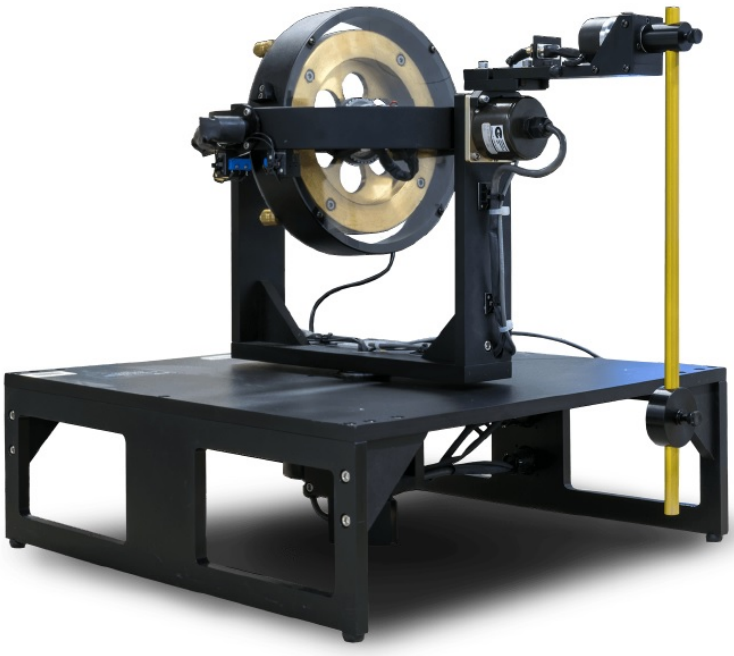
TUHH
Technische Universität Hamburg

ICS
Institute of Control Systems

Control Lab
ATC 2

LPV Control of a Gyroscope with Inverted Pendulum Attachment



23rd April, 2021

Room: Online | N-1.077

Summer Semester 2021

Contents

About this Document

This document is intended to help you with the design task *ATC 2—Gain-Scheduled Control of a Gyroscope with inverted pendulum* of the *Control Lab* practical course. It is written mainly in the style of a tutorial and should provide you with all the necessary tools and Matlab commands to solve the task. There are several Matlab files that you need to modify and execute in order to develop your own design. You are also encouraged to write your own code!

design_CMG_swing.m This file deals with the synthesis of an LPV controller designed for swing-up of the pendulum.

design_CMG_stab.m This file deals with the synthesis of a separate high performance LPV controller for stabilization of the pendulum at upright position.

These m.files are meant to provide a structure very similar to the tutorial given in this document.

! You need to complete the code on your own and submit the file.

simCMG_full.slx This file contains the nonlinear simulation model.

simScriptCMG.m This file load data of both swingup and stabilizing controller and simulate the entire model.

! You may need to modify the block diagram to suite your own design.

There are also several other auxiliary files provided. The code is guaranteed to work with Matlab 2016b 64bit, other versions might not be supported. You can get the latest Matlab version from http://www.tuhh.de/rzt/usc/matlab/index.html or use the pool computers. In addition to these files, we will provide you with a Pre-Release version of the LPVTools toolbox.

In this document, you will encounter blocks that indicate Matlab code:

```
[MATLAB COMMANDS] = USEFUL(TOOLS)
```

These are meant to get you started. You can (and should) use the `help` command within Matlab to find out more about a particular command.

Another thing that you will encounter are preparation tasks:

Preparation: Think about the benefits of gain-scheduling when compared to a single robust controller.

These are meant to prepare you for the question session that will take place prior to conducting the experiment.

Task

Design an LPV controller for the control moment gyroscope with an inverted pendulum attached to it and evaluate it in nonlinear simulation. Attach all necessary MATLAB and SIMULINK files that were provided to you no later than one week before the experiment via email to the responsible Tutor and Supervisor. You will get an email when your preparation is not sufficient to pass the Lab and will get time to revise your design.

Checklist

- ☐ I read this whole document carefully.
- ☐ I did all preparation tasks and can explain them.
- ☐ I completed `design_CMG_swing.m'.
- ☐ I completed `design_CMG_stab.m'.
- ☐ I submitted all MATLAB files.

1 PLANT

Control moment gyroscopes (CMGs) are used in various applications from attitude control in spacecrafts to stabilization of ship motion to only mention a few examples. Gyroscopes in general also fulfill prominent roles in motion sensing and navigation. From a dynamical aspect, CMGs correspond to coupled nonlinear systems with challenging rotational dynamics affected by friction and pose dependent disturbances due to manufacturing imperfections. Hence, they can be used as a test-bed for nonlinear controller design.

Attaching an inverted pendulum to one of the gimbals makes the already nonlinear coupled CMG even more complex. In this lab, the complex task of swing-up and stabilization of a CMG-actuated inverted pendulum is divided into simpler tasks to enable design of simpler controllers, thereby improving tractability of the synthesis conditions and achieving better performance for this complex system. One LPV controller is designed for swing-up with an energy based computation of its reference trajectory. While for stabilization, a separate high-performance LPV controller is synthesized.

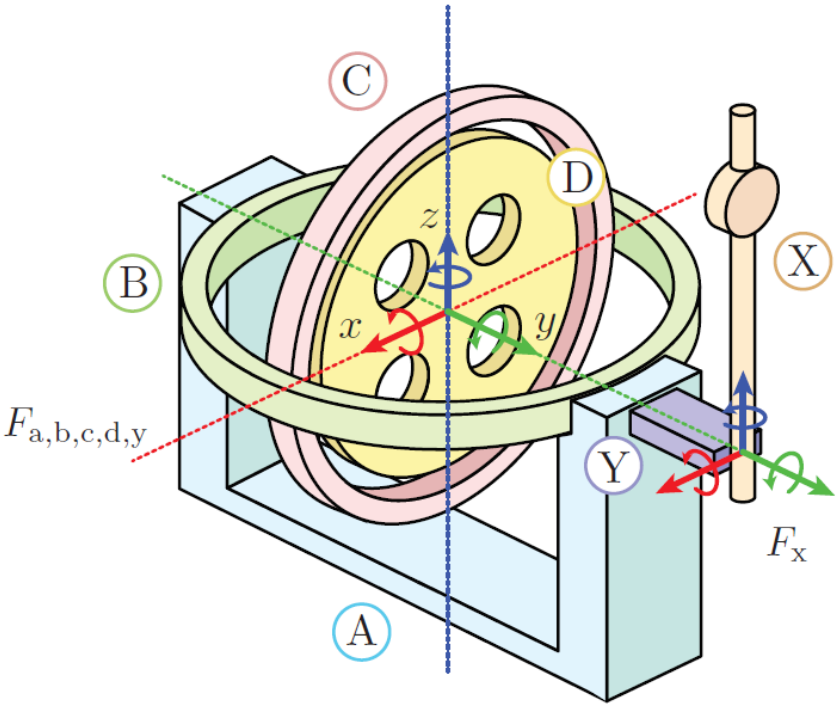


Figure 1: Schematics of the CMG with the inverted pendulum attachment

1.1 Plant Description

The plant is modelled in terms of bodies A, B, C, D, X and Y, see Fig. 1. Body D is a disk (flywheel) with rotation angle q_1 around the y-axis in frame F_d. This disk is actuated by a motor with an applied torque τ_1 aligned with the positive direction of q_1. Body C with a rotation angle q_2 around the x-axis in frame F_c is the gimbal encompassing the disk. Body C is also actuated in terms of a motor with generated torque τ_2 applied in the positive direction of q_2. Body B is the gimbal encompassing gimbal C which is

locked in the position shown in Fig. 1 (i.e. $q_3 = 0$) for this setup; this is necessary, since the encoder of the pendulum replaces the encoder of the body B. Body A is the gimbal encompassing body B with rotation q_4 in the positive direction around the z-axis in frame F_a. Body Y is the attached plate of the inverted pendulum and is modelled as an inertia attached to body A. The frames of references for bodies A, B, C, D and Y are all centred at the middle of body A, but attached to their respective body. Finally, body X is the inverted pendulum attached to body A with rotation q_x in the positive direction around the y-axis in frame F_x. Frame F_x is attached to the pendulum and centred at the rotation point of the pendulum (see Fig. 1). The angular velocities of the disk, gimbals and pendulum are denoted by ω_1, ω_2, ω_4 and ω_x respectively. The friction can be assumed to be viscous between all the different connected axis in the form of f_v, ω. All rotational angles are measured by incremental encoders.

1.2 Linearization and LPV Modeling

Due to complex dynamics of both the CMG and the inverted pendulum, Neweul-M^2 software package for matlab is used to model the system. The system is nonlinear, but it can be approximated by a Jacobian linearization about an operation point given by a fixed flywheel rotation speed $\dot{q}_{1,0}$ and fixed angular positions $q_{2,0}$ (recall that $q_3 = 0$ in this setup and as such not considered). This yields a collection of linear time invariant (LTI) state space model:

$$\begin{aligned} \dot{x} &= A(\dot{q}_{1,0}, q_{2,0})\ x + B(\dot{q}_{1,0}, q_{2,0})\ u\ , \\ y &= C(\dot{q}_{1,0}, q_{2,0})\ x\ . \end{aligned} \tag{1}$$

This is not an equivalent representation of the nonlinear system, because we are neglecting first and second order derivatives of $\dot{q}_{1,0}$ and $q_{2,0}$ in Taylor series expansion during Jacobian linearization that would result from a true "linearization about a time-varying operating point". What we have here is rather a continuous parameterization of a family of Jacobi linearizations, i .e. , for a given value of ρ , the LPV model coincides with the LTI model obtained from linearization at that operating point. Since we use states of the system to define this operating point, we are generating a *quasi*-LPV model.

Prep. 1.1: Make sure you understand why this is not an equivalent representation of the nonlinear plant and hence why we have no a priori guarantees that the controller

works on the nonlinear plant. Compare the approach to what is discussed in Expl. 1.1 of the *Advanced Topics in Control* lecture notes.

Here in the case of pendulum attached with the gyroscope, two different LPV models of the plant are considered; one used for the synthesis of a swing-up controller and one for the stabilizing controller. Because the states corresponding to the angles q_1 and q_2 do not influence the IO map, they can be truncated from the system.

Both LPV models are of the following form

$$\begin{aligned} \dot{x}(t) &= A(\rho(t))\, x(t) + B(\rho(t))\, u(t) \ , \\ y(t) &= Cx(t) \ , \end{aligned} \tag{2}$$

where in case of the stabilizing controller,

$$\begin{aligned} x_{st} &= \begin{bmatrix} q_4 & q_x & \omega_1 & \omega_2 & \omega_4 & \omega_x \end{bmatrix}^T \ , \\ u &= \begin{bmatrix} \tau_1 & \tau_2 \end{bmatrix}^T \ , \\ \rho_{st} &= \begin{bmatrix} q_2 & \omega_1 \end{bmatrix}^T \ , \\ C_{st} &= \begin{bmatrix} I_3 & 0 & 0 \\ 0 & 0 & I_2 \end{bmatrix} \ , \end{aligned} \tag{3}$$

and for the swing-up controller,

$$\begin{aligned} x_{sw} &= \begin{bmatrix} q_4 & \omega_1 & \omega_2 & \omega_4 \end{bmatrix}^T \ , \\ u &= \begin{bmatrix} \tau_1 & \tau_2 \end{bmatrix}^T \ , \\ \rho_{sw} &= \begin{bmatrix} q_2 & \omega_1 \end{bmatrix}^T \ , \\ C_{st} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \ . \end{aligned} \tag{4}$$

In order to use the synthesis techniques for gain-scheduled controllers that were discussed in class, we first need to define a grid representation of the parameter-space ($\mathcal{P}$) and the rate bounds ($\mathcal{V}$). In order to do this, we can use the following matlab commands:
In case of Swing-up controller,

```
q1dot   = pgrid(`q1dot',linspace(30,60,5),[-10 10]);
q2      = pgrid(`q2',linspace(-60*pi/180,60*pi/180,21), [-2 2]);
dom     = rgrid(q1dot, q2);
```

To load the linearized model into the matlab workspace you can use the provided function

```
G_ = lin_CMG_swing(q1dot.griddata,q2.griddata);
```

And in case of stabilizing controller,

```
q1dot   = pgrid(`q1dot',linspace(30,60,5),[-10 10]);
q2      = pgrid(`q2',linspace(-60*pi/180,60*pi/180,7), [-2 2]);
dom     = rgrid(q1dot, q2);
```

To load the linearized model into the matlab workspace you can use the provided function

```
G_ = lin_CMG_stab(q1dot.griddata,q2.griddata);
```

which contains a parameterization of the linearization at different operating points. We can then form a parameter-dependent state space object in both Swing-up and stabilizing case by calling

```
Gp = pss(G_,dom)
```

This object can be used very similar to the `ss` objects that you are familiar with. We can for example use

```
bodemag(Gp,b)
sigma(Gp,w)
```

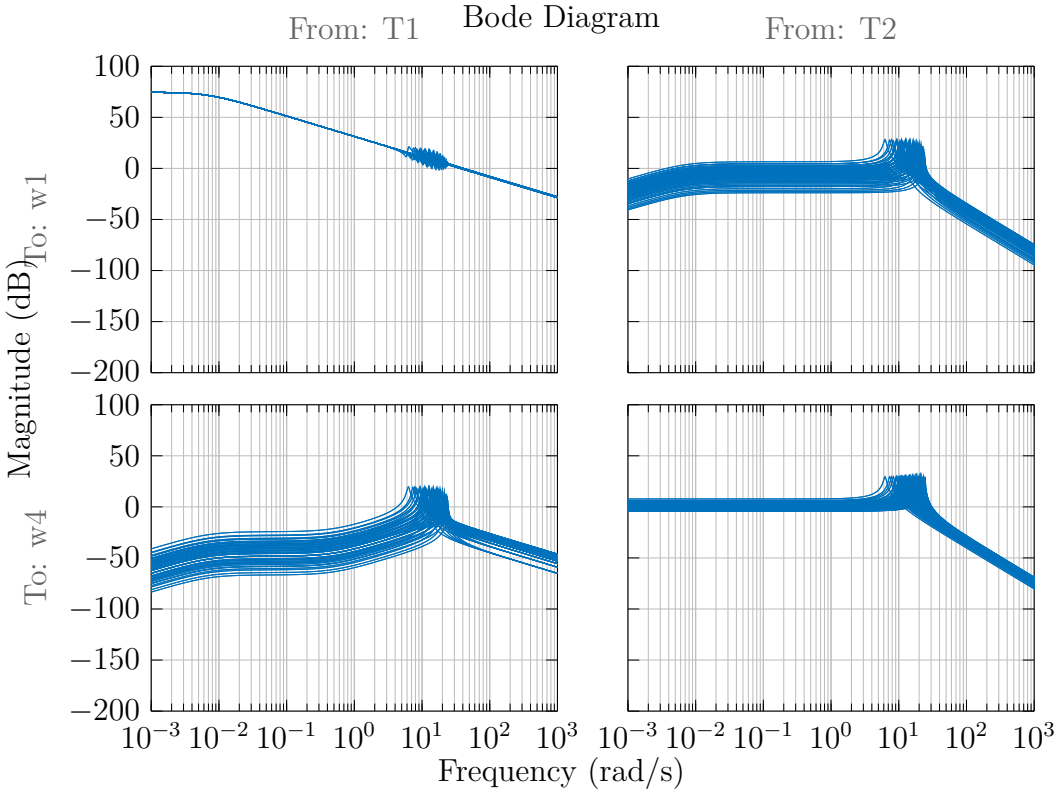
to bring up a bode magnitude plot and a sigma plot of the plant for each of the cases as shown in Figure 2 and Figure 3.

Prep. 1.2: From Figure 2 and Figure 3, are the parameter variations large? What appear to be the most dominant effects?

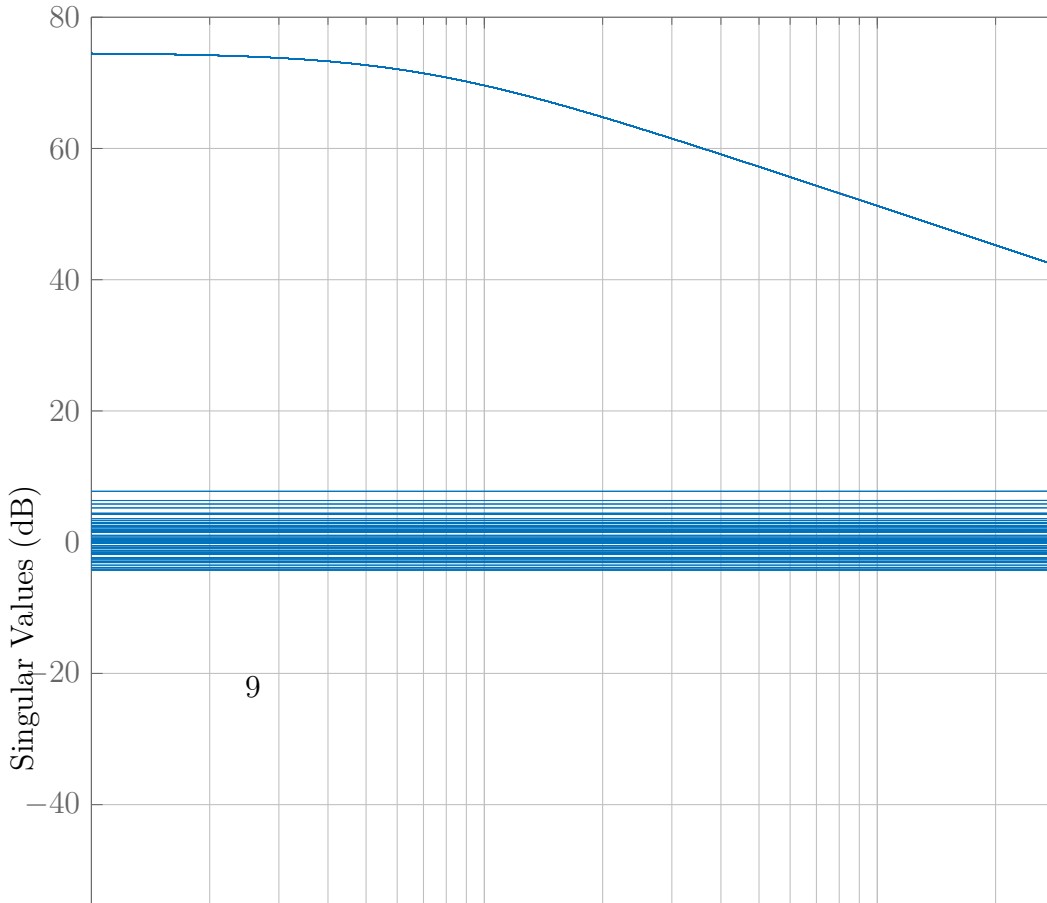
Note that we can access a variety of different properties of `pss` objects i. e. ,

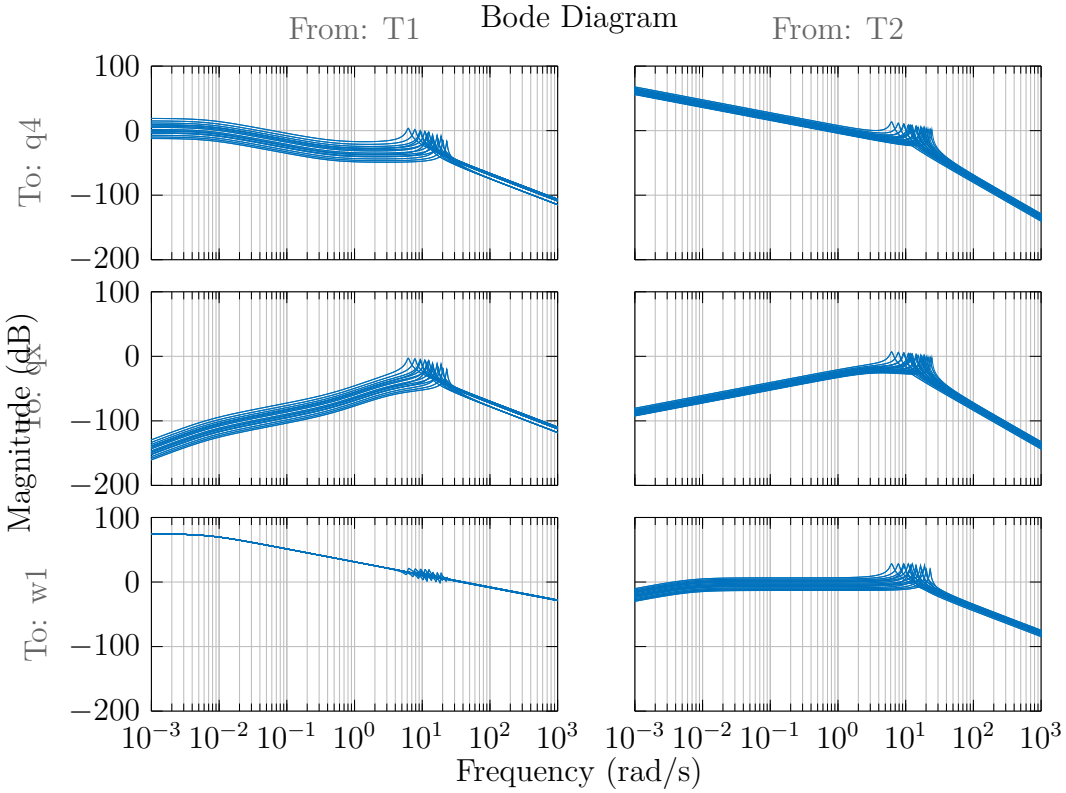
```
properties(Gp)     % shows available properties
Gp.Data            % returns underlying array of ss objects
Gp.Parameter       % returns scheduling parameters of the model
```

Prep. 1.3: Which representation of the frequency response, Bode plot or sigma plot, is more useful for design purposes? Why?

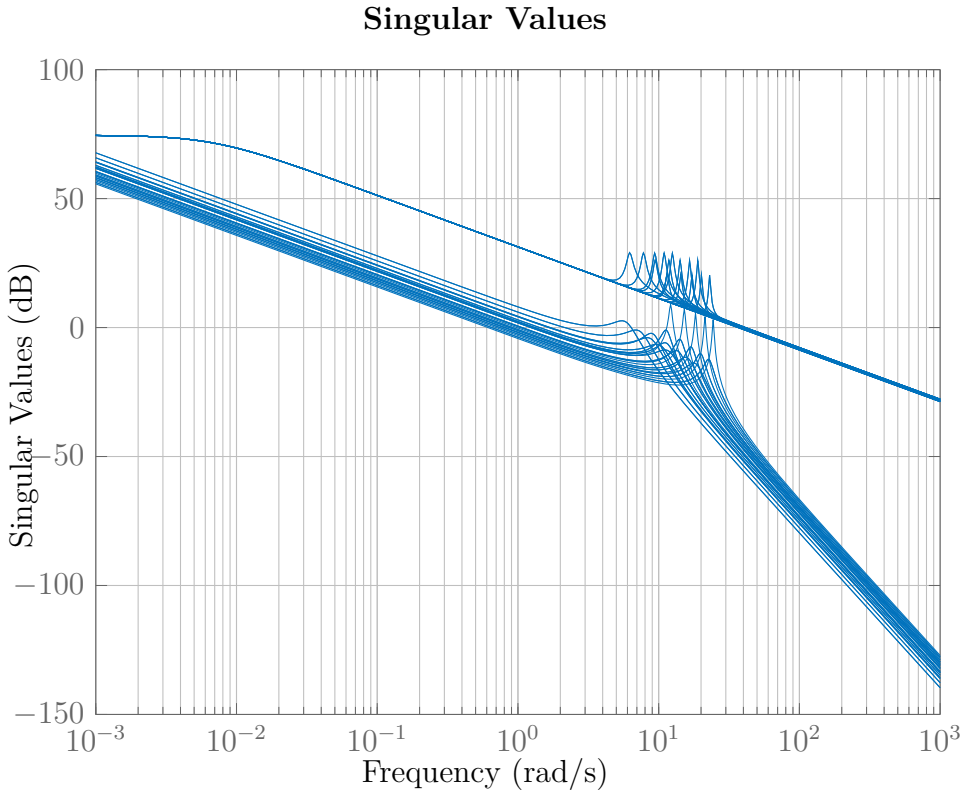


(a) Bode magnitude plot





(a) Bode magnitude plot



(b) Sigma plot

Figure 3: Stabilization: Frequency response

1.3 Initial Scaling

Scalings are of dire importance when working with frequency domain synthesis techniques. The following (very simple) scaling just normalizes the input and output values of the plant, such that the magnitude one corresponds both to the maximum expected change in the reference signal $D_{r,\max}$ and the maximum actuator capacity $D_{u,\max}$, i. e.

$$u = D_{u,\max}^{-1}\, u_{\text{real}} \tag{5}$$

$$y = D_{r,\max}^{-1}\, y_{\text{real}} \tag{6}$$

Just think of a rescaled input vector $B_s = B\, u_{\max}$ and a rescaled output vector $C_s = \frac{1}{r_{max}} C$ for the SISO case and keep in mind that since we are dealing with MIMO systems, we need to use diagonal matrices $D_{u,\max}$ and $D_{r,\max}$ instead. The usefulness of this scaling becomes apparent when we realize that a unit step reference now corresponds to the maximum input that we expect and that the available control signal is now also 1. Thus, if a unit step results in a control signal less than 1, the actuators are likely not to saturate. In terms of frequency domain indicators, this means that we can look at the transfer function $K\,S$ and try to achieve $\|K\,S\|_\infty < 1$. Once a controller for the synthesis model is designed, the scalings have to be reversed when the controller is applied to the original model:

$$u = K\, y \tag{7}$$

$$u_{real} = \underbrace{D_{u,\max}\, K\, D_{r,\max}^{-1}}_{K_{\text{real}}}\, y_{\text{real}} \tag{8}$$

Figure 4 illustrates the procedure.

In the case of Swing-up, reasonable assumptions on the largest allowed input signals is

$$\tau_{2,\max} = 2.44\,\text{Nm}\ ,$$

The outputs can be scaled for instance with

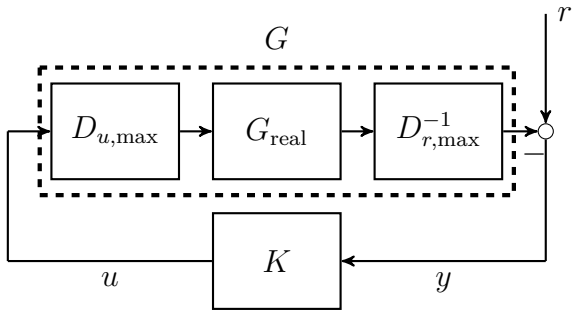
$$\dot{q}_{1,\max} = 45\ (\text{rad/s})\ , \quad \dot{q}_{4,\max} = 1.5\ (\text{rad/s})\ ,$$

And in the case of Stabilization, reasonable assumptions on the largest allowed input signals are (in Nm)

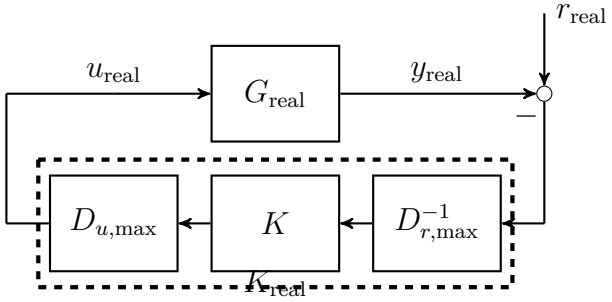
$$\tau_{1,\max} = 0.666 \text{ Nm}, \qquad \tau_{2,\max} = 2.44 \text{ Nm}$$

due to the different gearing mechanisms of the two motors. The outputs can be scaled for instance with

$$q_{4,\max} = 45\frac{\pi}{180} \text{ (rad)}, \quad q_{x,\max} = 10\frac{\pi}{180}\text{(rad)} , \quad \dot{q}_{1,\max} = 10 \text{ (rad/s)} .$$



(a) Scaling the actual plant to obtain a synthesis model



(b) Scaling the controller for implementation on the actual plant

Figure 4: Scaling for controller design based on frequency responses

2 CONTROLLER DESIGN—STATE FEEDBACK

2.1 Overview

The control structure used to achieve swing-up and stabilization consist of two separate control loops, one for swing-up and one for stabilization, in order to achieve better performance, especially for stabilization.

The control structure is as follows:

- Swing-up is achieved using a cascaded structure. The outer loop is based on an energy-based regulator C_e which computes the necessary angular acceleration $\dot{\omega}_4$ to swing-up the pendulum. This reference signal is integrated to compute a reference angular velocity ($\omega_{4,d}$) which is fed to the inner velocity LPV controller C_{sw} . An LPV controller is needed for this loop because these dynamics still heavily depend on the gimbal angle q_2 and the angular velocity of the disk ω_1.
- Near the unstable equilibrium position, $q_x = 0$, a switching logic S switches control authority to a stabilizing controller C_{st} whose task it is to stabilize the pendulum.

Linear switching strategies similar to the one used here have been used for a long time and have become a standard method for swing-up and stabilization of inverted pendulums. However, due to the complex nature of the CMG, a more advanced LPV extension of these control strategies is necessary for swing-up and stabilization. The controller structure implemented here can be seen in Fig. 5, where G represent the plant, S the switching logic, C_{st} the stabilizing LPV Controller, C_{sw} the swing-up controller, and C_e the energy based regulator.

Prep. 2.1: Make sure you understand the idea behind the cascaded structure. Why is it not a good idea to implement this using a single controller?

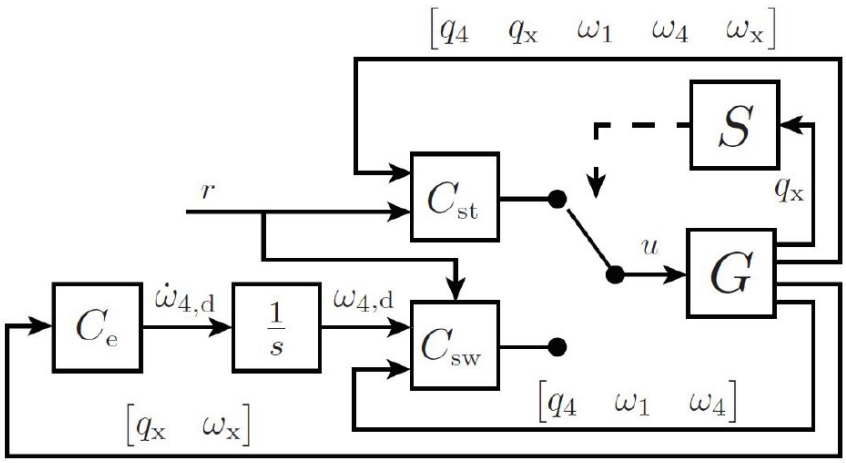


Figure 5: Control Structure

2.2 Energy Based Regulator

As mentioned, in order to do the swing-up of the inverted pendulum, a cascaded structure is used which consisting of an outer energy-based regulator $\mathrm{C_e}$ and an inner velocity LPV controller $\mathrm{C_{sw}}$. The energy-based regulator computes the required angular acceleration, $\dot{\omega}_4$, to increase the total energy of the pendulum subsystem (i. e. to swing up the pendulum), this signal is then integrated to obtain a reference for ω_4 for the swing-up LPV controller to track.

Using Lagrange equations of the second kind the equations of motion of the pendulum subsystem can be derived to be

$$\mathrm{M}_x\mathrm{L}^2\dot{\omega}_x - \mathrm{M}_x\mathrm{LR}\dot{\omega}_x \cos q_x - \mathrm{M}_x\mathrm{gL}\sin q_x = 0 \ , \tag{9}$$

where $\mathrm{M_x}$ denotes the mass of the pendulum, L the length of the pendulum arm, R the radius from the pendulum to the centre of the CMG, and g the gravitational acceleration. The total energy of the pendulum is given by

$$\mathrm{E} = \frac{1}{2}\mathrm{M}_x\mathrm{L}^2\omega_x^2 + \mathrm{M}_x\mathrm{gL}(\cos q_x - 1) \ , \tag{10}$$

differentiating with respect to time and using Eqn. 9 we get

$$\dot{\mathrm{E}} = \omega_x \mathrm{M}_x \mathrm{LR} \dot{\omega}_4 \cos q_x \ . \tag{11}$$

To control the energy, the Lyapunov function candidate

$$\mathrm{V} = \frac{1}{2}(\mathrm{E} - \mathrm{E}_0)^2 \tag{12}$$

is chosen, where E_0 is the total energy in the upright position. Using Eqn. 11, we can compute the time derivative of the Lyapunov function:

$$\dot{\mathrm{V}} = (\mathrm{E} - \mathrm{E}_0)\omega_x \mathrm{M}_x \mathrm{LR} \dot{\omega}_4 \cos q_x \tag{13}$$

which should be negative so that $\mathrm{V} \to 0$ and $\mathrm{E} \to \mathrm{E}_0$. Assuming we can control $\dot{\omega}_4$ and choosing it equal to

$$\dot{\omega}_4 = k_1(\mathrm{E} - \mathrm{E}_0)\omega_x \cos q_x \ , \tag{14}$$

where k_1 is a tuning parameter, Eqn. 13 becomes

$$\dot{\mathrm{V}} = k_1 \mathrm{M}_x \mathrm{LR}((\mathrm{E} - \mathrm{E}_0)\omega_x \cos q_x)^2 \ , \tag{15}$$

which is always negative for $k_1 < 0$. The energy in the upright position of the pendulum is equal to zero, therefore $\mathrm{E}_0 = 0$.
Due to limited bandwidth of C_{sw}, the output of the energy-based regulator is not tracked perfectly. The energy of the pendulum subsystem is, in reality, not exactly described by Eqn. 10. This causes the pendulum to not swing up completely using the described structure. In order to compensate for this discrepancy a tuning parameter k_2 was added to the calculation of the kinetic energy in Eqn. 10 resulting in

$$\mathrm{E} = k_2 \frac{1}{2} \mathrm{M}_x \mathrm{L}^2 \omega_x^2 + \mathrm{M}_x \mathrm{gL}(\cos q_x - 1) \ , \tag{16}$$

where k_2 (in combination with k_1) was then used to fine tune how fast and how far the pendulum would swing up, making sure it would not overshoot but also that it would get sufficiently close to the upright position. For the simulation studies, you can start with

tuning parameters as $k_1 = -2.5$ and $k_2 = 1.5$ and tune the values to have a satisfactory behaviour. The matlab function that implements the energy based regulator can be found in the 'Reference' - Swing up block in the simulink file.

Prep. 2.2: Make sure you understand the basic idea of energy-based regulator. Why were the tuning parameters k_1 and k_2 added. How does different values of k_1 and k_2 affect the system.

2.3 Swing-up Controller

In the previous control lab ATC 1, we used both state feedback control and output feedback control. However usually not all states of a system are available for feedback. Even if they can be estimated, it might not be desirable to use state feedback control. State feedback assumes perfect knowledge of all states, that is, the effects of measurement noise, uncertainty, etc. are inherently ignored. The correct way to address feedback problem is to consider output feedback with possible imperfect measurements. In this section, we will design a gain-scheduled output feedback controller to swing up the pendulum.

2.3.1 Four-Block Design

Swing-up LPV controller, C_{sw}, is designed using the four-block mixed-sensitivity loop shaping technique. In a four-block design, we seek to minimize the weighted norms of the transfer functions S, KS, SG and T_i, see *Exercise 17.2* in the '*Optimal and Robust Control*' lecture notes and the '*Control Lab — ORC 2*' documentation. In the Control Lab ORC 2 experiment, it becomes apparent that while disturbance rejection and robustness are generally good with this design approach, simultaneously achieving satisfactory tracking performance requires the use of what is called a two-degrees-of-freedom controller. Such a controller receives the available measurements and the reference as two independent signals (instead of a single error signal) and consequently decouples the tracking problem from other requirements such as disturbance rejection. We consider the same modification of the four-block problem that was used in the Control Labs ORC 2 and ATC 1.

The generalized plant for an output feedback problem is of the form

$$\left[\begin{array}{c} \dot{x} \\ \hline z \\ v \end{array}\right] = \left[\begin{array}{c|cc} A(\rho) & B_w(\rho) & B_u(\rho) \\ \hline C_z(\rho) & D_{zw}(\rho) & D_{zu}(\rho) \\ C_v(\rho) & D_{vw}(\rho) & D_{vu}(\rho) \end{array}\right] \left[\begin{array}{c} x \\ \hline w \\ u \end{array}\right], \tag{17}$$

where w to z is used as performance channel, v is the feedback signal consisting of a linear map of the states and external inputs, u the control input, x the state vector and ρ the scheduling variable.

The generalized plant that we are using is depicted in Fig. 6 and has three different performance inputs: r is the reference signal for y, d_i is an input disturbance acting on the plant and d_o can be thought of as measurement noise or disturbances on all available feedback signals. The performance outputs z_1 and z_2 reflect the design objectives and the available outputs are v_1 and v_2. The signal v_1 is used for feedback while v_2 is actually the reference signal and hence used in a feed-forward fashion.

Selecting the weights for this problem is not trivial and it is important to understand what we actually want to achieve. From the synthesis, we get a two-degrees-of-freedom controller, i. e. a controller with two independent inputs:

$$u = \begin{bmatrix} K_y & K_r \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} K_y & K_r \end{bmatrix} \begin{bmatrix} d_0 - y \\ r \end{bmatrix} \tag{18}$$

We can separate these two independent controllers and draw the resulting closed-loop system as in Fig. 7.

The transfer functions that appear in our design problem can thus seen to be

$$\begin{bmatrix} \tilde{z}_1 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} \mathrm{SG} & \mathrm{S} & \mathrm{I - SGK}_r \\ \mathrm{T}_i & \mathrm{K}_y\mathrm{S} & \mathrm{S}_i\mathrm{K}_r \end{bmatrix} \begin{bmatrix} \tilde{d}_i \\ \tilde{d}_o \\ r \end{bmatrix} \tag{19}$$

The transfer functions from the first two inputs, i. e. d_i and d_o form the classical four block design problem. We can verify, that the transfer function from r to y is SGK_r and hence the term $\mathrm{I - SGK}_r$ in Eqn. 19 represents the tracking error $(r - y)$. What we are mainly interested in are disturbance rejection (SG) and tracking ($\mathrm{I - SGK}_r$) for low frequencies and a roll-off in the controller ($\mathrm{K}_y\mathrm{S}$ and $\mathrm{S}_i\mathrm{K}_r$) for high frequencies. Thus, the

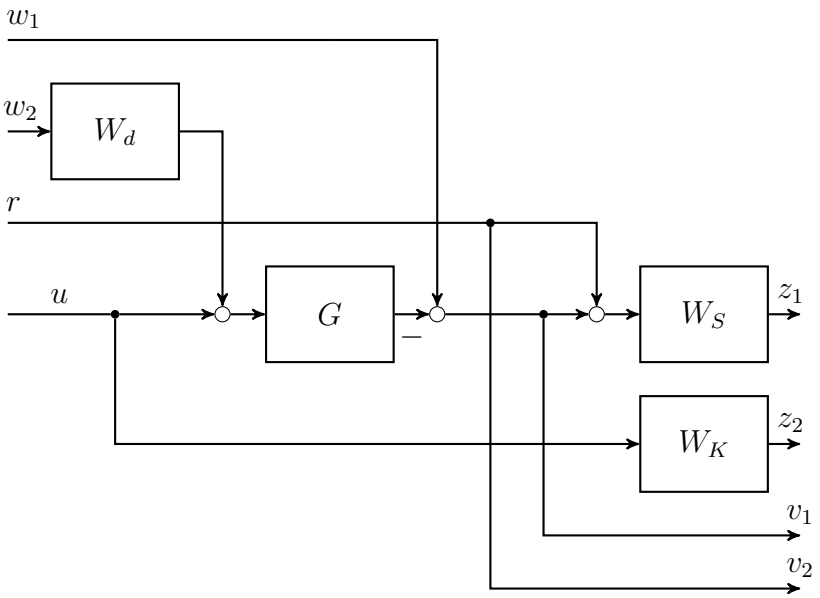


Figure 6: Open-loop generalized Plant P for the four block mixed sensitivity design with two degree of freedom

standard choice of $W_1 = \text{diag}(W_{\omega_1}, W_{\omega_2})$, as a low pass and $W_2 = \text{diag}(W_{\tau_1}, W_{\tau_2})$ as a high pass make also sense with this setup. The trade-off between tracking, disturbance rejection and control effort is handled via the input weights W_{d_i} and W_{d_o}.

Prep. 2.3: Understand the weighting structure and how to use it for tuning.

Prep. 2.4: Why is two-degree-of-freedom control helpful for tracking ?

In order to specify weights, we could use the parameterization from the ORC lecture (possibly with `pgrid` objects for parameter-dependent weights) to define weighting filters manually. A built-in command of the Robust Control Toolbox that we can use instead is

```
W_1 = makeweight(dcgain1, bandwidth1, feedthroughgain1)
W_2 = makeweight(dcgain2, bandwidth2, feedthroughgain2);
W   = mdiag(W_1,W_2);
```

where `dcgain` specifies the low frequency gain, `bandwidth` specifies the frequency at which the gain is one and `feedthroughgain` specifies the high frequency gain.
Designing filters using makeweight command, for example $S(j\omega)$, the area below 1 (0 db) and above $|\ S(j\omega)\ |$ is equal to the area above 1 and below $|\ S(j\omega)\ |$, when $|\ S(j\omega)\ |$ is plotted on a logarithmic scale. In simple words, the filter needs to cross the 0 db. For

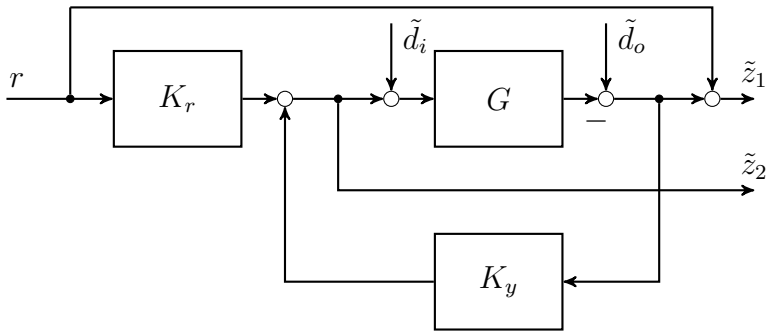


Figure 7: Open-loop generalized Plant P for the four block mixed sensitivity design with two degree of freedom

details see chapter 16 of ORC Lecture notes. However using the 'makeW' command, we have the flexibility to design first order shaping filter that is independent of this restriction.

```
W_1 = makeW(dcgain1, bandwidth1, feedthroughgain1, `h')
```

For our synthesis tools to work, we need to first assemble the generalized plant that includes the desired performance inputs and outputs. The generalized plant for the four-block formulation is depicted in Figure 6.
In order to assemble it, use the `sysic` command. You should look up the help in Matlab, but to get you started, consider the following code:

```
systemnames = `G WS WK Wdi Wdo';
inputvar = `[r(2);di(2);do(2);u(2)]';
outputvar = `[WS; WK; Wdo-G; r]';
input_to_G = `[u+Wdi]';
input_to_Wdi = `[di]';
input_to_Wdo = `[do]';
input_to_WS = `[r+Wdo-G]';
input_to_WK = `[u]';
cleanupsysic = `yes';
P = sysic;
```

2.3.2 Synthesis

Stability and the induced $\mathcal{L}_2$-norm constraints can be expressed as linear matrix inequalities (LMIs), see Theorem 3.2 of the *Advanced Topics in Control* lecture notes for LMI

condition regarding output feedback. Besides stabilizability of (A,B_u) and detectability of (A,C_v), we require that D_{zu} has full column rank and that D_{vw} has full row rank. If these conditions hold, the generalized plant can be transformed into:

$$\begin{bmatrix} \dot{x} \\ \hline z \\ v \end{bmatrix} = \left[\begin{array}{c|cc} A(\rho) & B_w(\rho) & B_u(\rho) \\ \hline C_z(\rho) & D_{zw}(\rho) & \begin{bmatrix} 0 \\ I \end{bmatrix} \\ C_v(\rho) & \begin{bmatrix} 0 & I \end{bmatrix} & 0 \end{array}\right] \begin{bmatrix} x \\ \hline w \\ u \end{bmatrix} , \tag{20}$$

We already defined the gridded parameter space and the rate bounds, so all that is left, is to define basis functions that represent the functional dependence of the candidate Lyapunov matrix P on the scheduling parameters, i. e. ,

$$P(\rho) = \sum f_i(\rho)\, P_i \; . \tag{21}$$

Prep. 2.5: Why do we need to assign basis functions? What are the decision variables when solving the optimization problem?

We can do this in Matlab by using

```
Pbasis = [];
Pbasis(1) = basis(1,0);
```

The first argument of `basis` is the function and the second argument is the partial derivative. Thus, the example above defines a parameter-independent Lyapunov matrix $P = P_1$ and the partial derivative of this constant is zero. If we want to add parameter-dependent terms, the `pgrid` objects can be used:

```
Pbasis(2) = basis(q1dot, `q1dot', 1);
Pbasis(3) = basis(q2, `q2', 1);
```

As such the Lyapunov matrix is $P = P_1 + P_2\ \dot{q}_1 + P_3\ q_2$. Other arbitrary complex expression can be added too. However, we need to pay close attention to defining the partial derivatives correctly. How to choose a "good" set of basis functions is to a large extend still not clear, see also exercise 3.2 in the Advanced Topics in Control lecture notes. After defining basis function for two candidate Lyapunov matrices X and Y, the controller can be obtained by the command

```
[K,gam,INFO] = lpvsyn(P, nmeas, ncont, Xbasis, Ybasis);
```

Depending on the dynamic order of the system, the number of basis functions for the Lyapunov matrices and especially the number of grid points, this synthesis may take very long . Tuning the controller can thus become a really time-consuming task, and as we all know: time is money. Thus, it is usually a better idea to use either parameter independent Lyapunov matrices while trying to find weights or to simply perform a number of LTI synthesis steps with a common weighting structure. The main conceptual difference is the following: The first approach gives an upper bound during tuning, since the use of a common constant Lyapunov matrix is conservative. Our responses will usually look better when using the actual parameter-dependent synthesis once a satisfactory tuning was found. The second approach, on the other hand, gives a lower bound. We cannot exceed the performance of an LTI controller at a single grid point, even if we used infinitely many basis functions for the Lyapunov matrices. Our results will thus usually look worse with the LPV controller and we can add basis functions in order to try to make the deterioration small.

Transformation of the generalized plant into the form represented by Eqn.20, as well as balancing and scaling to reduce sensitivity to numerical issues are automatically performed within the LPVTools' 'lpvsyn' command. Further, just as in regular H∞ control it is usually desirable to use a suboptimal, rather a truly optimal synthesis. Since LPV synthesis is even more prone to numerical issues than the H∞ synthesis a suboptimal synthesis is invoked. The default suboptimality factor is 1.2, i. e. , 20%. For further insight on how the synthesis tool work, see Chapter 3 of *Control Lab -ATC 1 Manual*.

Prep. 2.6: Experiment with different choices of basis functions. How do they affect synthesis complexity and achievable performance?

2.3.3 Controller Post-Processing

If a parameter-dependent Lyapunov matrix X is used, the controller also depends on scheduling rates. Often, this dependence can simply be ignored, although again, any theoretical guarantees are lost in doing so. For the swing-up, we would require $\dot{\omega}_1$ to be available online. It's your decision whether you want to implement the rate dependence with differentiation filters or ignore it. In order to neglect it, we can interpolate the controller for zero rates:

```
K = lpvelimiv(lpvinterp(K_rate, `q1dotDot', 0));
```

2.3.4 Design Task

Design a gain-scheduled output feedback controller to achieve the following design specifications:

- make sure the bandwidth of the velocity controller should be 5-10 times faster than the natural frequency of the pendulum
- at least 2dB gain and 14.5 degree phase multiloop disk margin at every grid point
- that disturbances lead to outputs of less than 3 in magnitude at every grid point
- that disturbances are compensated after 3 seconds on all channels at every grid point
- that the control input magnitude never exceeds 1 at any time
- a controller with a fastest pole < 2000 rad/s when evaluated at a grid point.

2.4 Stabilizing Controller

2.4.1 Four-Block Design with Velocity Feedback

Just like the Swing-up controller, C_{sw}, the stabilizing controller, C_{st}, is also designed using the four-block mixed sensitivity loop shaping techniques using the two-degrees-of-freedom approach. Again the main motivation behind it is to simultaneously achieve satisfactory tracking performance along with required disturbance rejection and robustness. As such again, at low frequency we aim to achieve disturbance rejection and tracking while a roll-off at high frequencies. Thus, a suitable choice would be $W_1 = \text{diag}(\ W_{q_4}, W_{q_x}, W_{\omega_1})$ as a low pass filter and $W_2 = \text{diag}(W_{\tau_1}, W_{\tau_2})$ as high pass filter.
Moreover, in addition to the outputs q_4, q_x,and ω_1, we will also consider angular velocities ω_4 and ω_x as available feedback signals. In practice, we still need to estimate these by implementing differentiation filters, which cause phase loss and amplifies measurement noise. We neglect the phase loss that is introduced by these filters in our model and rely on the robustness of the controller to deal with it . Experiments show that with these extra measurements, the controller is able to increase damping much better in the closed loop response than with just the angles. However it must be noted that these signals are not part of the performance channel and are therefore not considered for the tracking objective. Thus, r is a vector with 3 entries while y is a vector with 5 entries. The error $(y - r)$ thus is not well defined unless we are more precise and write (in Matlab notation) $r - y(1:3)$. We can add the additional output to the plant using

```
Gv = pss(G.a, G.b, [G.c;zeros(2,4) eye(2)], [G.d;zeros(2,size(G.b, 2))]);
```

Similar to the swing-up controller, in order for the synthesis tools to work we will again need to assemble the generalized plant using the `sysic` command. The generalized plant for the four block design is depicted in Figure 6. Afterwards we will need to define the basis function. Once both the generalized plant and the basis function are defined, we can solve the state feedback gain using the command 'lpvsfsyn'.

2.4.2 Switching Logic

In order to stabilize the pendulum in the upright position, control authority needs to switch from the cascaded loop (using the energy-based regulator and LPV velocity controller) to the stabilizing LPV controller. To accomplish this, several different switching logics can be implemented which smoothly switches the system to the stabilizing

controller when the angle of the pendulum is close to zero. The angle at which the switch happens, $\mid q_x \mid < 0.15$ rad, was found by heuristically tuning the switching angle until a desired response was achieved.

Smooth transition can be achieved by using a ramp weighting of the output of both controllers. This can be done by taking a convex combination of control input of both the swing-up and the stabilizing LPV controller during the transition, i. e. :

$$u = \alpha u_{sw} + (1 - \alpha) u_{st} \ ,$$

where u_{sw} denotes the control input generated by $\mathrm{C_{sw}}$, u_{st} the control input generated by $\mathrm{C_{st}}$ and α is a ramp from 1 to 0 with slope 10 when switching from swing-up to stabilization and a ramp from 1 to 0 when switching from stabilization to swing-up. However, experimentally it was found that implementing even a hard switch gave satisfactory results in which case when the absolute value of angle q_x is greater than 0.15 rad, Swing-up controller is activated else the Stabilizing controller is activated. To avoid wind-up effects in the controller states, the integrators are reset when the controller is switched off, e.g. when switching from the swing-up LPV controller to the stabilizing LPV controller the integrators in the swing-up controller are reset.

2.4.3 Design Task

Design a gain-scheduled state feedback controller to achieve the following design specifications:

- a controller with a fastest pole < 2000 rad/s when evaluated at a grid point
- the input and output disturbances should be attenuated within 5 seconds on all channels at every grid point
- that disturbances lead to outputs of less than 5 in magnitude at every grid point
- that the control input magnitude never exceeds 1 at any time
- at least 0.5 dB gain and 3.5 degree phase multiloop disk margin at every grid point

3 LINEAR EVALUATION OVER THE GRID

We can only assess whether our controller performs well in nonlinear simulation and on the actual experiment. Nevertheless, linear analysis for individual grid points can be very helpful during the design. If the linear responses are not satisfactory, for sure, the nonlinear ones won't be either.

3.1 Frequency Response Analysis

A first step in evaluating our controller can be a frequency response analysis. We can calculate all the six different transfer functions of interest using `loopsens`. The structure `loops` then contains all closed-loop transfer functions which are accessible as

```
loops = loopsens(G,K);
lpvgetfield(loops,'So')      % S
lpvgetfield(loops,'Si')      % Si
lpvgetfield(loops,'To')      % T
lpvgetfield(loops,'Ti')      % Ti
lpvgetfield(loops,'CSo')     % KS
lpvgetfield(loops,'PSi')     % SG
lpvgetfield(loops,'Stable') % verifies that the closed loop is stable
```

Matlab uses a different convention than we do and labels the plant with P instead of our usual G, and the controller with C instead of K. Sticking to our notation, the transfer functions that we obtain are in the order of appearance $S = (I + G\,K)^{-1}$, $S_i = (I + K\,G)^{-1}$, $T = I - S$, $T_i = I - S_i$ as well as $K\,S$ and $G\,S_i$, which are the same as $S_i\,K$ and $S\,G$, respectively. Since we are considering nonsquare plants for the output feedback case, make sure that you look at the correct input/output pairs. For example, when looking at step responses, we are interested in just y and not $\dot{y}$, so only the first three channels are of interest.

3.2 Time Domain Analysis

You can perform step response simulations on the linear models at all grid points simultaneously with the `step` command.

3.3 Robustness

Linear robustness margins cannot guarantee anything for LPV systems. Nevertheless, the same argument as for the step responses remains true: the parameter varying nature will just make things worse and hence linear margins at a grid point are a first indicator whether a design can succeed. Just as in *Control Lab—ORC 2 and ATC 1*, we only consider the multiloop disk margin and are further just interested in a worst case analysis, i. e. the lowest margins across the grid. Note that this is a VERY conservative measure, so we can't expect too much. Still, try to aim for something like 3dB gain and 15 degree phase margin.

```
[~,~,~,~,~,~,MMIO]= loopmargin(Gv,Ky);

GM = lpvgetfield(MMIO,'GainMargin');
PM = lpvgetfield(MMIO,'PhaseMargin');
lpvmin(GM(2))
minGM = db(lpvmin(GM(2)))
minPM = min(lpvmin(PM(2)))
```

Note that the analysis for the state feedback controller would require to break the loops for every *state* in addition to the outputs. You can try this on your own if you like.

4 NONLINEAR SIMULATION AND IMPLEMENTATION

Once you complete and run `design_CMG_swing.m` file, save the data from workspace using the following command in Command Window:

```
save(`LPVswing.mat')
```

Also do the same after completing and running `design_CMG_stab.m` file using the command

```
save(`LPVstab.mat')
```

4.1 Simulation Environment

The file `simCMG_full.slx` provides a nonlinear simulation environment resembling the control moment gyroscope. The structure of the model is depicted in Figure 8.

The Red Block represents the control moment gyroscope and contains the equations of motion implemented as a mex-file/C-code. It takes the control signal $u = [\tau_1 \quad \tau_2]^T$ as an input (in Nm). The outputs of the block are the physical available output $y = [q_2 \quad q_4 \quad q_x]^T$ [rad], the estimated velocities $\dot{y} = [\dot{q}_1 \quad \dot{q}_2 \quad \dot{q}_4 \quad \dot{q}_x]^T$ [rad/s] used for feedback, the saturated (i. e. actually applied) inputs [Nm] and a signal bus that contains all states of the simulation model.

The Orange Block represents the controllers. It also contains the reverse scalings that we used during the design.

The Blue Block is the signal generator for the reference trajectory. The reference block generate reference signal for both swing-up controller, $\dot{q}_{1,ref}$, $\dot{q}_{4,ref}$, and Stabilizing controller, $\dot{q}_{1,ref}$, $q_{4,ref}$, $q_{x,ref}$.

The Green blocks are associated with the user interface.

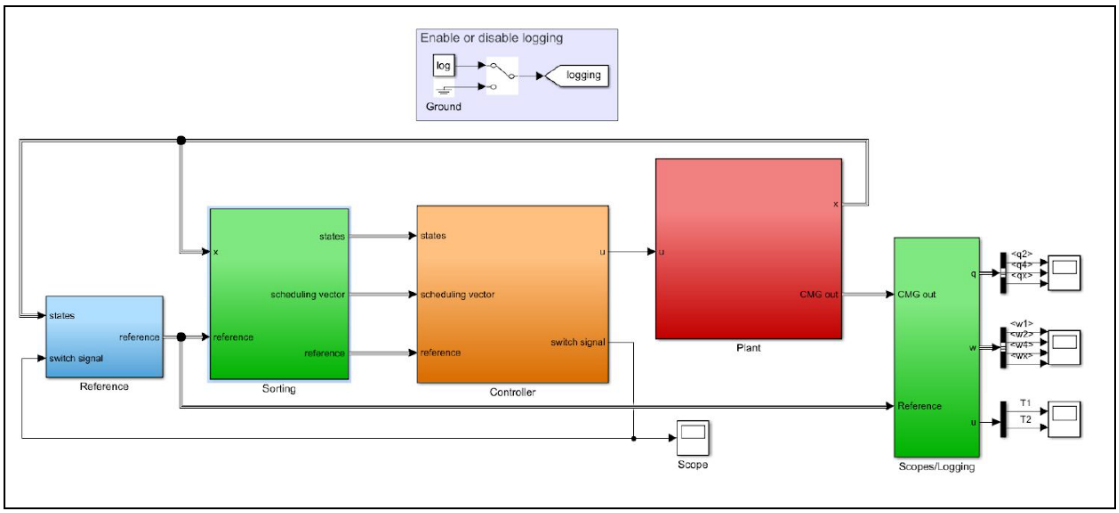


Figure 8: Simulink Simulation: Plant, Controller and interface-related blocks

Prep. 4.1: Familiarize yourself with the simulation. You should be able to explain the purpose of every single block.

4.2 Lookup Table Implementation

The default method of implementing gridded LPV controllers is in terms of lookup tables with element-wise linear interpolation. Simulink provides a block called `n-D Lookup Table` for this purpose.

We need to specify the number of dimensions, the table data and breakpoints. An example implementation for the state feedback gain dependent on the two parameters $\dot{q}_1$, and q_2, and looks as follows:

```
Table data:   reshape(F.Data,[size(F,1)*size(F,2), size(F.Domain)])
Breakpoints 1:  1:size(F,1)*size(F,2)
Breakpoints 2:  F.Parameter.q1dot.GridData
Breakpoints 3:  F.Parameter.q2.GridData
```

As inputs, a vector containing the linear indices `1:size(F,1)*size(F,2)` and the two parameters are required. The output is a vector containing all entries of the matrix F. It can be transformed into the correct dimensions using the `reshape` block.

A dynamic controller is implemented analogously: The three state space matrices A, B, and C can all be stored as lookup tables in the form described above and then the dynamic system is formed by including an integrator of compatible dimension.

Prep. 4.2: Verify the implementation in the Simulink model. You can use the `display` block or `scopes` to actually see the varying entries in the matrices during simulation.

Alternatively, a complete nonlinear implementation is possible that calculates the controller directly from the Lyapunov matrices in every sampling instant, see Exercise 3.2 of the *Advanced Topics in Control* lecture notes. In this case, memory requirement is usually lower, since only the coefficients of the Lyapunov matrices need to be stored. Computational complexity is on the other hand larger, since several matrix multiplications and inversions need to be performed. While this is in general more accurate and computationally indeed tractable, the implementation requires extreme care to include all scalings and transformations performed during synthesis. As these quantities are not accessible when using commercial tools such as the LPVTools, we will not address this possibility further.

4.3 Goals

To run the simulation (simScriptCMG.m), load the swing up and stabilizing controllers into the workspace and achieve the following tasks:

- Tune the values of k_1 and k_2 such that a balance is achieved between how fast and how far the pendulum would swing-up
- Swing up should be achieved within 30 seconds meaning $q_x = 0$ or 2π.
- q_4 should be tracked to the desired angle
- ω_1 should be tracked to 45 rad/sec and ω_4 to the speed specified by the energy regulator

5 EXPERIMENT

During the lab, the file `full_model.slx` will be provided to interface the control moment gyroscope. Its structure is depicted in Figure 9. As you can see, it is very similar to the nonlinear simulation.

The blue block represents the control moment gyroscope and contains interfaces to the Digital/Analog and Analog/Digital converters of the physical plant. The inputs and outputs of the block are the same as in simulation except that differentiation filters are used to estimate the states.

The orange block are identical to that used in the simulation.

The green blocks are identical to that used in the simulation except that the output structure is now named `expdata`. Code for plotting this data will also be provided.

The yellow blocks are associated with a dedicated "start-up" controller and the switching logic necessary to switch to the LPV controller. The start-up controller is used to bring the plant to a certain operating point. It consists of a simple PI controller acting on $\dot{q}_1$ and q_2. The first yellow switch on the upper left is used to activate/deactivate the start-up controller. The second can be used to reset q_3 and q_4 in order to define an operating point.

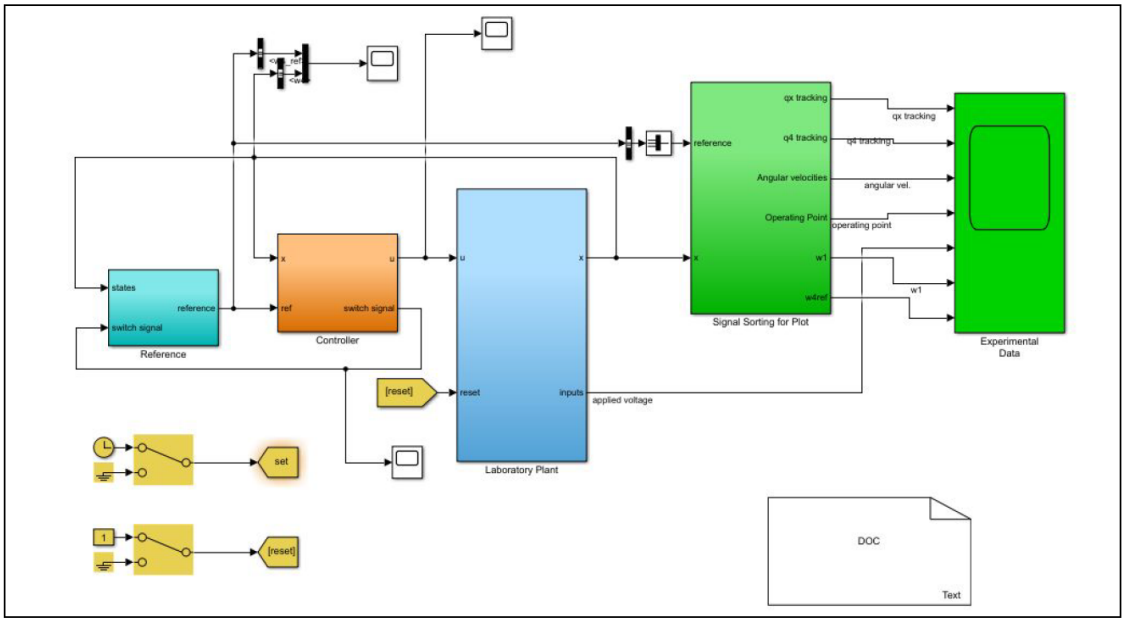


Figure 9: Simulink model: Plant, Controller, Start-Up Controller and interface-related blocks

Prep. 5.1: Make sure your design file produces all the files necessary to run the nonlinear simulation by simply executing it.

! Bring this files with you to the experiment as you may need to retune your controller.

Goal

Implement your controller in the Simulink real-time interface and try it out on the actual plant. You should be able to achieve

- Swing up within 30 seconds
- That the pendulum should stabilize at the upright position rather then over swing and reject any disturbances which can cause it to deviate from the said position.
- cross coupling should not exceed 2.5° per 45° command (i. e. should be less than 6%)
- that the control input magnitude never exceeds 10 V at any time
- that the control input is smooth and does not result in mechanical wear (you'll hear what we mean if you fail to achieve this).

If necessary, retune your controller and try again.

! Be careful and obey all safety requirements when working on the experimental device. An unstable controller can cause fast and unpredictable motion of the gimbals which can cause severe injuries.