CONTROL LABORATORY ENG5022
SERVO-MOTOR CONTROL AND SYSTEM SIMULATION – CONTINUOUS AND DIGITAL
PART 1

This is **Part 1** of the lab handout.
You must first complete this part before continuing with Part 2.

TABLE OF CONTENTS

INTRODUCTION

OBJECTIVES

The aim of this exercise is to design and evaluate a controller which regulates the position of the output shaft of a servo motor.

The exercise has **two parts**: in **Part 1** you will model the system and assess the performance using a **linear model** of the system. In **Part 2** you will use a **realistic model** of the system which represents aspects you will encounter with a **real system**, so you can compare the performance with the linear simulation. The lab will run over **two sessions**, during which you are expected to complete **both parts**.

At the end of the two lab sessions, you should be able to:

- derive the mathematical model of the servo plant from first principles and how to implement this in MATLAB,
- appreciate the differences between the system simulation and the behaviour of the real system,
- design and implement a continuous controller,
- use the Nyquist stability criterion to estimate the robustness of the feedback control system,
- implement the continuous-time controller and evaluate its performance,
- create a digital equivalent of the plant for use in a simulation,
- design a digital controller by emulation, based on the continuous controller design,
- compare the performance of the digital controller with its continuous counterpart, and assess the impact of the sampling time on the digital control loop,
- evaluate the controllers with a real servo motor system, and compare their performance with simulations.

SOME PRACTICAL TIPS

You will be using MATLAB and Simulink 2020a or above throughout this lab. This software is available on School of Engineering computers which can be accessed on the Virtual PCs (COSE Desktop) https://rdweb.wvd.microsoft.com/arm/webclient/index.html. You can also install Matlab / Simulink on your own computer (create an account using your UofG email address on www.mathworks.com and you'll get a download link), or run the lab on the Mathwork webclient (https://matlab.mathworks.com).

.

ASSESSMENT

Each student must

1) answer the questions throughout this lab sheet and complete the corresponding **Moodle quizzes (one quiz for each Part)**, and
2) write a **short individual report**. The report must not exceed 3 pages (excluding plots and figures) and should contain:
   - your name and student number,
   - a description of the work done in the lab, explaining the tools and methods used,
   - a summary of the results obtained and the observations made, including all plots required in the lab sheet,
   - a discussion of what other methods could be used to improve the results.

SUBMISSION

You need to

- complete the **two Moodle quizzes** available on the course Moodle page
- submit
  - the **short individual report** as a **PDF file** under **Lab report submission,**
  - the 7 Simulink model files (extension ".slx") which correspond to the 5 models mentioned in this lab sheet under **Simulink model submission**

  on the Control Systems M Moodle page,

by the submission date shown on Moodle.

The Moodle quizzes and report will be assessed and contributes 10% to your final mark.

NOTE: This assignment is part of the assessment for this course, so you are required to complete the submission in order to gain credits. As the individual assignment is part of your summative assessment it is subject to the University's plagiarism policy (https://www.gla.ac.uk/myglasgow/leads/students/plagiarism/).

While you can discuss the lab with your fellow students, you must create the Simulink models and complete the lab individually.

1 PART 1: SIMULATION

1.1 OPEN LOOP SYSTEM

We will start by deriving the analytical model of the open loop system from first principles by modelling the electrical and mechanical components of the servo motor system. This plant model will then be implemented in MATLAB and we will analyse its frequency response, using the Nyquist plot as an analytical method and comparing this with experimental results. Finally, we will compare the open loop response of the model with that of the real system.

1.1.1 SYSTEM MODELLING

The Quanser QUBE-Servo 2 is a direct-drive rotary servo system. Its motor armature circuit schematic is shown in Figure 1 and the electrical and mechanical parameters are given in Table 1. The DC motor shaft is connected to the *load hub*. The hub is a metal disk used to mount the disk load, and has a moment of inertia of J_h. A disk load is attached to the output shaft with a moment of inertia of J_d .

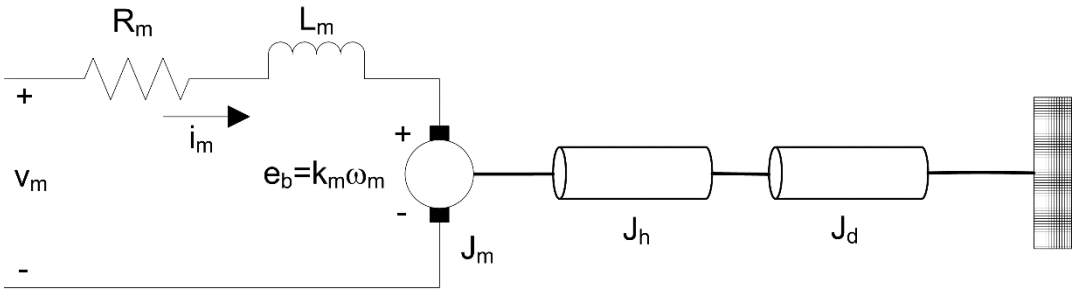


Figure 1: QUBE-Servo 2 DC motor and load

Here v_m is the supply voltage, R_m and L_m are the motor resistance and inductance, respectively, e_{emf} is the back emf produced by the motor as it turns and i_m is the current through the motor. The motor torque is denoted by τ_m, the shaft angle is θ_m and the motor shaft velocity is $\omega_m = \dot{\theta}_m$.

Using Kirchhoff's voltage law we obtain the following equation:

$$v_m(t) - R_m i_m(t) - L_m \frac{di_m(t)}{dt} - e_{emf}(t) = 0$$

Since $L_m \ll R_m$, the motor inductance can be ignored, and the equation can now be simplified to

$$i_m(t) = \frac{v_m(t) - e_{emf}(t)}{R_m}$$

The back emf created by the motor is proportional to the motor shaft velocity, $\dot{\theta}_m = \omega_m$, i.e. $e_{emf}(t) = k_m \dot{\theta}_m(t)$. Replacing in the equation above results in

$$i_m(t) = \frac{v_m(t) - k_m \dot{\theta}_m(t)}{R_m}$$

The torque generated by the motor is proportional to the current through the motor,

$$\tau_m(t) = k_t i_m(t)$$

where k_t is the motor-torque constant. Replacing i_m with the equation above results in

$$\tau_m(t) = k_t \frac{v_m(t) - k_m \dot{\theta}_m(t)}{R_m}$$

We now consider the mechanical aspects of the motor. Applying Newton's 2nd law of motion to the motor shaft results in

$$(J_m + J_h + J_d)\ddot{\theta}_m(t) = \tau_m(t)$$

where J_m, J_h and J_d are the moment of inertia of the motor, the hub and the disk, respectively.

The moment of inertia of a disk about its pivot, with mass m and radius r, is

$$J = \frac{1}{2}mr^2$$

which allows us to calculate J_h and J_d, given their mass and radius. The total equivalent moment of inertia is therefore,

$$J_{eq} = J_m + J_h + J_d = J_m + \frac{1}{2}m_h r_h^2 + \frac{1}{2}m_d r_d^2$$

Thus

$$J_{eq}\ddot{\theta}(t) = k_t \frac{v_m(t) - k_m\dot{\theta}_m(t)}{R_m}$$

or

$$J_{eq}R_m\ddot{\theta}(t) + k_t k_m\dot{\theta}(t) = k_t v_m(t)$$

Q1.1: Take the Laplace transform of this equation and derive the transfer function $\boldsymbol{G(s) = \frac{\theta(s)}{V_m(s)}}$

You should now implement this system in MATLAB.

Using the MATLAB editor, write a MATLAB script to calculate the numerator and denominator coefficients of the transfer function, using the physical parameters given in Table 1 at the end of this section.

Bring this transfer function into monic form, i.e. make sure that the coefficient associated with the highest power of *s* in the denominator is equal to 1. This can be achieved by dividing numerator and denominator by the coefficient associated with the highest power of *s* in the denominator.

Q1.2: Write down the resulting transfer function in monic form:

Q1.3: What are the poles and the zeros of the plant transfer function?

1.1.2 *CREATION OF TRANSFER FUNCTIONS IN MATLAB*

You will use the function `tf` from the Control System toolbox to implement the transfer function which you have derived, in MATLAB, using a *transfer function object* (TF object).

The expression `P = tf(num,den)` creates a continuous-time transfer function with numerator(s) and denominator(s) specified by the vector-variables `num` and `den`. The output variable `P` is a *TF object* storing the transfer function data.

In the Single-Input Single Output (SISO) case, `num` and `den` are the row vectors of numerator and denominator coefficients ordered in descending powers of *s*. These two vectors need not have equal length and the transfer function need not be proper.

For example,

- `h = tf([1 0],1)` specifies the pure derivative $h(s) = s$,
- `g = tf([15],[6 3 2])` specifies the proper transfer function $g(s) = \dfrac{15}{6s^2 + 3s + 2}$.

Amend the MATLAB script which you have created above to implement the system transfer function using the `tf` command and assign it to the variable `P`.

Symbol	Description	Value
DC Motor		
R_m	Terminal resistance	8.4Ω
k_t	Torque constant	0.042 N.m/A
k_m	Motor back-emf constant	0.042 V/(rad/s)
J_m	Rotor inertia	4.0×10^{-6} kg.m^2
L_m	Rotor inductance	1.16 mH
m_h	Load hub mass	0.0106 kg
r_h	Load hub mass	0.0111 m
Load Disk		
m_d	Mass of disk load	0.053 kg
r_d	Radius of disk load	0.0248 m

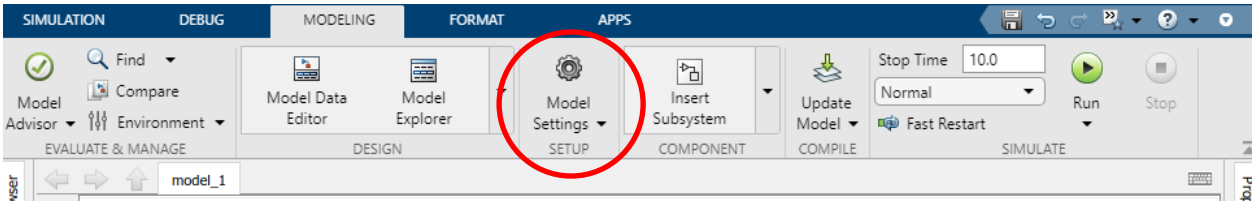
Table 1: QUBE Servo 2 system parameters.

1.1.3 *SIMULATION OF THE PLANT MODEL*

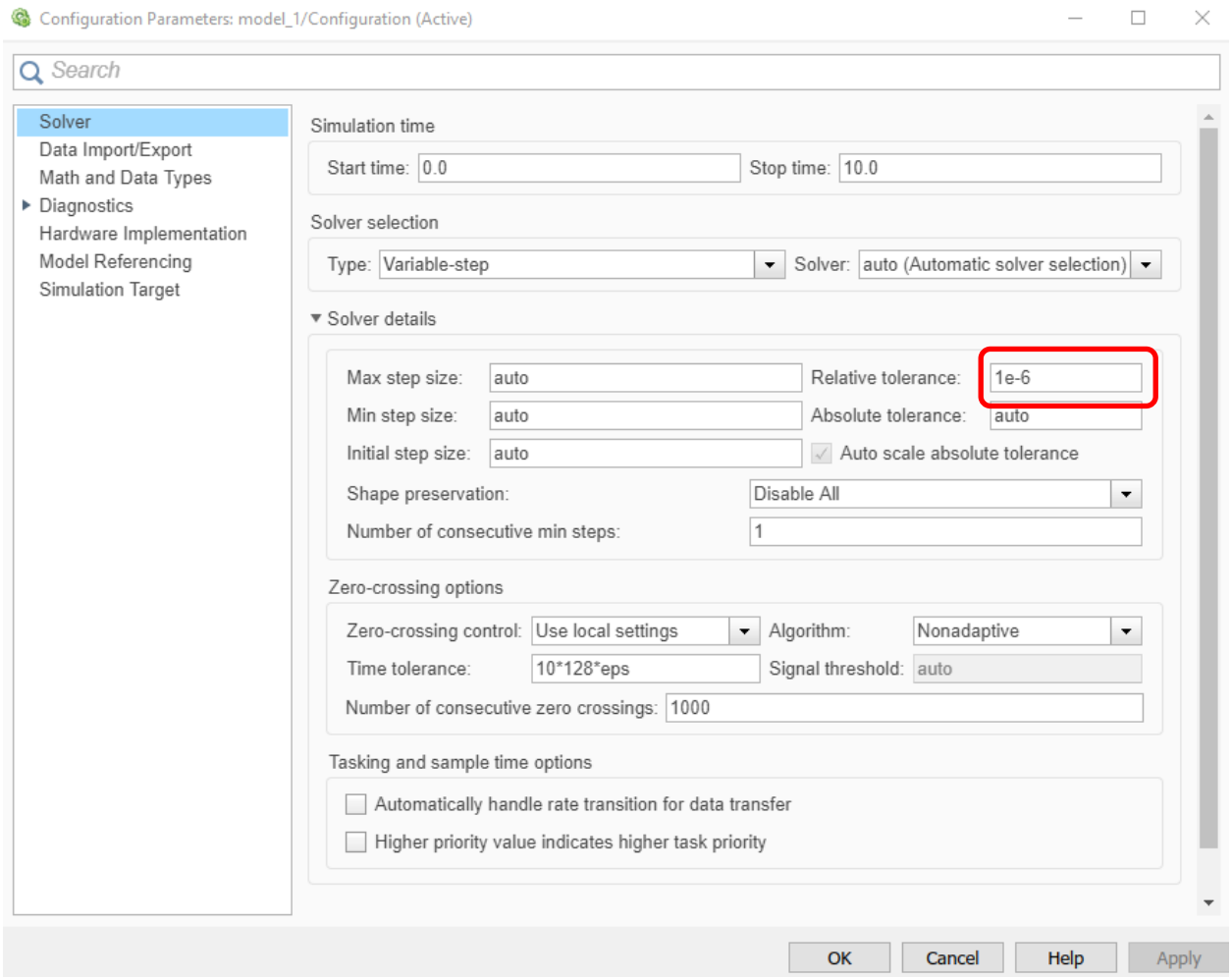
In this part you will implement a simulation of the plant model in Simulink and evaluate this.

From within MATLAB, open Simulink and create a new model. By default, the model is set to simulate a continuous-time system.

To increase the simulation accuracy, you need to make a change to the model settings which can be found under the **Modeling** tab.

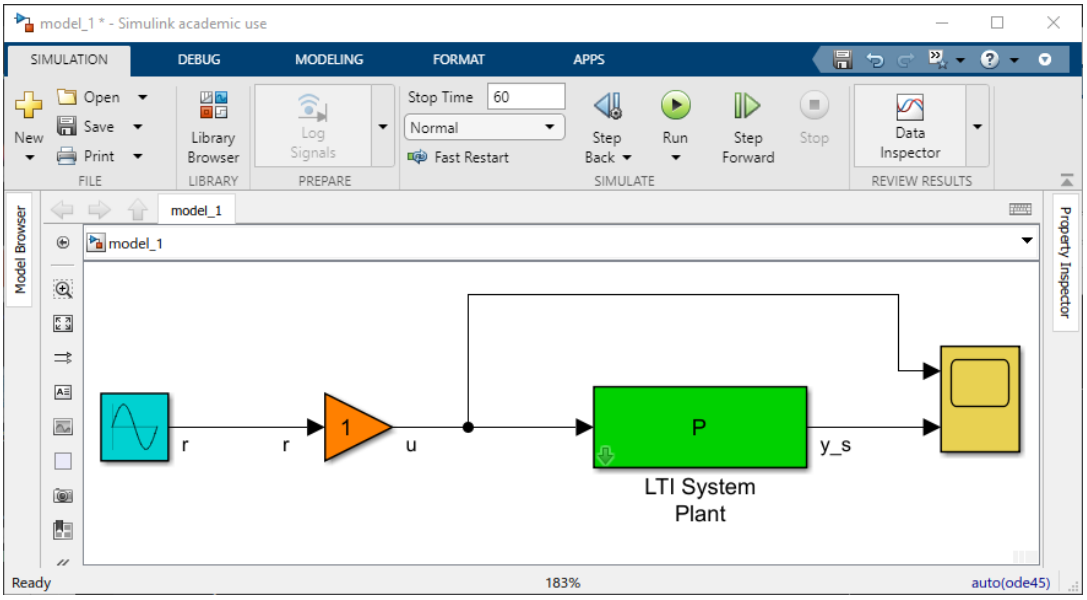


In the Configuration window, select the **Solver** option on the left and expand the **Solver details**, then set the **Relative tolerance** to **1e-6**, as shown in the figure below.



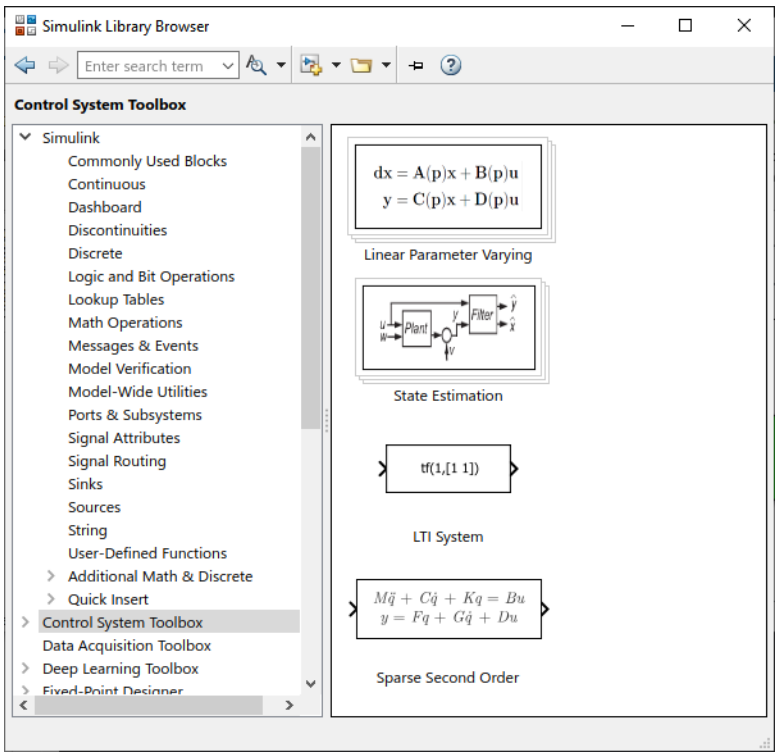
Hit OK to apply these settings to the model!

Use the basic building blocks in the library to reproduce the system under analysis in Simulink. Drag and drop the blocks from the library window to the model window, then connect the block inputs/outputs together (see Model 1). Set the simulation time to 60 (seconds).



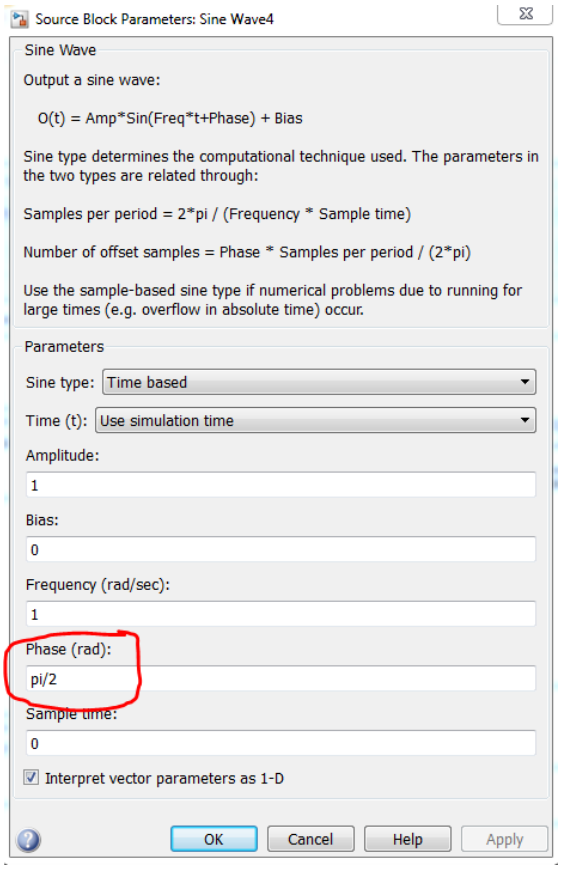
Model 1. Open-loop system with simulated plant

For the transfer function, you can use the block called *LTI System*, which encodes a transfer function previously defined in the workspace; the block can be found in the library *Control System Toolbox*.



Once you have dragged the block into your model, double click on it to open its properties and make sure that the system variable for the plant TF has the same name as the variable of your Simulink *LTI System* block (e.g. `P`).

The system input should be a *Sine Wave* block (can be found in the library *Simulink\Sources*). Double click on the block to set the parameters of the sine wave as following:



Input and output of the system are plotted on a *Scope* block (can be found in the library *Simulink\Sinks*). Double click on the *Scope* to open the plot and view the signals (after running the simulation).

Scope block setup

Open the Scope and click on the *Settings* icon (top left). On the *History* (or *Logging*, depending on version) tab:

- **Tick** the box "Save (or log) data to workspace" and choose a variable name. Set "Format" as "Structure with time". This will allow you to save the data generated and plot them later for your report.
- **Note:** You can find the variable as part of the structure name **out** in the Matlab workspace, i.e. `out.ScopeData`.

Plotting and saving results

You can plot the data generated using the command `plot` with the elements of the ScopeData structure (saved from the Scope) as arguments. `ScopeData.time` is the time, `ScopeData.signals(1).values` are the elements of the first Scope panel, and `ScopeData.signals(2).values` are the elements of the second Scope panel. For example, to plot the data from the lower Scope panel (i.e. the output of the real servo and the output of the simulated model) over time, you can use the command:

```
plot(ScopeData.time, ScopeData.signals(2).values)
```

You can also save the variable to a file, so you can plot it later (for example for the report).

We suggest saving every Simulink model and corresponding MATLAB script you will create. You can use a sequential number that matches model numbers in the figure in this hand-out, e.g. model1.slx and script1.m.

Run the Simulink simulation for the two different input frequencies of 1 rad/s and 5 rad/s, and record the output amplitude and the time shift between input and output signal at each frequency.

Q1.4: Describe what you observe. What is the difference between the simulation at 1 rad/s and at 5 rad/s? Does this correspond to what you would expect?

Include plots of the simulation responses in your report.

1.2 CONTINUOUS-TIME CONTROLLER

In this part of the experiment we will use a loop-shaping approach together with the root-locus method to select controller gains. The closed loop system will then be analysed using the Nyquist stability criterion and simulations. **In Session II, you will implement the controller with the real servo system and evaluate its performance.**

1.2.1 CONTROLLER DESIGN

Firstly, we will select a suitable compensator $C(s)$ in the continuous domain. Note that the plant already has a pole at 0 (i.e. integrative behaviour), hence the loop gain $L(s) = C(s)P(s)$ also has a pole at zero, and $L(j\omega)$ has therefore infinite gain at $\omega = 0$. For this reason, we do not need to include an integrator in the compensator: A **PD controller** structure (equivalent to a *lead compensator*) can hence be used:

$$C(s) = K_p + K_d \frac{s}{\tau s + 1}$$

where K_p and K_d are the proportional and derivative gain factors, respectively. The denominator term with the time-constant τ is required to make the derivative term realisable. This structure can be rewritten as

$$C(s) = K \frac{T_D s + 1}{\tau s + 1}$$

or

$$C(s) = KD(s)$$

with $D(s) = \frac{T_D s+1}{\tau s+1}$.

We will design the controller in two steps: first, we setup the controller frequency response defined by $D(s)$, using, and then we will choose a suitable controller gain K using the **root-locus method**.

1.2.1.1 LOOP SHAPING FOR D(S)

We can decide to use the zero of the compensator (at $-1/T_D$) to cancel the high-frequency pole of the plant (which is located at approximately -10, see your calculations earlier on). We can then add an extra pole at $-100 = -1/\tau$ to ensure that the compensator is *proper* (i.e. realisable). The resulting compensator transfer function $D(s)$ is:

$$D(s) = \frac{\frac{1}{10}s+1}{\frac{1}{100}s+1} = \frac{\frac{100}{10}s+100}{s+100} = \frac{10s+100}{s+100}$$

Implement the transfer function $D(s)$ in MATLAB using the `tf` commands (as you have done for the plant), calling the TF system `D`.
Note that the overall compensator is $C(s) = KD(s)$, but for now we assume $K = 1$.
Evaluate the effect of the compensator on the loop gain $L(s)$ by evaluating $L(s) = C(s)P(s)$ for $K = 1$.

In MATLAB, you can simply multiply the transfer functions of your compensator with the transfer function of your plant, i.e. `C=K*D;` and `L=C*P;`.

Generate a bode plot showing $P(s)$, $C(s)$ and $L(s)$ (using `bode(P,C,L)`). To add a legend (showing which line belongs to which transfer function), use `legend P C L`.

Show the stability margins (right-click in the plot, select `Characteristics->Minimum Stability Margins`).

> **Q1.5: Plot the Bode diagram and include this in your report. Explain the shape of the magnitude and phase of P(s), C(s) and L(s), based on the poles and zeros of the transfer functions. In which way does C(s) affect the loop gain (considering the phase and the magnitude)?**

Q1.6: What are the cross-over frequencies for P(s) and L(s)? What are the gain and phase margins?

1.2.1.2 CONTROLLER GAIN K

So far, we have analysed $C(s) = KD(s)$ with $K = 1$. We will now analyse the system to select a more suitable value for the gain *K*, using the *root-locus* analysis.

The *root-locus* of a feedback loop can be obtained using the MATLAB command `rlocus`. See `doc rlocus` to get more information on how to use this command.

On the MATLAB command prompt, enter the command `rlocus(L, [0:0.001:5])`.

This plots the root-locus of the system for values of the feedback gain *K* ranging from 0 to 5. The three 'x' (one at the origin, one at approx. -10, the other at -100 on the real axis) indicate the location of the open loop poles (i.e. for a feedback gain of 0), the 'o' indicates the open loop zero (from the compensator, which cancels the pole at -10). The three lines show how the locations of the poles change as the feedback gain is increased. If you left-click on one of the lines you will see a pop-up window showing the feedback gain at this point of the root-locus, the location of the pole, and the corresponding characteristics of the closed loop (damping, overshoot and frequency).

Using the root-locus plot, find the feedback gains for which

A. **the damping is 1.0 and the frequency of the slowest pole is 5 rad/s,**
B. **the overshoot is 5%.**

Q1.7: Note the values for the feedback gains in the table below and determine the locations of all three poles from the root locus plot for these gains.

		Location of poles		
	Feedback gain K	**Pole 1**	**Pole 2**	**Pole 3**
A				
B				

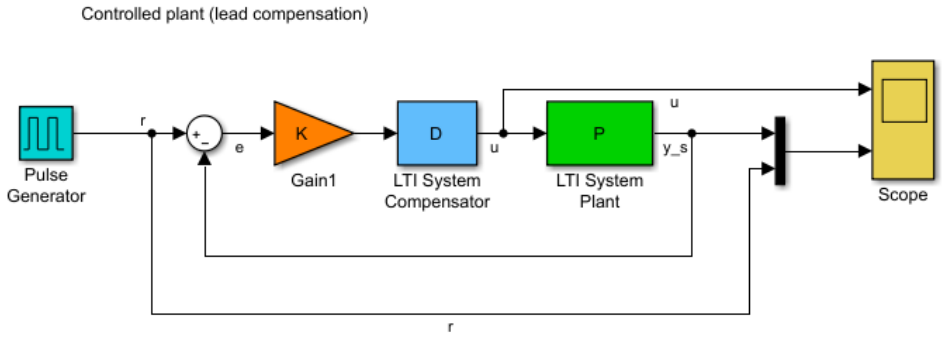
Table 2: Root locus gains and poles

1.2.2 SIMULATION WITH SIMULINK

Create a model similar to 1 (the open loop system with simulated plant), however this time introduce closed-loop feedback with a controller *C*(*s*), as shown in the figure below (Model 2).

In this model, the control loop is now closed (feedback control) and the *Gain* block is the feedback gain. As a signal source, you can use a *Step* or a *Pulse Generator* (sequence of steps). If you use the pulse generator, adjust the "pulse width" to 50% of the period, to give enough time to the system to settle at each pulse.

Make sure the Scope block is set up as described on page 9. You can also copy-paste the Scope block from Model 1 by drag-and-drop, preserving its properties.



Model 2. Feedback system with compensator and simulated plant

Set the *Controller Gain* value in the model to the values you have found above and evaluate the responses by running the simulation for each gain value.

In the top plot of the Scope, the control signal `u` is shown. In the bottom plot, the reference and the output are displayed.

Verify that the overshoot and damping specified from the root-locus analysis correspond with those obtained in the simulation. Look also at the control signal which is shown in the top plot of the scope.

Q1.8 Plot `y_s` and `r` against time for the two different gain values. How do the responses for the two gain values differ?

Include the plots of the simulation responses in your report.

1.2.3 NYQUIST ANALYSIS

You will now use the Nyquist plot to analyse the stability of the system.

Generate the Nyquist plot of the loop gain for the two gain values (`K_A`, `K_B`, corresponding to A and B in Table 2) which you have found above and include this in your report. Use the command `nyquist(K_A*L,K_B*L)` where `K_A` and `K_B` are the values of the gains.

The point '-1' is marked with a red + in the plot. The *Nyquist stability criterion* states that the closed loop system becomes unstable if the Nyquist plot encircles '-1'. You can derive a *phase-margin* (where the Nyquist plot crosses the unit circle) and a *gain-margin* (where the Nyquist plot crosses the negative real axis) from this plot which indicate by how much your control loop can change before it becomes unstable – in other words, how *robust* the control loop is.

From the right-click menu in the Nyquist plot, select *Characteristics -> Minimum Stability Margins*. This will display the unit-circle and mark the points at which the plots cross it. This indicates the *phase margins*. If you move the mouse over this point you will see a pop-up showing the phase-margin, the frequency and the resulting *delay margin.* Note that the *gain-margin* is infinite in our system (because the Nyquist plot never crosses the negative real axis).

Q1.9: For the two gain values obtained above, determine the *phase margins* together with their *frequencies* and the corresponding *delay margin.* Enter the values you find in the table below.

	Phase margin (deg)	**Frequency (rad/s)**	**Delay margin (s)**
A			
B			

Table 3: Margins

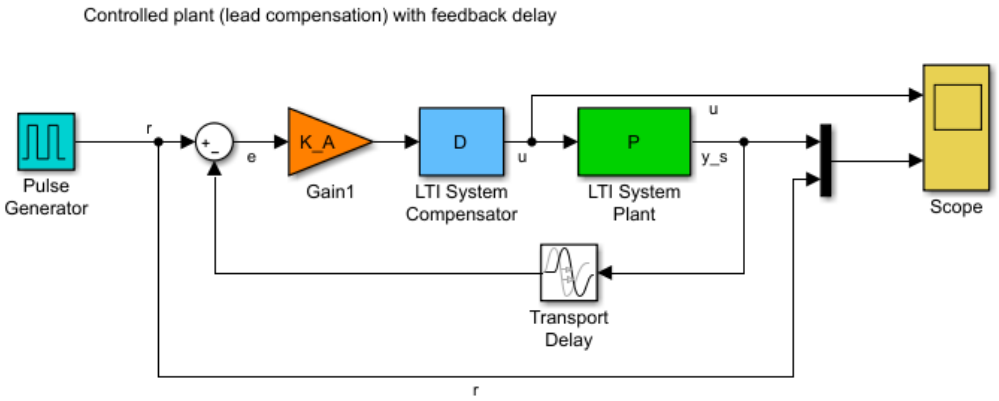
Q1.10: Generate the Bode plot of the loop gains for the two gain values using the command `bode(K_A*L, K_B*L)` and include them in your report. Determine the phase margin from this plot and compare to those in Table 3.

The phase margin ϕ determines how much "phase" can be added to the system before it becomes unstable. "Phase" is related to a time-delay, using the frequency ω at which it occurs:

The time-delay ΔT associated with the phase ϕ (in rad) for a given period of oscillation, T, can be calculated using the equation $\frac{\Delta T}{T} = \frac{\phi}{2\pi}$. The frequency of oscillation, ω, in rad/s, can be converted to the period using $T = \frac{2\pi}{\omega}$. Inserting this into the equation above, the time-delay at a specific frequency ω can be calculated as $\Delta T = \frac{\phi}{\omega}$.

This means that the system remains stable as long as the time-delay added to the loop is smaller than the *delay-margin* resulting from the *phase-margin*.

Evaluate this in a simulation for the value of the gain for <u>case B</u>. In the Simulink model of Model 2, add a *Transport* Delay block (from Simulink\Continuous), as shown in Model 3 below. Enter the gain value from case B (`K_B`) in the Controller Gain block. Then change the *Time Delay* value of the Transport Delay block to a value slightly smaller than the delay margin which you have determined from the Nyquist plot and run the simulation. Repeat this with a value which is slightly larger than the delay margin.



Model 3. Feedback system with compensator and simulated plant, and added feedback delay.

Q1.11 Include the plots of `y_s` and `r` against time for both cases in your report. How does the system behave in both cases?

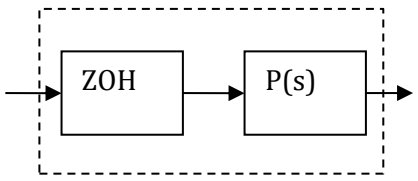
1.3 DIGITAL CONTROL

The scope of this part of the laboratory is to analyse the control loop in simulations, assuming that it is implemented using a digital computer, with a non-infinitesimal sample time. We will see that if the sample time is small enough, then the digital controller performance is relatively similar to the continuous one. Conversely, as the sample time increases, the performance of the digital controller degrades until instability appears.

We will design the digital controller by emulation, i.e. by emulating the continuous controller designed in the previous section.

1.3.1 DISCRETE EQUIVALENT OF THE PLANT

Before implementing a digital controller, we need to find the discrete equivalent model of the (continuous) plant `P(s)`. This is done assuming that the plant is preceded by a zero-order hold (ZOH):



P_zoh(z)

Q1.12: Given the plant transfer function `P`, using what you have learnt during the course, and the transforms tables, find the discrete equivalent `P(z)`. Assume a sample time T = 0.01 s.

$$P_{zoh}(z) = \left(1 - \frac{1}{z}\right) \mathcal{Z}\left\{\frac{P(s)}{s}\right\} = \ldots = \frac{az + b}{z^2 + cz + d} =$$

The same result can be found using the MATLAB function: `P_zoh = c2d(P, T, 'zoh')` which converts a continuous system to its discrete equivalent using the "zoh" rule and a sample time of T. **Verify your result.**

1.3.2 DIGITAL DESIGN BY EMULATION

We will now convert the designed continuous controller D(s) into a digital one, using one of the "emulation" techniques. Design by emulation essentially means that we an existing controller, designed in the continuous domain, into the discrete domain. In other words, the discrete controller "emulates" the continuous one, trying to achieve similar characteristics.

Here in particular we will use the "pole-zero matching" technique. This technique will be explained during the class, however it consists of three steps to compute poles, zeros and gain for a discrete transfer function, given its continuous counterpart. It relies on the correspondence between poles/zeros in the s-plane and in the z-plane.

1. Each pole s_i in the s-plane is mapped into a pole $z_i = e^{s_i T}$ in the z-plane.
2. Each finite zero s_i in the s-plane is mapped into a zero $z_i = e^{s_i T}$ in the z-plane.
    a. Zeros at $s = \infty$ are mapped to $z = -1$

3. The gain K_d of the digital controller is selected in such a way as to preserve the behaviour at a critical frequency, in this case we select $s = 0$ (corresponding to $z = 1$):

$$K_B\, D(s)\big|_{s=0} = K_d\, D_d(z)\big|_{z=1}$$

Q1.13: For the controller gain from B (`K_B`), find the zeros and poles of the discrete transfer function of the compensator, Dd(z), and its gain (`K_d`) analytically:

$K_d =$

$D_d(z) = \dfrac{z+a}{z+b} =$

Implement the compensator transfer function in MATLAB with the variable `Dd`, using the command `Dd = tf(NUM,DEN,T)`, where `NUM` and `DEN` need to be set according to the poles and zeros found through the rule above, and T is your sample time (0.01s).

Define also the digital gain `K_d`.

Note that `tf(…,…, T)` now creates a discrete TF because the sample time T is specified.

Q1.14: Use your report to show the Bode diagrams of the continuous and digital compensators using the bode function in MATLAB: `bode(K_B*D, 'b', K_d*Dd, 'r--')`. Describe the differences/similarities at low and high frequencies. Graphically or by other means, estimate the phase (deg) at 200 rad/s of both continuous and digital compensators.

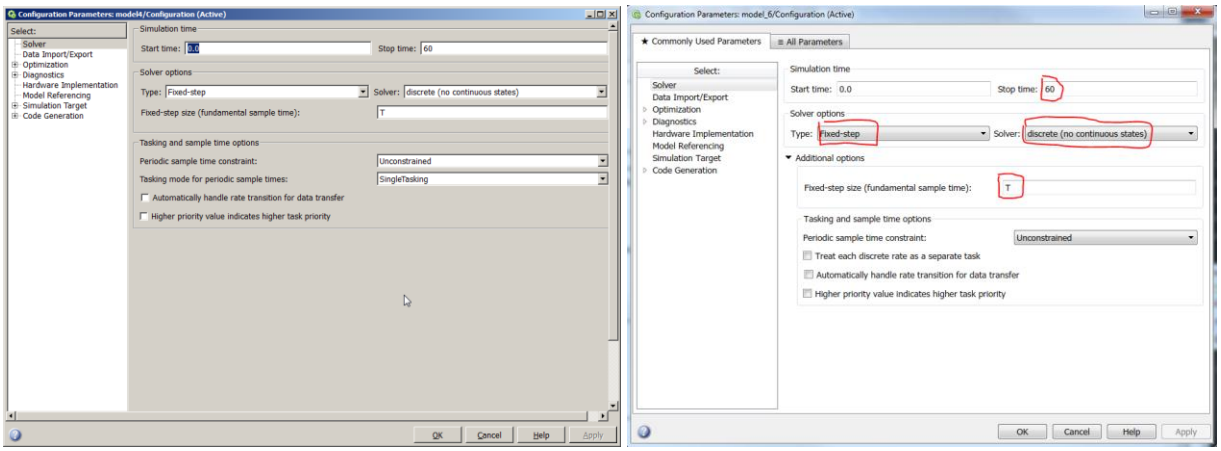
1.3.3 *Simulation of the Digital Controller*

Let's now set up a discrete (digital) simulation in Simulink.

Create a new Simulink model. Open "Simulation->Model Configuration Parameters" and select the "Solver" page. Set the following parameters:

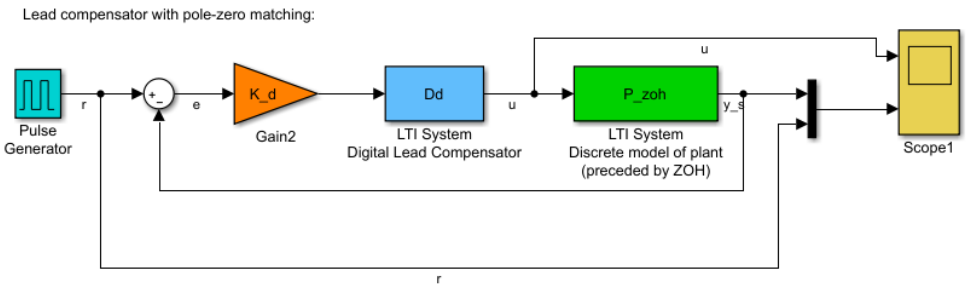
- **Solver options-> Type: fixed-step**
- **Solver: discrete (no continuous states)**
- **Fixed-step size: `T` (Simulink will pick the value of `T` from MATLAB workspace)**
- **Stop time: 60 (seconds)**
- **Tasking mode for periodic sample time: "SingleTasking"**

Define the sample time variable `T` in the MATLAB workspace as `T = 0.01`.



This Simulink model is now configured to simulate a discrete system, using difference equations and discrete transfer functions.

We will first test the performance of the digital controller using the digital equivalent of the plant. Build a discrete feedback loop as shown Model 4 below, using the transfer functions found above. Note the use of discrete transfer functions `P_zoh` and `Dd`.



Model 4. Digital feedback system with compensator and simulated plant

Q1.15: Use your report to compare the time history of `y_s` of the simulations of the digital feedback control loop (Model 4) with the analogue (continuous-time) implementation (Model 2). How does the performance compare in terms of overshooting and settle time?

1.3.4 EFFECT OF THE SAMPLE TIME

We will now increase the sample time, and re-assess the performance of the controller.

Set the sample time to `T = 0.04` in the MATLAB workspace.

Q1.16: Re-compute the discrete equivalent of the plant `P_zoh`, the lead compensator `Dd` and its gain `K_d` for the new sample time.

$$P_{zoh}(z) = \frac{az+b}{z^2+cz+d} =$$

$$K_d =$$

$$D_d(z) = \frac{z+a}{z+b} =$$

1.3.4.1 SIMULATED PLANT

Run the digital feedback system (Model 4) again, with the new sample time of 0.04 s.

Q1.17: Use your report to plot again the time history of `y_s`. Is it still stable?