# **System Identification** – Practical Assignment 3
# Transient Analysis of DC Motor Step and Impulse Responses

## Logistics and assignment introduction

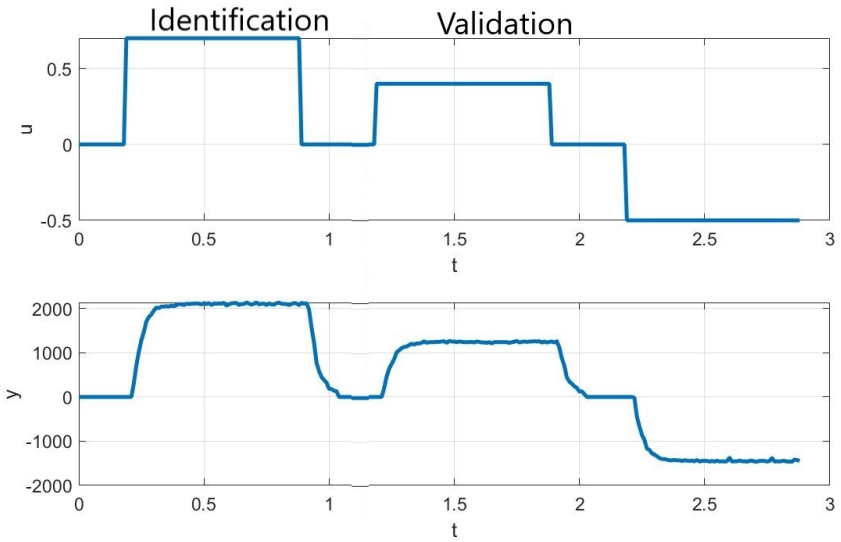
Please reread the logistics part of lab 2, the same rules will apply to this lab. The only thing that changes is the DropBox link, which will be communicated separately.

In this assignment we will perform transient analysis of first-order step and impulse responses for the DC motor, using real data – see the course material, *Transient Analysis of Step and Impulse Responses*.

Presumably you already did this in Lab 1, but if you did not, start by familiarizing yourself with the DC motor system and the ways in which input signals can be applied and output signals can be read, using the guide linked from the website.

## 1 Transient analysis of the step response

- Start by obtaining a dataset. To keep things simple, create a single, longer sequence of data containing both the identification and validation data. We will use a sampling period of $0.01$ s ($10$ ms). After a $0.3$ s range of zero inputs, apply a step input signal with amplitude $0.7$ and a length of $0.7$ s, followed by another range of zero inputs to return the system to the zero operating point, and then two step signals of length $0.7$ s, with magnitudes $0.4$ and $-0.5$ respectively, separated by a third range of zeros. Apply the input to the system and record the response; see the picture below for how you can expect a result to look like. **Important note**: To minimize system wear, separate the code that generates the data from the code that performs the rest of the steps below (easiest using different script sections, see *Code Sections* in the Matlab documentation), and **regenerate the data only when necessary** (e.g. not every time you change something in the transfer function).
- Isolate the data range corresponding first step and copy it to new input and output vectors; this will be our identification data.
- Develop a transfer function model of the system with the method described in the lectures, using the first step signal and response from the data. Include instructions that output to the console the transfer function, as well as the gain $K$ and time constant $T$, when your script is run.
- Validate your model using the validation data (the last two steps). Use Matlab function `lsim` to simulate the response of the identified transfer function to the validation input. The validation should consist of: (a) a plot where the system output is compared with the model output on the same graph; (b) and the computation of the MSE. Both of these results should be automatically produced by the Matlab code you provide.




## 2 Transient analysis of the impulse response

- To obtain the impulse-response dataset, construct an input that contains firstly a small signal with a value $u = 0.1$ and lets the system reach steady state (we therefore have non-zero initial conditions for the impulse response). Then, the input should apply a sequence of 3-4 impulses with value $u = 1$ and each of length 1-2 time samples; letting the system reach steady state again after each impulse. Apply the input and record the system response. **Important note**: Like for the step response, do not rerun this step everytime you change something in your code for the next steps.

- How much larger should the impulses be for them to be considered a correct practical implementation of an ideal impulse (do not try to implement this on the actual system, as it may exceed saturation limits)? Let's name this factor $\alpha$.

- Identify the transfer function from the first impulse. It is best to use $y_{ss}$ for obtaining the gain rather than $y_{max}$ (using $y_{max}$ is possible, but requires a correct rescaling by $\alpha$).

- Validate the identified model using the `lsim` function applied on validation data consisting of all impulse responses except the first. Important observations: (a) You should use a state-space model and take into account the nonzero initial condition. (b) The time vector used in `lsim` must be equidistant, unlike the real time vector obtained from the system, in which the sampling instants are not perfectly spaced. When simulating, use either the imposed sampling time, or the average sampling time computed *a posteriori* from the data. If the model deviates too much due to the timing, use `interp1` to resample the real-system output with a constant sampling rate and compare with that resampled signal instead of the aperiodically sampled, original signal.

Some relevant Matlab functions: `tf, ss, lsim, find, sum, interp1`. Operations on ranges of vectors will also be important.