Assignment 2
System Dynamics and Controller Design

In this assignment, you will design a control system for a linear positioner (i.e., a DC motor with a leadscrew). You will use a simulation tool to design the controller and later validate your design with a physical system.

The process consists of the DC motor that drives a gear box, which in turn moves a lead screw. The lead screw advances the position of the nut. The linear position of the nut is the output variable of the process. The motor and its components have a rotational inertia, $J$, and suffer from a frictional effect, $R_F$. The motor provides a torque, $\tau_M$, that is proportional to the driving voltage, i.e. $\tau_M = k_M \cdot V_M$. Note that voltage and current inputs are equivalent if we consider the motor coil as predominantly resistive. The balanced torque equation is given by Equation 1:

$$J \frac{d\omega}{dt} + R_F \omega = \tau_M = k_M \cdot V_M \quad (1)$$

The angular position is described by Equation 2

$$\phi = \int_0^t \omega \, dt + \phi_0 \quad (2)$$

where $\phi_0$ is the initial position and $\omega$ is the angular velocity. The lead screw is attached to a nut, and the $x$-position of the nut is determined by a gear box and by the pitch of the lead screw with a combined ratio of $p = 0.18 \text{ cm}/\text{rad}$. For each revolution of the motor, the nut therefore advances by $0.18 \text{ cm}$, and the linear position of the nut, $x$, can be represented by the equation $x = p\phi/2\pi$, where $p = 0.18\text{cm}/\text{rad}$. The initial position $x_0$ is related to $\phi_0$ through $x_0 = \phi_0 p/2\pi$. The leadscrew pitch and the gear box are two multiplicative parameters in the model that we have combined into one variable $p$; we will not consider them separately.

To sense the position, the nut is connected to a linear potentiometer of 10cm length. For now, we can assume the sensor (i.e., the potentiometer) to be normalized in such a way that the center position provides 0V, and the leftmost and rightmost positions give $-5\text{V}$ and $+5\text{V}$, respectively. The voltage of the potentiometer is proportional to the position $x$ with a gain of 1V/cm. A second potentiometer is used to vary the set point position.

---

1In reality, this is predominantly a reverse electromagnetic force, but it can be conveniently modeled as a friction

2The gear box has a translation ratio of 1:20, but this is not important in this context
$x_{set}$, which has the same voltage range as the potentiometer for the output position $x$.

The motor can be described by the following constants first defined in Equation 1:

$$J = 0.1 \frac{\text{Nms}^2}{\text{rad}}, \quad R_F = 0.5 \frac{\text{Nms}}{\text{rad}}, \quad k_M = 2.5 \frac{\text{Nm}}{\text{rad}} \cdot \frac{1}{\text{V}}$$  \hspace{1cm} (3)

1 Analysis of the Process

1.1 Theory

Determine the Laplace transform of the process while fully considering the initial conditions $x_0$ and $\omega_0$. Determine the zeroes and poles of the transfer function.

Henceforth, assume initial conditions $x_0 = x(t = 0) = 0$ and $\omega_0 = \omega(t = 0) = 0$. Perform an inverse Laplace transform (i.e. time domain) for a step function where $V_M$ is zero for $t < 0$ and jumps to $V_M = 1 \text{ V}$ for $t \geq 0$. Plot the time function.

1.2 Simulation

Draw a Scilab / Xcos block diagram that describes the process. The input is the motor drive voltage, $V_M$, and the output is the position $x$. Run a simulation of the process for a step input (1V, applied at $t = 0$ with $x_0 = x(t = 0) = 0$) and plot the position $x$ as a function of time until it reaches 1cm ($10^{-2} \text{ m}$). Compare this plot with the function plot you obtained in the previous section.

2 Design of the Controller

We will now add two elements to the motor/leadscrew system: First, ambient conditions may cause dust to accumulate in the leadscrew, thus causing a disturbance in the system. In the Laplace-domain computations, the friction can be modeled as a simple time-dependent torque $\tau_D(s)$ on the left-hand side of the balanced-torque equation. Second, a controller is added. The controller uses the voltage difference between the two potentiometers (setpoint $V_{set}$ and position $V_p$) to generate the error signal $\epsilon(s) = V_{set}(s) - V_p(s)$. The error signal is the controller input. The controller has the general unitless transfer function $H(s)$. The output voltage from the controller is fed
into a power driver, conveniently calibrated at \(1\text{V}/\text{V}\), to provide the motor drive voltage \(V_M\) with sufficiently high current capability.

### 2.1 Theory

For now, we have three possible controllers at our disposal:

- A P-controller with the transfer function \(H(s) = k_P\) (i.e., a simple linear amplifier)

- An integrator (I-controller) with the transfer function \(H(s) = k_I/s\)

- A combination of the above, that is, a PI-controller with the overall transfer function \(H(s) = k_P + k_I/s\).

Analyze the performance of the controllers as follows:

1. Determine the closed-loop transfer functions for setpoint and disturbance for the PI-controller. From here, derive the transfer functions of the P- and I-controllers by setting \(k_I = 0\) and \(k_P = 0\), respectively.

2. Determine the stable range for \(k_P\) and \(k_I\) when the controllers are used individually.

3. For the PI-controller, determine the joint requirements for \(k_P\) and \(k_I\) to keep the closed-loop system stable.

4. Provide expressions for the steady-state disturbance rejection for all controllers that are stable.

5. For the P-controller only, tune \(k_P\) to achieve the critically damped case (double-pole) and an underdamped case that allows approximately 4.6\% overshoot, but no subsequent undershoot. For these values of \(k_P\), what is the steady-state disturbance rejection in percent?

6. When you take the \(k_P\)-value for the critically damped case and add a small integral component with \(k_I = k_P/10\), what are your pole locations? What is the step response for a unit step input at the setpoint \(V_{set}(s) = 1\text{V} \cdot 1/s\)? What is the steady-state disturbance rejection for this controller?
2.2 Simulation

Complete the Scilab / Xcos block diagram of the process by adding the controller and the disturbance. The disturbance can be modeled as a step function that feeds into the torque summation point. To better distinguish setpoint changes and disturbance changes, delay the disturbance by 5 seconds, i.e., the disturbance becomes nonzero at $t \geq 5s$.

1. Use the P-controller with the two values for $k_P$ that you found in the previous section under Item 5. Plot the unit step response of your closed control loop for both values of $k_P$ and show that the step response matches the theoretical expectations with regards to overshoot and disturbance rejection.

2. From the step response plots, determine the settling time into a 2% tolerance band.

3. For the PI-controller with $k_I = k_P/10$ as in Item 6 of the previous section, plot the step response and determine the settling time. Show that the steady-state disturbance rejection matches your expectations from the theory section.

2.3 Design through Simulation

The design goal is to achieve the fastest possible settling time into a 2% tolerance band under the constraints

- Overshoot of less than 2% (tolerance band)
- Steady-state disturbance rejection of 2% or better (i.e., the disturbance is attenuated by a factor of 0.02 or smaller)

By using your simulation from the previous section, find a combination of $k_P$ and $k_I$ that (a) meets the constraints and (b) minimizes the settling time. Document some combinations that you tried in a table with $k_P$ and $k_I$, the settling time, and the disturbance rejection. What controller type with what parameters do you prefer?

TURNIN AND GRADING OF PART 1

This is a teamwork assignment. Each team provides one typewritten report. Grade points are assigned as follows:
3 Practical Controller Realization

In this part, you will compare the simulated positioner to a physical positioner. Your will have access to an assembly that is composed of the motor/gearbox/leadscrew assembly and a printed circuit board that holds the position and setpoint potentiometers, the motor driver, and a microcontroller unit that allows the motor to perform several tasks.

3.1 Determination of the Process Constants

How close does the real motor follow the model in Equation 1 and the corresponding block diagram given in Chapter 8 of the class compendium? Examine the motor’s step response as follows:

- Use the 9-pin serial cable provided to connect the serial port of the DC Motor to the serial port of any computer in Room 310 (Circuits Lab) or Room 209 (Undergraduate Lab). If you want to use your laptop, you may need an USB-to-RS232 converter.

- Start and set up a serial terminal program (see Appendix)

- Power up the DC motor controller model while holding down the pushbutton on the board for at least one second. The microcontroller should now be in an open-loop configuration where you can apply a step input to the motor by pressing the pushbutton. The motor returns to a rest position after you release the button. At the same time, the DC motor model sends its speed data to the serial terminal on the PC.

- Verify that you can receive the speed data: move the setpoint potentiometer all the way to the right, then press and release the
pushbutton. In the terminal program, you should see data similar to these:

<table>
<thead>
<tr>
<th>Motor power (%)</th>
<th>99.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t/ms</td>
<td>Enc/10ms</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>40</td>
<td>21</td>
</tr>
<tr>
<td>50</td>
<td>24</td>
</tr>
<tr>
<td>60</td>
<td>24</td>
</tr>
<tr>
<td>70</td>
<td>25</td>
</tr>
<tr>
<td>80</td>
<td>25</td>
</tr>
<tr>
<td>90</td>
<td>25</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

The motor power is determined by the setpoint potentiometer from 0% (0V) to 100% (12V). The subsequent data contain the time in ms (first column) and the number of encoder interrupts counted within a 10ms interval (second column). The encoder disc has 50 slots. A count of, for example, 25 interrupts in 10 ms indicates 1/2 revolution in 10 ms or 50 revolutions per second (rps). **You must perform this step before moving on.**

- Acquire the step response for at least 8 different power settings between approximately 20% and 100%. Plot the speed (in rps) as a function of time (in ms). For each curve, determine the time constant of the process and the equilibrium speed. Note that you need to use nonlinear curve fitting methods to obtain the time constant. Software for data analysis (Qtiplot, Gnuplot, Octave) can help you. Alternatively, use data transformation to use linear regression (see waterbath homework).

- Provide a table with the voltage (12V × power setting), the time constant and the equilibrium speed. In addition, plot the equilibrium speed over the voltage.

- Determine the constants $J$, $R_F$, and $k_M$ (Equation 1). If you can’t determine a constant, explain why and use the constant given
in Equation 3. You may assume that the motor is a first-order process (in fact, statistical analysis of the data, i.e., comparison of single-exponential versus double-exponential response, will favor the single-exponential model).

- Discuss similarities and differences of the model in Chapter 10.1 to the real-world DC motor you have examined. Specifically, identify where the model in Equation 1 fails.

3.2 P-Control: Simulation vs. Reality

Adjust your Scilab/xcos model to reflect the constants you found in the previous section. To match the loop gain, the sensor should be set to 102.4 cm$^{-1}$, because the controller uses an ADC value from 0 to 1023, depending on the potentiometer position from 0 to 10cm. If your position is in mm, use 10.24mm$^{-1}$ with the position range from 0 to 100mm. Your setpoint function should reflect the same range, i.e., 0 to 1023.

- In your Scilab/xcos model, find the value of $k_P$ that leads to critically damped behavior (no overshoot) and determine the settling time into a 5% tolerance band.

- In the DC motor assembly, find the value of $k_P$ that leads to critically damped behavior. You can partly automate this task with a function generator and an oscilloscope:
  
  - Connect a function generator to the first channel of an oscilloscope. Choose a square-wave signal of about 0.5Hz. This square-wave signal behaves like the repeated application of a step function.
  
  - Adjust the square wave amplitude (offset and gain) so that the square wave jumps between 1V and 3V. This corresponds to a setpoint potentiometer setting alternating between 2.5cm and 7.5cm.
  
  - Use a BNC T-piece to route the square-wave signal to the left BNC connector of the model. Slide the small switch on the board to the right. The model now obeys the external setpoint and should try to follow the square-wave signal.
- Connect the right BNC terminal to the second oscilloscope channel. Trigger the oscilloscope on the first channel. You should see both traces: the square wave setpoint and the actual position trying to follow the setpoint.

- Turn the $k_I$ and $k_D$ potentiometers to the left ($k_I = 0$ and $k_D = 0$). Monitor the oscilloscope traces and adjust $k_P$ to the critically damped case (no overshoot). Determine the 5% settling time.

- Compare the step response and the settling time between model and real-world positioner. Discuss similarities and differences. Specifically, what behavior (hint: nonlinearity) leads to the much slower settling time and the different slope near a step change?

- Complete your Scilab/xos model with the necessary nonlinear function and verify that the behavior of the model now matches the measured real-world data. In your report, show both the oscilloscope curves and the Scilab plot.

### 3.3 PID Controller Tuning

Find a combination of $k_P$, $k_I$ and $k_D$ to minimize the settling time. Overshoot is allowed as long as the overshoot does not exceed the 5% tolerance band. This task is similar to Section 2.3, except that you use a real-world system with a wider tolerance band and a saturation nonlinearity. Enter the controller parameters into your simulation and compare the simulation to the real-world system.

**TURNIN AND GRADING OF PART 2**

This is a teamwork assignment. Each team provides one typewritten report. Grade points are assigned as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Task</th>
<th>Description</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3.1</td>
<td>(process constants)</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>(P-control)</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td>(PID-control)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Turnin</td>
<td>Typewritten, legible, timely</td>
<td>5</td>
</tr>
</tbody>
</table>
APPENDIX: MO-SERIAL and HYPERTERM

The serial interface obeying the RS-232 standard is used for data transfer between widely different devices, for example, a data logger and a PC. Like USB, the RS-232 serial interface uses a 2-wire data connection. Unlike USB, the RS-232 protocol does not prescribe a software interface layer. The convenience of USB – automatic device recognition and automated setup of the data transfer – does not exist for RS-232. On the other hand, RS-232 does not need a special per-device kernel driver (the infamous driver CD-ROM that is required for almost every USB device in Windows). Unlike USB, the simplicity of the RS-232 protocol allows long-distance transfer over conventional phone lines. The easiest way to transfer data over a RS-232 interface is to send clear text. Software exists that can transmit and receive clear-text data and files. Two such programs are moserial (Free software 3, Linux, possibly Windows) and Hyperterminal (Windows only, not Free software). In the following section, the steps to set up the terminal programs for communication with the DC motor control model are provided.

moserial

- Get moserial from your friendly Linux repository or from http://live.gnome.org/moserial
- Start moserial from the start menu or by typing moserial in a console.
- Click on the Port Setup button
- Choose the device (/dev/ttyS0 for built-in serial interfaces and /dev/ttyUSB0 for USB cable interfaces)
- Choose the baud rate (bits per second) of 57,600, 8 data bits, 1 stop bit, no parity. Uncheck both handshake checkboxes. Click OK.
- Click the Connect button. You are now ready to transfer data.
- You may copy and paste received data from the terminal window, or use the Receive File button to capture data into a file.

3The capital F in Free indicates free as in speech as opposed to free as in beer. For more information, see http://www.gnu.org/philosophy/free-sw.html
Hyperterm

• Under the Start menu, go to Programs → Accessories → Communications → HyperTerminal

• Location Information: Enter Area Code 706

• Phone and Modem Options: Click OK

• Connection Description: Enter File Name (Icon Does Not Matter)

• Connect to: COM1 (built-in serial interface only)

• COM1 Properties:
  – Bits per second: 57600
  – Data Bits: 8
  – Parity: None
  – Stop Bits: 1
  – Flow Control: Hardware

• SAVE THE FILE (i.e., the terminal setup file)

• You can capture the data from Hyper Terminal in to a text file by going to Transfer Menu → Capture Text. Save the file. All replications will be saved to that file. When you are done, go back to the transfer menu and select Stop Capture Text. This will prevent you from having to manually copy and paste all of the data.