NLR 2 / P2.2: PREPARATION TASK - SYNCHRONOUS MACHINE

Notes:

- Realizing the exact linearization of the example system via MATLAB Symbolic toolbox gives the necessary introduction to the software specifications that are needed for the actual laboratory work.
- Furthermore, the basic principle of exact input/output linearization shall be demonstrated for the MIMO case. The obtained functionalities shall then be utilized and extended for the lab work.
- Useful short cuts of MATLAB Live Script: *Ctrl+Alt+G* - Latex Input interface; *Alt+Enter* - Switch between math and text mode; *Ctrl+Enter* - Run section; *Ctrl+Alt+Enter* - Insert section break
- Open calculations are marked with *"[ ]"* and have to be filled by you. Additionally, use the preparation tasks for orientation.
- Feel free to choose an alternative way or approach if the suggested strategy in the script does not comfort you.

```matlab
% Setup:
rootpath = 'C:\...'; % INSERT data path here!
cd(rootpath)
addpath functions models % include auxilary functions & simulation models
set(0,'defaulttextinterpreter','latex') % Latex style labels
```

Exemplary MIMO exact input-output linearization for synchronous machine

System model

The motor can be described by the following nonlinear system of differential equations:

$$\begin{aligned}
\dot{x}_1 &= -x_1 + x_3\cos(x_4) + u_1\\
\dot{x}_2 &= -x_2 + x_3\sin(x_4) + u_2\\
\dot{x}_3 &= -x_3 + x_1\cos(x_4) - x_2\sin(x_4)\\
\dot{x}_4 &= x_3
\end{aligned}$$

with the d-q current states x_1, x_2 (*Park-Transformation*), the angular velocity x_3 and the respective angle x_4. Consider the following output relation:

$$\begin{aligned}
y_1 &= x_1\sin(x_4) + x_2\cos(x_4)\\
y_2 &= x_4
\end{aligned}$$

(2.1) *Bring the system into an input-affine form for further calculations!*

```matlab
% Define symbolic variables:
syms x1 x2 x3 x4 real
syms u1 u2 real
z = [x1 x2 x3 x4]';
```

```
u = [u1 u2]';
% Input-affine system form:
f = []
g = []
h = []
```

Thus, an input-affine form $\dot{x} = f(x) + g(x)u$ with output $y = h(x)$ has been formulated.

Relative degree and decoupling

In order to determine the relative degree automatically, the Lie Derivative has to be implemented. It is defined in the following itaritve manner:

$$L_f^0 h(x) = h(x)\,, \quad L_f h(x) = \frac{\partial h}{\partial x} f$$

$$L_f^i h(x) = L_f L_f^{i-1} h(x)$$

Then, the respective relative degree can be determined step by step by computing the decoupling matrix $\alpha(x)$:

$$\alpha(x) = \begin{bmatrix} L_g L_f^{r_1-1} h_1(x) \\ L_g L_f^{r_2-1} h_2(x) \end{bmatrix}$$

(2.2) *Determine the vector relative degree as well as the sum relative degree and examine the exact I/O linearizability! In order to compute symbolic Lie derivatives* $L_f^i h(x)$*, formulate a MATLAB function* `lie_derivative(f,h,x,i)`*.*

```
% First output derivatives: y1 = h1(x)
alpha11 = simplify(lie_derivative(g,lie_derivative(f,h(1),z,0),z,1))
...
% Second output derivatives: y2 = h2(x)
alpha21 = simplify(lie_derivative(g,lie_derivative(f,h(2),z,0),z,1))
...
```

Supposingly, the vector relative degree $r = [1, 3]$ is obtained with a full sum vector relative degree of 4. This ensures that there is no internal dynamics in the transformed system. For the exact I/O linearizability the respective decoupling matrix must be proven to be non-singular.

(2.3) *Bring the system into a form which directly relates the inputs* u *to derivatives of the output* y*!*

$$\begin{pmatrix} y_1^{(r_1)} \\ y_2^{(r_2)} \end{pmatrix} = \begin{pmatrix} L_f^{r_1} h_1(x) \\ L_f^{r_2} h_2(x) \end{pmatrix} + \begin{bmatrix} L_g L_f^{r_1-1} h_1(x) \\ L_g L_f^{r_2-1} h_2(x) \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

```
% Decoupling matrix:
alpha = []
simplify(det(alpha))
% Input-output relation:
```

```
beta = []
```

The relation $y^{(r)} = \beta(x) + \alpha(x)u$ has to be established for further calculations.

Linearizing transformation

Applying the input transformation

$$u = \alpha(x)^{-1}(w - \beta(x)),$$

a new linear system in block Brunovsky form

$$\begin{aligned}\dot{z} &= Az + Bw\\ y &= Cz\end{aligned}$$

can be obtained in combination with the state transformation

$$z = \Phi(x) = \begin{pmatrix} h_1 \\ \vdots \\ h_1^{(r_1-1)} \\ h_2 \\ \vdots \\ h_2^{(r_2-1)} \end{pmatrix}$$

(2.4) *Determine the exact form of the transformed linear system (A,B,C), state the flat state transformation and calculate its inverse mapping!*

```
% Control law:
syms w1 w2 real % reference inputs
w = [w1 w2]';
uctrl = [] ;
% Flat state transformation:
Y = []
% Linear system block form:
A = []
B = []
C = []
% Inverse transformation:
syms z1 z2 z3 z4 real
Z = [z1 z2 z3 z4]';
zsol = solve( [] ,z);
Zsol = [zsol.x1 zsol.x2 zsol.x3 zsol.x4]'
```

For simulation purposes, the obtained structures are transfered to Matlab functions in order to be used in a Simulink model. Keep the order within the transformation in mind as it affects the linear control design in the z domain.

```
% Saving important functions for implementation:
matlabFunction(uctrl,'File','functions/flat_control','Vars',[z;w]);
matlabFunction(Y,'File','functions/transformation','Vars',z);
% Saving important functions for simulation:
matlabFunction(f+g*u,'File','functions/system_dynamics','Vars',[z;u]);
```

System implementation

Now, the control system can be implemented utilizing a linear control design methodology for the system with flat states.

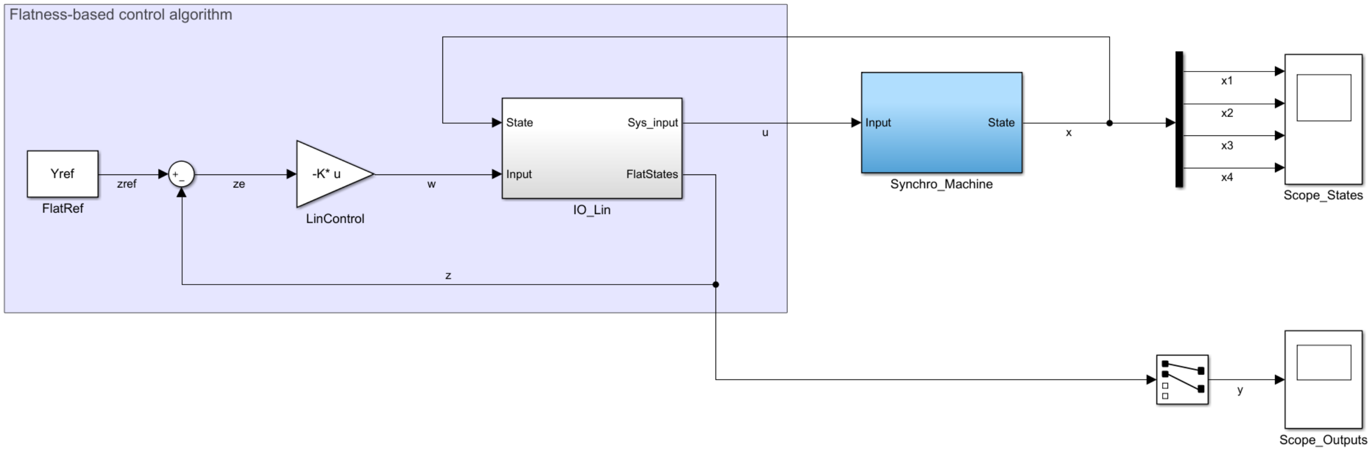
(2.5) *Design a state feedback controller for the linearized system by utilizing pole placement!*

Take advantage of the block Brunovsky form and treat each input-output block channel for separate coefficient placement with respect to the desired poles (controllability normal form).

```
% Smart pole placement:
K = []
A + B*K
eig(A + B*K) % check eigenvalues
```

(2.6) *Implement the system as a Simulink model and verify your specifications by simulative tests!*

The closed-loop system is simulated in MATLAB Simulink with the following simulation model structure:



```
% Simulation parameters & reference:
Tf = 10; % end time
ref = [0.1 -0.3]'; % output reference
Yref = [];
x0 = [0 0 0 0]'; % initial simulation conditions
% Execute Simulink model:
sim('synchro_mdl')
time = StateData.time;
xsim = [StateData.signals(1).values StateData.signals(2).values ...
        StateData.signals(3).values StateData.signals(4).values];
ysim = OutData.signals.values;
% Evaluated results:
% [please insert some nice plots here]
```

[Discussion]

Extended concepts

Especially for the flatness based reference tracking control problem, additional design prepration for the feedforward control and reference error stabilization has to be made.

***(2.7)** Realize a function for computing polyonomial trajectories in order to implement operating point transitions!*

In this case, the flat output reference tracking control problem becomes:

$$\begin{pmatrix} y_1^{(r_1)} \\ y_2^{(r_2)} \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} := \underbrace{\begin{pmatrix} {y_1^*}^{(r_1)} \\ {y_2^*}^{(r_2)} \end{pmatrix}}_{\text{feedforward}} + K \cdot \underbrace{\begin{bmatrix} y_1^* - y_1 \\ \vdots \\ (y_1^* - y_1)^{(r_1-1)} \\ y_2^* - y_2 \\ \vdots \\ (y_2^* - y_2)^{(r_2-1)} \end{bmatrix}}_{\text{generalized PD}} = w_{FF} + K\,\Delta z$$

with the gain K that stabilized the error dynamics asymptotically. Then, a polynomial shaped reference trajectory between two operating points can be constructed like this:

$$\begin{aligned} & y_f^*(t) = (y_{f,1} - y_{f,0})\,p(t) + y_{f,0} \quad \text{with} \quad p(0) = 0,\, p(T) = 1 \\ \Rightarrow \quad & y_f^{*(i)}(t) = (y_{f,1} - y_{f,0})\,p^{(i)}(t) \quad \text{with} \quad p^{(i)}(0) = 0,\, p^{(i)}(T) = 0\,\forall i \in \{1, \ldots, ?\} \end{aligned}$$

$$p(t) = \sum_{k=0}^{2r+1} c_k \left(\frac{t}{T}\right)^k \quad \text{with} \quad c_k \text{ s.t. BCs}$$

where $y_{f,0}$ is the starting and $y_{f,1}$ the final operating point at time T. A polynomial interpolation function shape is chosen for easier computation.

```
% Symbolic function generation:
syms t T positive % time argument & time horizon
y0 = sym('y0', [2 1]); % initial OP
yF = sym('yF', [2 1]); % target OP
r = []; % required derivatives -> max(vector relative degree)

... % [generate polynomials]

wFF = [];
zdes = [];
% Save & test functionality:
matlabFunction(wFF,zdes,'File','functions/feedforward','Vars',{t,T,y0,yF});
% Test:
time = 0:1e-2:Tf;
[wtest,ztraj] = feedforward(time,Tf,ref,[0;0]);
subplot(2,1,1); plot(time,wtest','LineWidth',1.5); grid
```

```
title('Feedforward control signal'); ylabel('$w_{FF}(t)$')
subplot(2,1,2); plot(time,ztraj','LineWidth',1.5); grid
title('Flat state reference trajectory'); ylabel('$z^\ast(t)$')
xlabel('transition time $t\in[0,T_f]$')
```

(2.8) *Repeat and examine important additional MATLAB functionalities!*

```
% (a) Matrix arrangement:
% help kron
% help blkdiag
N = 0;
A = magic(3);
B = [3 1 2]'; % =2222 in base 5
C = [1 4 6]; % =222 in base 8
AA = []
BB = []
% (b) Optimal control:
% help lqr
Q = [];
R = [];
K = []
disp(eig(AA-BB*K))
```