Numerical Analysis, Simulation and Control 2

M.Sc. Laboratory Advanced Control (WS 21/22)

Prof. Dr.–Ing. habil. Thomas Meurer, Chair of Automatic Control

The goal of this exercise is to learn the usage of the software package MATLAB/SIMULINK and its application in control engineering systems. Thereby MATLAB and the simulation environment SIMULINK are applied for system analysis and simulation of dynamic systems. All tasks in this exercise are solved with this software package. All SIMULINK model and simulation files must be stored compatible to the MATLAB/SIMULINK version *R2017b*.

For an introduction to MATLAB/SIMULINK, it is recommended to use the program help and further literature, such as [1].

If you have any questions or suggestions regarding this laboratory exercise, please contact

- Manuel Amersdorfer (maam@tf.uni-kiel.de).

2.1 Software Package MATLAB

MATLAB is an interactive program for scientific and technical calculations. The name MATLAB, derived from MATrix LABoratory, expresses that the special strength of this program lies in vector and matrix calculation.

2.1.1 MATLAB-Desktop

The MATLAB-desktop is the user interface of the program. It is individually adaptable and contains the following windows by default

- **Command Window**
  It represents the input area, which must be displayed in any case. Here all commands following the >> prompt can be entered and executed directly. If you close a command sequence with a semicolon, the output of MATLAB is suppressed. Pressing the Enter key causes the command line to be executed immediately. In the case of multi-line command entries, the Shift and Enter keys can be used to jump to a new line.

- **Editor**
  In the editor, functions, scripts or program code (e.g. C code) can be created and edited. The possibilities for step-by-step execution of command sequences, debugging, etc., which are common in programming environments, are provided. Scripts are also called M-files. They have the file extension `*.m` and are processed by an interpreter at runtime. Scripts contain MATLAB

command sequences, such as those entered directly in the *Command Window*. Functions also have the File extension `*.m`. They usually receive input values and return values.

- **Current Directory Browser**
  The *Current Directory Browser* displays the current working directory. Files and folders can be opened, created and edited. It is recommended that all files accessed during the calculation or simulation are located in a common local (current) directory.

- **Workspace Browser**
  The current variables are displayed in MATLAB in the so-called workspace. They can be accessed and modified by clicking on them.

2.1.2 Basic Commands

Download the zip-archive `u2.zip` from the course homepage in *OLAT*. This zip archive contains examples of various MATLAB commands in the files

- `Matlab_Intros/matlab_intro_part1.m`: Basic commands;
- `Matlab_Intros/matlab_intro_part2.m`: Graphical display of results;
- `Matlab_Intros/matlab_intro_part3.m`: Examples for the Control System Toolbox;
- `Matlab_Intros/mean_value.m`: Example of a function (for calculating the mean value of two numbers).

Exercise 1.1. *`Matlab_Intros/matlab_intro_part1.m` contains important basic* MATLAB *commands are shown by means of examples. Open this file in the* MATLAB*-editor and work through all commands step by step. Try to understand all the commands, and if necessary use the help function. To execute individual command sequences, select them in the editor and press F9. To execute an entire M-File, either enter its name (without file extension) in the* Command Window *or open the file in the Editor and press F5. Note that the function `Matlab_Intros/mean_value.m` is called. Keep in mind that this file must be located in the current* MATLAB *working directory or search path.*

Exercise 1.2. *Open the files `Matlab_Intros/matlab_intro_part2.m` and `Matlab_Intros/matlab_intro_part3.m` in the* MATLAB*-editor and work through all commands step by step. Try to understand all commands and use the help function if necessary. A good source is also the* MATLAB *documentation on* `https://de.mathworks.com/help/matlab/index.html`.

Exercise 1.3. *Consider the linear system of equations*

$$\begin{aligned} 2x_1 + x_2 - x_3 &= 1 \\ 7x_1 + 5x_2 + 2x_3 &= -4 \\ 13x_1 - 8x_2 + 5x_3 &= 15. \end{aligned} \tag{2.1}$$

Enter this system of equations in matrix representation and calculate the solution vector $\boldsymbol{x} = [x_1, x_2, x_3]^T$ with x_1, x_2 and x_3 in fraction representation. To solve the system of equations, make first use of the `inv()` command and secondly consider with the `mldivide()` command (or in its short form `\`). How do these commands differ and when should which be used?

<u>Hint:</u> The output format of the solutions can be set in MATLAB *with the command `format`.*

To extend the functionality of the basic program MATLAB, extension packages – so-called toolboxes such as SIMULINK – are offered. These toolboxes comprise functions and environments that provide a suitable tool for solving the problems that arise in different application areas.

An overview of the installed toolboxes can be obtained e.g., from the menu MATLAB Start → Toolboxes, which is located at the bottom left of the screen.

2.2 Simulink

SIMULINK is an extension of MATLAB that provides a graphical user interface for modeling, simulation and analysis of dynamic systems using signal flow graphs. SIMULINK models have the file extension `*.slx`. On the one hand, SIMULINK provides a library with pre-built function blocks and on the other hand, user-defined blocks can be created, in particular, so-called *Simulink MATLAB Function* blocks provide a flexible to achieve this. *Simulink MATLAB Function*s are programmed directly as MATLAB code and they can call arbitrary functions or scripts from M-files. Alternatively, user-defined functions can be defined as *Simulink S-Functions*, that be also programmed as a MATLAB M-file or in C. These allow the implementation of dynamic models in one block.

The SIMULINK command opens the *Simulink Library Browser*, which displays the pre-built function blocks. These can be moved into the SIMULINK model by drag & drop. The blocks are to be connected by signal flow lines. Blocks that are required frequently are in the structure of the *Simulink Library Browser*, e.g., in the following groups:

- `Continuous`: Blocks for the simulation of time continuous systems, among others the time continuous integrator `1/s`;
- `Math Operations`: Some mathematical operations, e.g., addition, multiplication, squaring;
- `Sinks`: Blocks that have only one input (or several inputs) but no output. For example, `Scopes` are used for the graphical representation of signals. Signals can be tapped at any position directly from signal flow lines. The `To Workspace` block allows the export of simulation results to the MATLAB workspace, where they are then stored as a variable for further evaluation;
- `Sources`: Blocks, which have only one output (or several outputs) but no input, such as signal generators.

The following procedure is recommended when creating a SIMULINK model:

1. It is advantageous to not enter all parameter values directly in SIMULINK but to define them together in an M-file, which is executed before the simulation. This way the parameters can be changed easily and centrally. To update the parameters, the M-file must be executed again. All variables in the MATLAB workspace are also available in SIMULINK;
2. If the mathematical expressions become more extensive, it is recommended to simplify the interconnection by using user-defined (programmed) blocks to avoid many single blocks. The corresponding blocks are located in the `User-Defined Functions` group. Both MATLAB functions and user-defined MATLAB functions can be used in the block `Fcn`. For dynamic systems the block `Simulink MATLAB Function` is suitable;
3. Another way to improve the clarity of models is the usage of subsystems (`Ports & Subsystems` → `Subsystem`);

4. To reduce the number of signal flow lines displayed on the screen, the blocks `From` and `Goto` from the `Signal Routing` group can be utilized.

Simulation settings in SIMULINK are defined in the menu *Simulation → Configuration Parameters*. Important here is the selection of the simulation duration and the integration algorithm. Furthermore, limits of the time step size and accuracy requirements can be set for the integration algorithm when numerically solving ordinary differential equations. In the context of this introduction, a detailed description of the used algorithms is omitted. An overview of some of the solvers for initial value problems available in MATLAB can be found in the help file for the functions `ode23`, `ode45`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb` (callable e.g. with `doc ode45`) under the *Algorithms* heading. Furthermore, it is recommended to take into account further literature such as [1].

The simulation can be started by clicking the start button ▶ or by pressing the Ctrl and T keys. Alternatively, the command `sim()` is be used to start a simulation from the input window or an M-file.

The implementation of a dynamic system, for which a model exists in the form of an (explicit) differential equation, can be realized either as a block diagram or as a user-defined MATLAB function. However, for more extensive systems to be simulated, the clarity of the models implemented in the form of block diagrams can be lost quickly, which is why the use of *Simulink MATLAB Functions* is preferable for such systems. In the context of this exercise, only so-called *Simulink MATLAB Functions* are used.

In the following, the two possibilities for the simulation of a dynamic system under SIMULINK, as block diagram and as *Simulink MATLAB Function*, will be briefly explained.

2.2.1 Implementation of Dynamic Systems as Block Diagram

Exercise 1.4. *Download the zip-archive `u2.zip` from OLAT and extract from the folder `pt2/` the files `set_parameter_pt2.m` and `simulink_testfile_pt2.slx`. Execute the file `set_parameter_pt2.m` to set the parameters and start the simulation of the model `simulink_testfile_pt2.slx`. The model contains the implementation of a simple PT_2-element in the form of block diagrams to realize the transfer function*

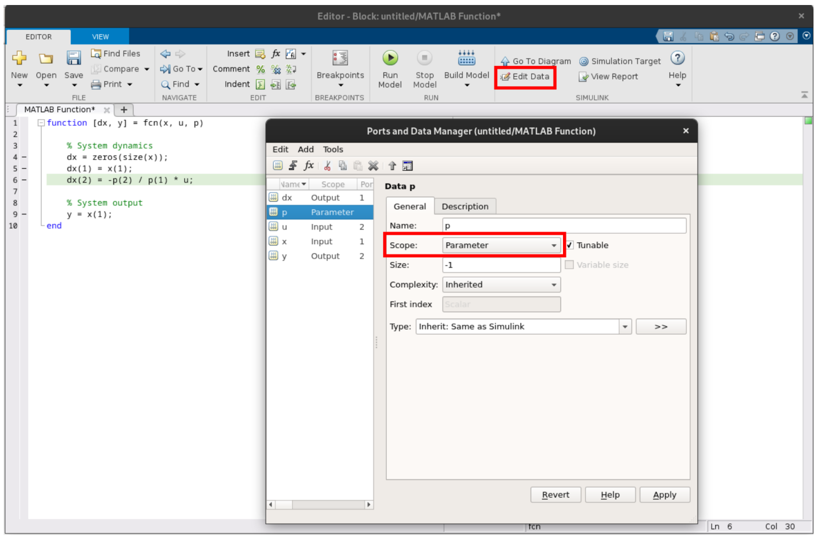
$$\hat{g}(s) = \frac{V}{1 + 2\xi Ts + (sT)^2}. \tag{2.2}$$

Try to understand the functionality of all blocks and use the help function if necessary.

Hint: For linear time-invariant systems, a simpler implementation is possible with the help of the Control System Toolbox. Here, an LTI system can be transferred directly in the form of a transfer function (e.g. (2.2)) or a state space representation with the block `LTI System` from the `Control System Toolbox` group.

2.2.2 Implementation of Dynamic Systems as *Simulink MATLAB Function*

In the following, the example of the tower crane from Section 1.5 in the previous laboratory exercise is used to show how a dynamic system can be simulated using an *Simulink MATLAB Function*.



```
function [dx, y] = fcn(x, u, p)

    % System dynamics
    dx = zeros(size(x));
    dx(1) = x(1);
    dx(2) = -p(2) / p(1) * u;

    % System output
    y = x(1);
end
```

Name	Scope	Port
dx	Output	1
p	Parameter	
u	Input	2
x	Input	1
y	Output	2

Data p

Name: p
Scope: Parameter ✓ Tunable
Size: -1 Variable size
Complexity: Inherited
First index Scalar
Type: Inherit: Same as Simulink

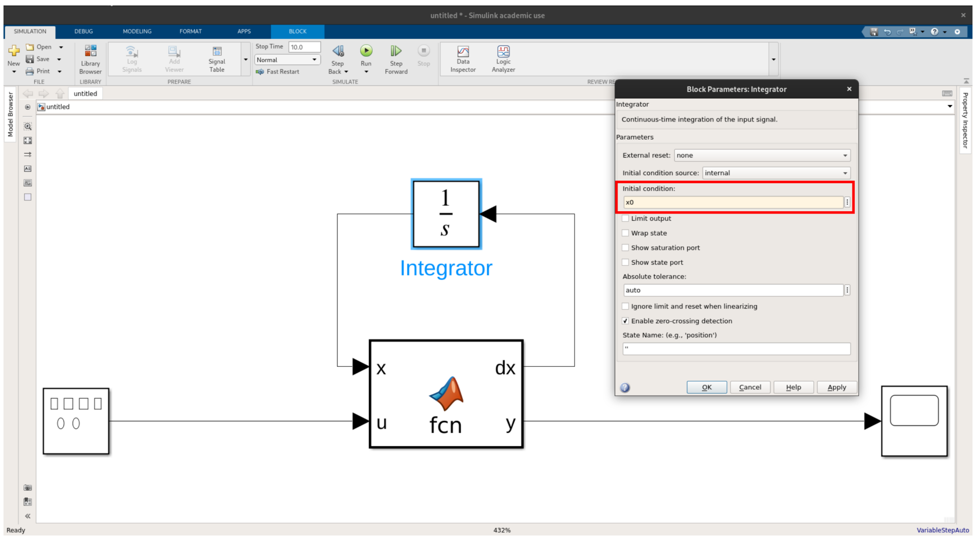
Figure 2.1: Opening the *Ports and Data Manager* from the editor and assigning the scope of the function arguments.

Exercise 1.5. *Download the zip-archive* `u2.zip` *from OLAT and extract the* MATLAB *script file* `tdk/tdk_sim_ini.m` *together with the simulation model* `tdk/tdk_sim.slx`*. Work through the* Simulink MATLAB Function *and execute the simulation. Try to understand the structure of the* Simulink MATLAB Function *on the basis of the following description and make use of the help function if necessary.*

Label the scopes with the appropriate names of the states.

Simulink MATLAB Functions provide a way to execute MATLAB code directly in Simulink. It provides a simple way to handle input and output signals as well as the configuration parameters of the block. Therefore, they are considered as function arguments in the comprising MATLAB code. In contrast to *Simulink S-Functions*, they require an additional integrator block for the definition of continuous-time systems or a delay block for discrete-time systems. Nevertheless, its usage is considered simpler and more intuitive which can speed up the development process.

When the *Simulink MATLAB Function* block is opened with a double click, Simulink automatically opens the editor window with the corresponding MATLAB code. It consists of a function whose transfer arguments consist of the variables for the values of the input signals and configuration parameters, e.g. physical parameters of the system. The return values of the function define the values of the output signals of the block. The type of the arguments can be assigned with the attribute *Scope* in the *Ports and Data Manager*, which is opened from the editor with the *Edit Data* button according to Fig. 2.1. There, also the size of the input and output ports is defined. Input and output signals can have scalar, vector or matrix-sized values and even `struct` in the case of bus ports, while parameters can have any MATLAB data type. By default, all function transfer arguments are considered as input ports of the block and all return values as output ports, and their size is determined automatically. The parameters have to be defined as variables in the MATLAB workspace or SIMULINK model workspace before the start of the simulation. Otherwise, the simulation is aborted after compiling the model and an error message is displayed. The same holds if the size of the signals in Simulink does not comply with the size of the ports.



Block Parameters: Integrator

Integrator

Continuous-time integration of the input signal.

Parameters

External reset: none

Initial condition source: internal

Initial condition: x0

☐ Limit output

☐ Wrap state

☐ Show saturation port

☐ Show state port

Absolute tolerance: auto

☐ Ignore limit and reset when linearizing

✓ Enable zero-crossing detection

State Name: (e.g., 'position') ''

Figure 2.2: Setting the initial condition of the dynamic system in the integrator block parameter configuration dialog.

The function body defines the processing of the input and parameter variables and the assignment of the return variables. For a continuous-time dynamical system, typically the derivatives are calculated, meaning the differential equation is expressed in the form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{2.3}$$

with the input variables for the system states $\boldsymbol{x}$ and system inputs $\boldsymbol{u}$, and the return value $\dot{\boldsymbol{x}}$. Additionally, the system output signal can be defined by calculating the output equation

$$\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}). \tag{2.4}$$

with the return value $\boldsymbol{y}$.

Outside the *Simulink MATLAB Function* an integrator has to be used to integrate the signal of the system state derivative output port `dx` of the block and feed the result back into the system state input port `x` of the *Simulink MATLAB Function* block. The initial state of the dynamical system is set as the initial condition: in the block parameter configuration dialog of the integrator, as shown in Fig. 2.2. The dialog can be opened by double-clicking on the integrator block.

Take into account the following notes in connection with *Simulink MATLAB Functions*:

- To include *Simulink MATLAB Functions* in SIMULINK models, use the block `User-Defined Functions` → `MATLAB Function`;
- The *Simulink MATLAB Function* can call any self defined matlab function and all standard MATLAB functions, but the may be some limitations on MATLAB toolbox functions (consider the MATLAB reference documentation for further details);
- For the implementation of discrete-time systems, a delay block with one step delay has to be used instead of the integrator.

Exercise 1.6. *Create a* SIMULINK *model for the double pendulum system according to Section 1.6, which simulates the nonlinear system model*

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, u), \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{2.5a}$$

$$\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, u) \tag{2.5b}$$

with $\boldsymbol{x} = [y\ \dot{y}\ \varphi_1\ \dot{\varphi}_1\ \varphi_2\ \dot{\varphi}_2]^T$, $\boldsymbol{y} = \boldsymbol{x}$ *and* $u(t) = \ddot{y}(t)$ *with a* Simulink MATLAB Function. *For the implementation, you can use the template file `dip/dip_student_sim_simulink.slx`. For your implementation in* MATLAB / SIMULINK, *follow the nomenclature of Table 2.1. Pass the initial states* $\boldsymbol{x}_0$ *to the integrator to the corresponding* Simulink MATLAB Function *as a parameter vector. Define these parameters in an M-file, which you execute before the start of the simulation. Note that the order of the entries of the initial states* $\boldsymbol{x}_0$ *corresponds to the order of the states* $\boldsymbol{x}$. *Select the initial states* $\boldsymbol{x}_0$ *and the input variable u and simulate the double pendulum.*

Self-check: For testing and adjusting your model, the files `dip_student_sim.slx` and `dip_sfun.p` in the folder `dip/` can be downloaded from the zip-archive `u2.zip` from the OLAT *course. Thereby `dip_sfun.p` is an encrypted* Simulink S-Function *of the double pendulum, that requires to have the initial states* $\boldsymbol{x}_0$ *passed from outside the function. The integration of this* Simulink MATLAB Function *takes place analogously to the* Simulink MATLAB Function *and is shown in the simulation model `dip_student_sim.slx`.*

Table 2.1: Nomenclature for MATLAB / SIMULINK.

Description	Symbol	MATLAB / SIMULINK
System states		
Car position	y	`y`
Car velocity	$\dot{y}$	`yp`
Angle of the pendulum arm 1	φ_1	`phi1`
Angular velocity of the pendulum arm 1	$\dot{\varphi}_1$	`phi1p`
Angle of the pendulum arm 2	φ_2	`phi2`
Angular velocity of the pendulum arm 2	$\dot{\varphi}_2$	`phi2p`
Parameters for pendulum arm 1		
Length	l_1	`l1`
Distance between center of mass and pivot point	l_{S1}	`l_S1`
Mass	m_{G1}	`m_G1`
Moment of inertia	J_{G1xx}	`J_G1xx`
Friction constant	k_{G1}	`k_G1`
Parameters for pendulum arm 2		
Length	l_2	`l2`
Distance between center of mass and pivot point	l_{S2}	`l_S2`
Mass	m_{G2}	`m_G2`
Moment of inertia	J_{G2xx}	`J_G2xx`
Friction constant	k_{G2}	`k_G2`
General parameters		
Gravitational constant	g	`g`

2.3 The Continuous-Time Extended Kalman Filter

To realize a state feedback control for a dynamic and fully controllable system, the state vector $\boldsymbol{x}(t)$ of the system must be known for all times $t \geq 0$. If not all state variables are metrologically available, but the considered system is (locally) completely observable, the unknown system states can be reconstructed from the evolution of the input and output variables on the basis of a state space model with the help of a state observer such as a Luenberger observer.

State observers like the Luenberger observer are suitable for deterministically disturbed systems. If a system is continuously affected by stochastic perturbations, so-called state filters, are better suited to estimate the states.

A frequently used state filter or state estimator, respectively, for linear systems, is the Kalman filter [4, 2, 3, 5], which exists in a time-discrete and a time-continuous version[1]. The Extended Kalman Filter (EKF) is an extension of the time-continuous Kalman filter to nonlinear systems with the state representation

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{w}, \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0 \,, \tag{2.6a}$$

$$\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{v} \,, \tag{2.6b}$$

where the additive disturbances $\boldsymbol{w}$ and $\boldsymbol{v}$ describe the process noise and the measurement noise, respectively. These disturbance variables are assumed to be mean-free, white, uncorrelated noise signals with Gaussian amplitude distribution, where the covariance matrix of the system noise is described by the matrix Q and the covariance matrix of the measurement noise by the matrix R.

The state equation for the estimated state $\hat{\boldsymbol{x}}(t)$ reads

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{f}(\hat{\boldsymbol{x}}, \boldsymbol{u}) + L(\boldsymbol{y} - \boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{u})) \,, \qquad \hat{\boldsymbol{x}}(0) = \hat{\boldsymbol{x}}_0 \tag{2.7}$$

with the feedback matrix L referring to Kalman's time-variant amplification matrix. This matrix is calculated using the equation

$$L = PH^{\mathsf{T}}R^{-1}. \tag{2.8}$$

Here, P is the covariance matrix of the state error

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \hat{\boldsymbol{x}} \tag{2.9}$$

that fulfills the matrix Riccati differential equation

$$\dot{P} = FP + PF^{\mathsf{T}} + Q - PH^{\mathsf{T}}R^{-1}HP \,, \qquad P(0) = P_0 \tag{2.10}$$

with F and H representing the Jacobian matrices

$$F = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}, \boldsymbol{u}} \tag{2.11}$$

and

$$H = \left.\frac{\partial \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}, \boldsymbol{u}} . \tag{2.12}$$

[1] To design the Kalman filter for an LTI system in MATLAB, the Control System Toolbox provides the `kalman` command.

Table 2.2: Parameter combinations for the measurement noise signal covariance σ^2, process covariance matrix $Q = \sigma_Q^2 I$ and measurement covariance matrix $R = \sigma_R^2 I$.

Idx.	σ^2	σ_Q^2	σ_R^2
1	10^{-3}	10^{-5}	10^{-3}
2	10^{-3}	10^{-5}	10^{-2}
3	10^{-3}	10^{-5}	10^{-4}
4	10^{-3}	10^{-3}	10^{-3}
5	10^{-2}	10^{-5}	10^{-3}
6	10^{-4}	10^{-5}	10^{-3}

Exercise 1.7. *For the double pendulum system according to Section 1.6, implement a continuous-time Extended Kalman Filter as `Simulink MATLAB Function`, allowing to pass $\hat{\mathbf{x}}_0$, P_0, Q, and R as external parameters.*

Hint: For the implementation of the state equation (2.7) *and the matrix Riccati differential equation* (2.10) *in the* Simulink MATLAB Function, *it is recommended to construct a common state vector. This vector consists of the vector $\mathbf{x}$ (in column representation) and the elements of the matrix P. To construct a column vector from a matrix `A` by stringing together its columns, the command `A(:)` can be used.*

Exercise 1.8. *Simulate the double pendulum together with the Extended Kalman Filter in the template Simulink file `dip_ekf_sim_student.slx` and init script `dip_sim_ekf_ini.m` in the folder `dip_EKF/`. Select different initial states for the system and the Extended Kalman Filter. Also try different values for the car acceleration $u = 0.1\ m/s^2$ and the measurements y, φ_1 and φ_2. To simulate an additive stochastic disturbance of the measurement, superimpose the measurement signals with a random signal with the covariance $\sigma^2 = 10^{-3}$. Use the* SIMULINK *block `Random Number` with sampling time of* 10 ms. *Make sure that the disturbance signal is statistically independent for each measurement.*

Try different combinations for the process and measurement covariance matrices as well as for the measurement signal noise covariance according to Tab. 2.2. Compare the estimated states with the states of the original system and write down your observations of the simulation results.

References

[1] A. Angermann, M. Beuschel, M. Rau, and U. Wohlfarth. *Matlab – Simulink – Stateflow. Grundlagen, Toolboxen, Beispiele.* Oldenbourg, 2005 (cit. on pp. 1, 4).

[2] A. Gelb. *Applied Optimal Estimation.* The M.I.T. Press, 1974 (cit. on p. 8).

[3] Mohinder S. Grewal and Angus P. Andrews. *Kalman Filtering: Theory and Practice with MATLAB.* 4th. Wiley-IEEE Press, 2014 (cit. on p. 8).

[4] R. E. Kalman. „A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1 (1960), pp. 35–45 (cit. on p. 8).

[5] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches.* USA: Wiley-Interscience, 2006 (cit. on p. 8).