW

Add a snippet of code to saturate the command and add the anti-reset windup. The VI already apply a saturation on the command. The variables **uMin** and **uMax** informs you about the existing saturation. If you command **uCMD** is greater that **uMax** you should saturate your integral term, do similarly with **uMin.**

Hint: Integral term saturation can be implemented by setting the current UI(k) term to the previous UI(k-1) stored one. The UI term can be stored in **Global[0]** for example.

2.4.3 Implementation of the PID

If time allows it you can add the computation of the derivative part UD.