Experimental part I: Control of the asynchronous machine

The aim of this experimental part is to model, synthesize and simulate the field-oriented control of an asynchronous machine.

1. Introduction

This experimental part focuses on the control of the asynchronous machine with squirrel-cage rotor. In the following, the basics for field-oriented control are presented. This includes the coordinate transformation, the mathematical description of the asynchronous machine and the discretization of the plant model. Subsequently, the field-oriented control and the design of the controllers are explained.
The execution of the experiment is divided into the following sections:

1. Analytical design of the current controller
   The system model of the asynchronous machine in the Laplace domain is determined and the current controller is designed on the basis of this model. The controlled plant is analyzed with regard to dynamics.

2. Continuous controller implementation in Matlab/Simulink
   Current and speed controllers are implemented in Matlab/Simulink and the results of the previous part of the task are checked.

3. Controller discretization
   The controllers are discretized and discretely analyzed and examined.

4. Controller programming in several S-Functions
   The time-discrete plants and controllers are implemented and tested as S-functions.

5. Controller programming in a single S function
   To facilitate laboratory implementation, all functions are combined in a single S-function.

2. Fundamentals

The input voltages applied to the asynchronous machine (ASM) are three-phase alternating variables. With the help of the space vector representation, the electrical quantities of a three-phase, symmetrical system can be defined as complex vector quantities in a two-axis coordinate system (COS) [1]. The three real equations of the three-phase system are reduced to a complex equation by the space vector representation.
The system of the asynchronous machine can be further simplified with the help of a coordinate transformation (park transformation). The advantage of the representation in a rotating coordinate system is the possibility of transforming alternating quantities into equal quantities at the corresponding angular velocity of the coordinate system.

2.1 Space vector representation and coordinate transformation

The time characteristic of a sinusoidal three-phase system without neutral can be described by the functions given in the equation (2.1).

$$\begin{aligned} f_a(t) &= \hat{f}\cos(\omega t) \\ f_b(t) &= \hat{f}\cos(\omega t - 120°) \\ f_c(t) &= \hat{f}\cos(\omega t - 240°)\,. \end{aligned} \tag{2.1}$$

The three phases a, b and c are spatially displaced by $120°$ each and the time-dependent angle of the three-phase magnitude is indicated by ωt.
The geometrical sum of the three linearly dependent phases of a three-phase system without neutral is zero:

$$f_a(t) + f_b(t) + f_c(t) = 0. \tag{2.2}$$

If there is such a three-phase three-wire system, according to the first Kirchhoff law [2] one phase can always be expressed by the other two phases. The three-phase system without neutral is thus uniquely determined by the linear combination of two phases. This basic property is used to define a space vector. The three phases are reduced by an amplitude invariant transformation to a complex-valued system with only two components - the so-called stationary $\alpha\beta$ coordinate system.

$$f_\alpha(t) + jf_\beta(t) = \underset{\rightarrow}{f}(t) \tag{2.3}$$

with:

$$\begin{aligned} f_\alpha(t) &= \frac{2}{3}f_a(t) - \frac{1}{3}f_b(t) - \frac{1}{3}f_c(t) \qquad \text{und} \\ f_\beta(t) &= \frac{1}{\sqrt{3}}\left(f_b(t) - f_c(t)\right) \end{aligned} \tag{2.4}$$

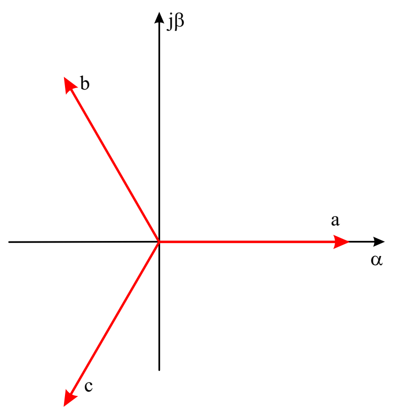
$\underset{\rightarrow}{f}(t)$ is called a complex space vector. In matrix notation, the transformation into space vectors can be represented as follows:

$$\begin{bmatrix} f_\alpha \\ f_\beta \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix} \tag{2.5}$$

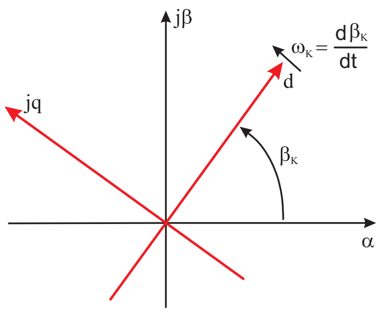
The reverse transformation of the space vector into phase values is analogous:

$$\begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} f_\alpha \\ f_\beta \end{bmatrix} \tag{2.6}$$

Space vectors can be represented in a two-dimensional orthogonal coordinate system. Fig. 1 a) shows the representation of the three-axis coordinate system and the coordinate system reduced to two axes. The Park transformation is often used for a simpler electrical description of rotating field machines. This coordinate transformation is a uniquely reversible transformation. It allows a rotation of the two-dimensional coordinate system with the arbitrarily selectable angular velocity ω_K. Figure 1 b) shows the functionality of the coordinate transformation into a rotating coordinate system.



(a) Three-phase system (a,b,c) (red) and two-axis (α,β)- COS (black)



(b) Stationary (α,β) - COS (black) and rotating (d,q)- COS (red)

Fig. 1: Transformation into the rotating coordinate system.

The angle β_K denotes the angle between the d-axis of the coordinate system rotating with the angular velocity ω_K and the real axis of the stationary coordinate system. The transformation instruction from the stationary coordinate system to a coordinate system rotating with ω_K is as follows:

$$\begin{bmatrix} f_d \\ f_q \end{bmatrix} = \begin{bmatrix} \cos(\beta_K) & \sin(\beta_K) \\ -\sin(\beta_K) & \cos(\beta_K) \end{bmatrix} \begin{bmatrix} f_\alpha \\ f_\beta \end{bmatrix} \tag{2.7}$$

The reverse transformation instruction is:

$$\begin{bmatrix} f_\alpha \\ f_\beta \end{bmatrix} = \begin{bmatrix} \cos(\beta_K) & -\sin(\beta_K) \\ \sin(\beta_K) & \cos(\beta_K) \end{bmatrix} \begin{bmatrix} f_d \\ f_q \end{bmatrix} \tag{2.8}$$

A proper selection of the rotational speed of the rotating coordinate system results in particularly simple relations in the mathematical description of the asynchronous machine used. If the rotational speed is selected as the rotational speed of the rotor flux $\underrightarrow{\Psi}_R$ of the asynchronous machine, the rotating coordinate system is generally referred to as the dq coordinate system [3]. Accordingly, the resulting components are called d- or q-components of the rotating coordinate system.

2.2 Fundamentals of asynchronous machines

In the following, the machine equations for a coordinate system 'K' rotating with the angular frequency ω_K are determined based on the single-phase equivalent circuit diagram of the asynchronous machine for dynamic operation. Fig. 2 shows the single-phase equivalent circuit diagram of an asynchronous machine for dynamic operation in a generically rotating coordinate system. It is assumed that the windings are distributed symmetrically around the periphery of the machine and are fed by a symmetrical three-phase voltage system. Furthermore, it is assumed as an approximation that iron losses can be neglected and that all resistances are constant.
All rotor sizes (index R) are, with the turn ratio *n* related to the stator (index S). The related quantities are usually marked with ', which is neglected in this case.
By forming the mesh equations on the stator and rotor sides of the equivalent circuit

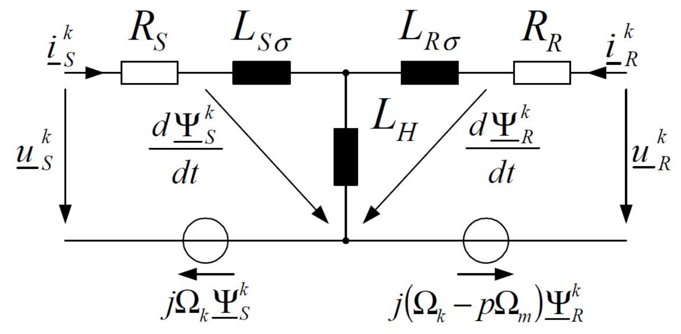

$\underline{i}_S^k$
R_S
$L_{S\sigma}$
$L_{R\sigma}$
R_R
$\underline{i}_R^k$
$\underline{u}_S^k$
$\frac{d\underline{\Psi}_S^k}{dt}$
L_H
$\frac{d\underline{\Psi}_R^k}{dt}$
$\underline{u}_R^k$
$j\Omega_k\underline{\Psi}_S^k$
$j(\Omega_k - p\Omega_m)\underline{\Psi}_R^k$

Fig. 2: Single-phase equivalent circuit diagram of an asynchronous machine for dynamic operation in a coordinate system rotating with ω_K $\Omega = \omega$.

diagram, the differential equations of the electrical behavior of the ASM can be obtained in any coordinate system:

$$\begin{aligned} \underrightarrow{u}_S^K &= R_S \underrightarrow{i}_S^K + j\omega_K \underrightarrow{\Psi}_S^K + \frac{d\underrightarrow{\Psi}_S^K}{dt} \\ \underrightarrow{u}_R^K &= R_R \underrightarrow{i}_R^K + j\left(\omega_K - p\omega_m\right) \underrightarrow{\Psi}_R^K + \frac{d\underrightarrow{\Psi}_R^K}{dt}. \end{aligned} \tag{2.9}$$

By rearranging the flux equations:

$$\begin{aligned} \underrightarrow{\Psi}_S^K &= L_S \underrightarrow{i}_S^K + L_H \underrightarrow{i}_R^K \\ \underrightarrow{\Psi}_R^K &= L_H \underrightarrow{i}_S^K + L_R \underrightarrow{i}_R^K \end{aligned} \tag{2.10}$$

with $L_S = L_{S\sigma} + L_H$ and $L_R = L_{R\sigma} + L_H$ the equations for the stator and the rotor current result:

$$\begin{aligned} \underrightarrow{i}_S^K &= \frac{1}{\sigma L_S} \underrightarrow{\Psi}_S^K - \frac{L_H}{\sigma L_S L_R} \underrightarrow{\Psi}_R^K \\ \underrightarrow{i}_R^K &= \frac{1}{\sigma L_R} \underrightarrow{\Psi}_R^K - \frac{L_H}{\sigma L_S L_R} \underrightarrow{\Psi}_S^K \end{aligned} \tag{2.11}$$

The total leakage factor σ is defined by :

$$\sigma = 1 - \frac{L_H^2}{L_S L_R}. \tag{2.12}$$

The motion differential equation for the mechanical angular velocity of the rotor is described by:

$$\frac{d\omega_m}{dt} = \frac{1}{\Theta}\left(M_i - M_L\right) \tag{2.13}$$

where Θ is the moment of inertia of the machine, M_L is the load torque and M_i is the air gap torque. The air gap torque can be calculated by:

$$M_i = \frac{3}{2} p \frac{L_H}{L_R} Im\left\{\underrightarrow{\Psi}_R^K \underrightarrow{i}_S^K\right\} \tag{2.14}$$

The rotor voltages of the asynchronous machine with squirrel-cage rotor are zero.

$$u_{Rd} = u_{Rq} = 0 \tag{2.15}$$

The dynamic behavior of the ASM in a rotating coordinate system can be represented by the signal flow diagram according to [1] shown in Fig. 3.

2.3 Discretization methods

To implement the control on a microprocessor, it must be discretized. Various methods are available. Frequently used methods are the Euler-Forward, Euler-Backward and the Tustin method. For the implementation the factor *s* in the transfer function is substituted by the factors in the table 2.1. The aim is to apply the inverse Z-transformation to convert the equation into the discrete domain. The transfer function must be brought into the following form:

$$G(z) = a \cdot z^{-1} + b \tag{2.16}$$

This form can be used to apply the displacement law. A description of these procedures can be found in [4] and [5].

Discretization method	Substitution
Euler-forward	$s = \frac{z-1}{T_A}$
Euler-backward	$s = \frac{z-1}{z \cdot T_A}$
Tustin	$s = \frac{2}{T_A} \frac{z-1}{z+1}$

Tab. 2.1: Substitution equations for discretization

2.4 Controller programming with S-Functions

S-Functions can be used to integrate C, C++ or Fortran code into MATLAB/Simulink. Thus the code for the microcontroller can be developed and debugged in MATLAB. [6] and [7] provide implementation instructions. At this point a short overview of S-Functions is given, whereby the following explanations refer to the integration of C code. The code in the S-functions is therefore written in C, possibly necessary libraries (<math.h>, etc.) have to be included via #include.
Both parameters and variables can be transferred from the Simulink interface. The parameters must be entered in the S-Function mask in Simulink and can then be read in using a pointer. The following command line is used for this:

```
#define Parameter1 *(mxGetPr(ssGetSFcnParam(S,0)))
#define Parameter2 *(mxGetPr(ssGetSFcnParam(S,1)))
```

For the definition of the input and output variables, fixed code fragments are also necessary, which are already embedded in the basic structure of the S-function. Both one-dimensional variables and vectors can be read or output. The integration is also carried out via pointers.
The appendix contains a documented example for the use of S-Functions.

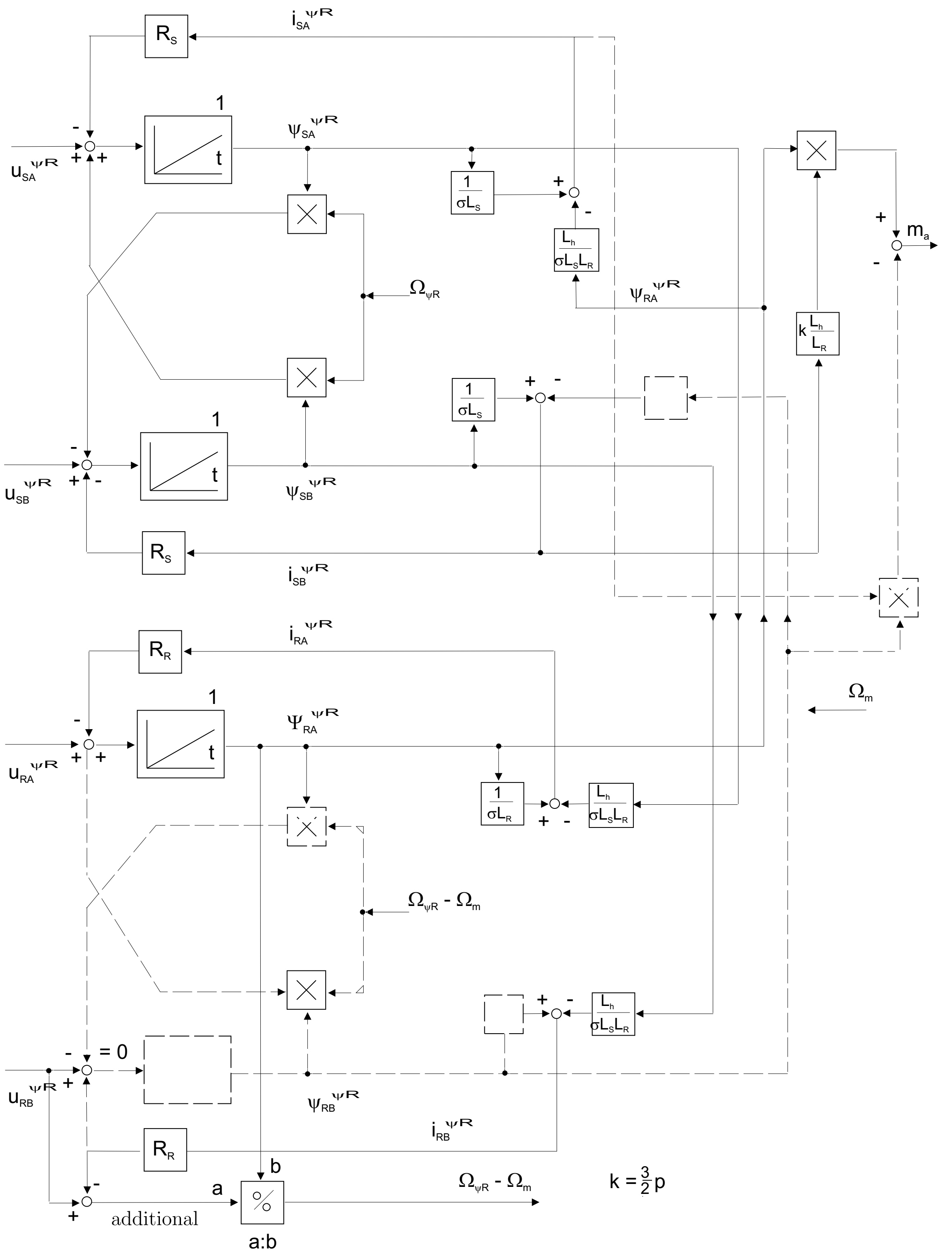


Fig. 3: Signal flow diagram of the ASM with rotor flux orientation.

C programming knowledge is required for programming the S-Functions. The most important commands are specified here. It is important to distinguish between the different data formats of variables and parameters:

- #defined-declared parameters: Usually written in capital letters, since these values cannot be changed. They are not numbers, i.e. in arithmetic operations it may be necessary to convert them to float or double (e.g. when dividing by a define).
- global variables or parameters (int, float or double): These values are available in all functions and are not passed to a function. Since global variables require more memory space and the clarity of an extensive program with many globally defined variables suffers, the number should be kept to a minimum.
- local variables (int, float or double): Only available locally in the corresponding function and must be passed accordingly when a function is called.

Pointers are used for the return of the function call. The two most important operators in operation with pointers are:

- "‘&"’: The address operator that, when applied to an object, returns the address of the corresponding object.
- "‘*"’: The dereferencing operator that, applied to a pointer, returns the object stored at that address.

If an address is passed during a function call and the function points to this address with a pointer, the value at the corresponding address can be changed in the called function. As an example a function call is given, whereby an additional declaration is necessary in the header file:

- Function call: function(variable, &address)

The value "‘variable"’ and the address "‘address"’ are passed. The called function has the following form:

- void function(int var, float *adress)

Note that the file format of the passed variables and the locally redefined variables must be the same, but the names may be different. The value stored at the given address can be changed and used by "‘*adress"’.

The main file formats used during the experiment are integer (int), float and double. Since the file formats int and float are used on the μController, the file format double should also be avoided in this part of the experiment if possible.

3. Field-oriented control

3.1 Introduction

The aim of field-oriented control (FOC) is to be able to set the torque-forming stator current independently of the field-forming stator current. In [3] different possibilities are shown how a decoupling of these stator current components can be achieved. For the control considered here, an orientation at the rotor flux of the asynchronous machine is selected. If the flux linkage of the rotor $\underset{\rightarrow}{\Psi}_R$ is selected as reference system, the q-component of the rotor flux linkage is zero ($\Psi_{Rq} = 0$). Since an asynchronous machine with a squirrel cage rotor is used, the rotor voltage components are also zero. If this is applied to the general machine equations, the machine equations for orientation on rotor flux, equations (3.17) to (3.19), result.

$$\Psi_{Rq} = 0 \tag{3.17}$$

$$\frac{d\Psi_{Rd}}{dt} = \frac{R_R}{L_R}\left(L_H i_{sd} - \Psi_{Rd}\right) \tag{3.18}$$

$$M_i = \frac{3}{2} p \frac{L_H}{L_R} \Psi_{Rd} i_{Sq} \tag{3.19}$$

The angular velocity of the rotating coordinate system is:

$$\omega_K = \omega_{\Psi_{Rd}} = \frac{R_R L_H}{L_R} \frac{i_{Sq}}{\Psi_{Rd}} + p\omega_M \tag{3.20}$$

These equations are applied in the flux-torque-model, which serves to calculate the flux, the torque, the angular velocity and the angle for the dq-transformation from the measured currents and the measured rotational speed.

3.2 Structure diagram

Fig. 4 shows the structural diagram of the field-oriented control. It is a cascaded control structure with an inner current control loop and outer flux and speed control. The stator currents of the machine and the motor speed are measured. The rotor flux is calculated using a flux model and the d-component of the stator current, see Fig. 5. Fig. 5 shows the

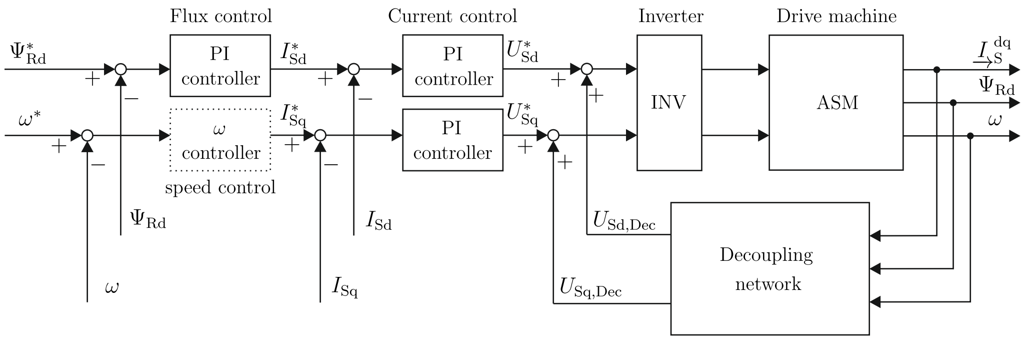


Fig. 4: Structure diagram of the FOC.

machine model of the ASM with orientation at the rotor flux. It also shows how the rotor flux, the machine torque and the transformation angle for the coordinate transformation are calculated.

3.3 Current control

3.3.1 Creating the Transfer Functions

First, the control for the inner current control loop is designed.
The differential equations for the current dynamics can be determined by converting the

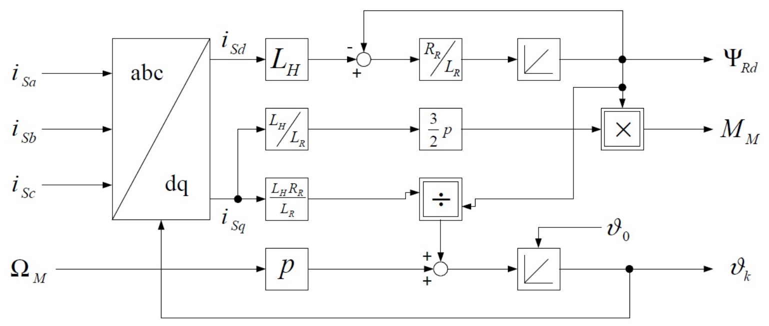


Fig. 5: Machine model of the ASM for orientation at the rotor flux.

machine equations and representing them in real and imaginary parts (d and q components):

$$\begin{aligned}
\frac{di_{Sd}}{dt} &= -\left(\frac{R_S}{\sigma L_S} + \frac{R_R L_H^2}{\sigma L_S L_R^2}\right) i_{Sd} + \frac{1}{\sigma L_S} u_{sd} + p\omega_M i_{Sq} + \frac{R_R L_H}{L_R}\frac{i_{Sq}^2}{\Psi_{Rd}} + \frac{R_R L_H}{\sigma L_S L_R^2}\Psi_{Rd} \\
\frac{di_{Sq}}{dt} &= -\left(\frac{R_S}{\sigma L_S} + \frac{R_R L_H^2}{\sigma L_S L_R^2}\right) i_{Sq} + \frac{1}{\sigma L_S} u_{sq} - p\omega_M i_{Sd} - \frac{R_R L_H}{L_R}\frac{i_{Sd} i_{Sq}}{\Psi_{Rd}} - p\omega_M \frac{L_H}{\sigma L_S L_R}\Psi_{Rd}
\end{aligned} \tag{3.21}$$

It can be seen that the differential equations for the stator current are coupled (dependence between d- and q-components and other quantities). This coupling can be reduced by selecting the input voltages u_{Sd} and u_{Sq} (decoupling network, see Fig. 4).
The decoupling network simplifies the current dynamics:

$$\begin{aligned}
\frac{di_{Sd}}{dt} &= -\left(\frac{R_S}{\sigma L_S} + \frac{R_R L_H^2}{\sigma L_S L_R^2}\right) i_{Sd} + \frac{1}{\sigma L_S} u_{sd}^* \\
\frac{di_{Sq}}{dt} &= -\left(\frac{R_S}{\sigma L_S} + \frac{R_R L_H^2}{\sigma L_S L_R^2}\right) i_{Sq} + \frac{1}{\sigma L_S} u_{sq}^*
\end{aligned} \tag{3.22}$$

This leads to the transfer function of the current dynamics of the ASM:

$$G_I(s) = \frac{I_{S,dq}(s)}{U_{S,dq}(s)} = V_S \cdot \frac{1}{1+sT_N} = \frac{\frac{L_R^2}{R_S L_R^2 + R_R L_H^2}}{1 + s\frac{L_S L_R^2 \sigma}{R_S L_R^2 + R_R L_H^2}} \tag{3.23}$$

The inverter can be simplified as a first-order delay element:

$$G_{SR}(s) = \frac{1}{1+sT_{SR}} \tag{3.24}$$

With $T_{SR} = \frac{1}{2f_P}$ (f_P: pulse frequency=sampling frequency).
The measured values required for the control are not continuously available, but are sampled at discrete points in time. The sampling results in a further delay which must be taken into account when designing the current control.
A sampling cycle passes from the sampling to the output of the voltage vector calculated from it. The control outputs a voltage vector averaged over a sampling period. For the controller design, this resulting delay can be approximated by a first-order delay

element:

$$G_{SR}^*(s) = \frac{1}{1 + sT_{SR}^*} \tag{3.25}$$

With $T_{SR}^* = \frac{3}{2f_P}$. Considering the introduced decoupling network, the transfer behavior of the current control for the d- and q-components can be treated equivalently. As a result, the PI current controllers are designed the same for the d and q components.
The plant therefore consists of two first-order delay elements, whereby the time constant of the pulse inverter is in most cases considerably smaller than the time constant of the asynchronous machine. A PI controller is used to control this plant.

3.3.2 Transfer function of the closed inner control loop

If the control transfer function of the closed inner current control loop is calculated from the selected PI parameters, a second order delay element results, cf. (3.26). As can be seen from this equation, the control dynamics of the reference variable depend only on the small time constant of the converter, provided that the manipulated variables are not limited. For small sampling times, the reference transfer function (3.26) can be approximated by a first-order delay element.

$$G_{N,I}(s) = \frac{1}{1 + s2T_{SR}^* + s^2 2T_{SR}^{*2}} \approx \frac{1}{1 + s2T_{SR}^*} \tag{3.26}$$

This simplification of the current control loop's leading transfer function leads to a simpler design of the overlaid flux and speed control, which will be briefly explained below.

3.4 Flux control - transfer function

The design of the inner current control loops for controlling the field-forming stator current (d-component) and for controlling the torque-forming stator current (q-component) was summarized in the previous section. The transfer function of the rotor flux linkage can be determined by means of the differential equation (3.18):

$$G_{\Psi_R}(s) = \frac{\Psi_R(s)}{I_{Sd}(s)} = V_S \frac{1}{1 + sT_N} = \frac{L_H}{1 + s\frac{L_R}{R_R}} \tag{3.27}$$

By designing the current control according to the technical optimum, the reference variable response of the inner closed control loop can be simplified and described as a first-order delay (3.26). A PI controller is again used to control the flux linkage of the rotor:

$$G_{PI,\Psi}(s) = \frac{I_{Sd}^*(s)}{\Psi_{Rd}^*(s) - \Psi_{Rd}(s)} = V_{R,\Psi}\left(1 + \frac{1}{sT_{N,\Psi}}\right) \tag{3.28}$$

The time constant of the flux linkage transfer function is much greater than the equivalent time constant of the closed loop inner current control. The resulting control problem of determining the PI control parameters is therefore analogous to the current control design. The optimization method according to the technical optimum is used again to calculate the parameters of the PI controller (3.32) and (3.33).

3.5 Speed control - Transfer function

The differential equation of the speed dynamics of the ASM is:

$$\frac{d}{dt}\omega_M = \frac{1}{\Theta}\left(M_{Mi} - M_L\right) \tag{3.29}$$

The load torque M_L can be considered as a disturbance and is not taken into account for the controller design. With equation (3.19) for the torque of the asynchronous machine, the transfer function of the speed dynamics can be set up:

$$G_{\omega_M}(s) = \frac{\Omega_M(s)}{I_{Sq}(s)} = \frac{\frac{3}{2}p\frac{L_H}{L_R}\Psi_{Rd}}{\Theta s} \tag{3.30}$$

It can be seen that the speed dynamics show a purely integrative behaviour. In this case, the symmetrical optimum (SO) is recommended after [3] to optimize the PI controller.

3.6 Tuning of the PI controller

3.6.1 Tuning of the PI controller according to the technical optimum

The design of the PI controller should lead to the compensation of the largest time constant of the given plant and to avoid a permanent control deviation in steady-state operation [3]. The requirement for compensation of the largest time constant leads to the greatest possible dynamic for the controlled plant under consideration. In this case, the control dynamics are only determined by the time constant of the pulse inverter. [3] summarizes a procedure for optimizing the control behavior of this type of plant.
Transfer function of the PI controller:

$$G_{PI,I}(s) = \frac{U^*_{S,dq}(s)}{I^*_{S,dq}(s) - I_{S,dq}(s)} = V_{R,I}\left(1 + \frac{1}{sT_{N,I}}\right) = V_{R,I} + \frac{V_{R,I}}{sT_{N,I}} \tag{3.31}$$

Tuning of the PI current control by means of the technical optimum (TO) [3]:

$$V_{R,I} = \frac{T_1}{2V_S T_\sigma} \tag{3.32}$$

$$T_{N,I} = T_1 \tag{3.33}$$

With T_1 largest time constant, T_σ smaller time constant and V_S gain of the considered plant.

3.6.2 Tuning of a PI controller according to symmetrical optimum

The equation of the PI controller is:

$$G_{PI,\omega}(s) = \frac{I^*_{Sq}(s)}{\Omega^*_M(s) - \Omega_M(s)} = V_{R,\omega}\left(1 + \frac{1}{sT_{N,\omega}}\right) = V_{R,\omega} + \frac{V_{R,\omega}}{sT_{N,\omega}} \tag{3.34}$$

With optimization according to the symmetrical optimum:

$$T_{N,\omega} = 4T_\sigma \tag{3.35}$$

$$V_{R,\omega} = \frac{T_1}{2V_S T_\sigma} \tag{3.36}$$

In some cases the SO leads to a too high gain. In these cases the generalized SO can be used. A factor a is introduced when determining the controller parameters. This results in the general tuning rules:

$$T_{N,\omega} = a^2 T_\sigma \tag{3.37}$$

$$V_{R,\omega} = \frac{1}{a}\frac{T_1}{V_S T_\sigma} \tag{3.38}$$

3.7 Limitation of manipulated variable

In this section, the limitations of the manipulated variables required for the implementation of the FOC in the laboratory are explained.
The d-component of the field-oriented control regulates the rotor flux linkage of the drive machine and the q-component regulates the speed on the machine side. If we consider the machine equations in the orientation to the rotor flux linkage, see (3.18) and (3.19) it can be seen that the generation of the torque of the drive machine depends on the rotor flux linkage. With regard to the limitation of the manipulated variable of the individual PI-controllers of the cascaded control to be implemented, this means that the control of the field-forming d-component, i.e. the maintenance of the rotor flux linkage, should always have priority over the control of the torque-forming q-component. Only if the drive motor is excited to the nominal rotor flux chaining can the nominal torque and thus the complete dynamic operating range of the asynchronous motor used be utilized.
First, the implementation of the limitation of the manipulated variable of the outer flux and speed controllers is explained. The maximum current $i_{\max}$ for limiting the manipulated variable of the rotor flux controller is defined by the maximum permissible current of the drive machine used:

$$i^*_{Sd,limit} = \begin{cases} i^*_{Sd} & \text{for} \quad i^*_{Sd} \in [-i_{max}, i_{max}] \\ i_{max} & \text{for} \quad i^*_{Sd} > i_{max} \\ -i_{max} & \text{for} \quad i^*_{Sd} < -i_{max} \end{cases} \tag{3.39}$$

$$i_{Sq,max} = \sqrt{i^2_{max} - i^{*2}_{Sd,limit}} \tag{3.40}$$

$$i^*_{Sq,limit} = \begin{cases} i^*_{Sq} & \text{for} \quad i^*_{Sq} \in [-i_{Sq,max}, i_{Sq,max}] \\ i_{Sq,max} & \text{for} \quad i^*_{Sq} > i_{Sq,max} \\ -i_{Sq,max} & \text{for} \quad i^*_{Sq} < -i_{Sq,max} \end{cases} \tag{3.41}$$

Now the realization of the limitation of the manipulated variable of the inner current controller is explained. The maximum permissible voltage which may be set by the current controllers is determined by the pulse width modulation of the pulse inverter used. In order to avoid overmodulation when generating the switching pattern for the power semiconductors, the maximum permissible control voltage $u_{\max}$ of the current controllers is limited to $\frac{1}{\sqrt{3}}U_{DC}$. Since the flux linkage of the driving machine should be maximum, the d-component has priority. The permissible manipulating range of the current control of the q-component then follows from the limitation of the d-component:

$$u^*_{Sd,limit} = \begin{cases} u^*_{Sd} & \text{for} \quad u^*_{Sd} \in [-u_{max}, u_{max}] \\ u_{max} & \text{for} \quad u^*_{Sd} > u_{max} \\ -u_{max} & \text{for} \quad u^*_{Sd} < -u_{max} \end{cases} \tag{3.42}$$

$$u_{Sq,max} = \sqrt{u_{max}^2 - u_{Sd,limit}^{*2}} \tag{3.43}$$

$$u_{Sq,limit}^{*} = \begin{cases} u_{Sq}^{*} & \text{for} \quad u_{Sq}^{*} \in [-u_{Sq,max}, u_{Sq,max}] \\ u_{Sq,max} & \text{for} \quad u_{Sq}^{*} > u_{Sq,max} \\ -u_{Sq,max} & \text{for} \quad u_{Sq}^{*} < -u_{Sq,max} \end{cases} \tag{3.44}$$

3.8 Integrator Windup

If the manipulated variable of the controller is limited, the so-called integrator windup effect can occur. This means that the integrator of the controller further integrates the control deviation in the manipulated variable limitation without increasing the manipulated variable. If the control deviation then becomes smaller, the integrator causes an unintended delay of the manipulated variable. As a result, significantly greater overshoots and settling times can occur, for example, during reference variable jumps [8]. In Fig. 6 is a classic anti-windup network according to [8] for a PI controller. The difference between the unlimited and the limited manipulated variable is formed. This value is weighted with the time constant T_a and fed back to the input of the integrator. If a limitation of the manipulated variable now occurs, the anti-windup network prevents the control difference from being integrated, thus preventing strong overshooting. The anti-windup network is only active if a limitation of the manipulated variable occurs. A first choice for the selection of the time constants T_a is to set it equal to the reset time of the controller $T_a = T_N$ [8].

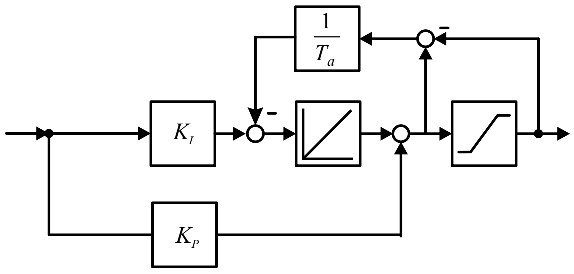


Fig. 6: Anti-windup network.

4. Experiment preparation

Experimental part I is divided into five experimental sections. The respective experimental sections are to be prepared on the basis of the following questions:

1. Questions about experimental sections 1 & 2 - Controller tuning
   a) Determine the missing parameters in the table 6.2 from the appendix (Note that the mechanical power of the machine is given!).
   b) Derive the equation (3.21) of the current dynamics using the equations (2.9) to (2.11). (Note: Also use equations (3.17), (3.18) and (3.20))
   c) List the advantages and disadvantages of the d-q transformation.
   d) Specify the decoupling network with equations and as a structural diagram to decouple the current dynamics from equation (3.21) and explain the principle.
   e) Explain why a linear plant model can be used.
   f) Determine the (as linearly approximated) open loop transfer function of the inner and outer control loop.
   g) Explain in which cases the technical optimum is applied and in which cases the symmetrical optimum. What is the aim of the optimization methods (symmetrical optimum and technical optimum)?
   h) Tune PI controllers of the field-oriented control for the system parameters listed in the appendix.
   i) Describe the phenomena of the integrator windup. Which quantities should be limited? Which quantity has priority?
2. Questions about experimental section 3 - Controller discretization
   a) Why does a discrete analysis have to be carried out? What influence does discretization have in the context of controller synthesis? How could we proceed with controller synthesis instead? When will another procedure be necessary?
   b) Delimit the time-continuous from the time-discrete system Fig. 4.
   c) Perform the discretization of an integrator $Y(s)/U(s) = 1/s$ with the Forward-Euler method and determine the time-discrete representation $y(k)$.
   d) Discretize the PI controller for the three given methods.
   e) Discretize the flux model, decoupling network and anti-windup using the Euler forward method.
   f) Sketch all discrete space vectors of a 2 level voltage converter for a stationary coordinate system. What is the magnitude of the maximum output voltage for a DC link voltage U_{dc}? What AC voltage can this inverter provide?
   g) Which two switching sequences can be implemented with space vector modulation in sector 4 if both zero space vectors are to be used?
3. Questions about experimental sections 4 & 5 - Controller programming
   a) What is an S-function? Explain why it makes sense to use S-Functions, and when they are used.
   b) Draw the flowchart of an S-function. Which routines are necessary in an S-function? Within which routine is a function called (e.g. a PI controller)?

c) How is the S-function compiled?

d) From the results of experimental section 3, create a C code for the following functions:

   i. PI controller with limitation and anti-windup. Remember that the anti-windup only affects the integral part of the controller.

   ii. Decoupling network

   iii. Flux model

   Use the block diagrams from former experimental sections and transform the transfer functions into the time-discrete domain. Define input variables, output variables and states.

5. Experiment execution and experiment evaluation

Record the results of all subsequent subtasks in the protocol. Use separate Simulink models for each experimental section.

1. Experimental section 1 - Controller synthesis

   a) Copy the simulation template from the network directory:
   \\leaserver\peprk\master_laboratory\experimental_part_1\basis
   into your working directory. Start the simulation interface and make sure that the plant parameters correspond to those in Appendix 1.

   b) Create an m-file for the following analytical considerations. Use the plant parameters from Appendix 1 for the considerations.

      i. Create the continuous transfer functions of the inner control loop plant and design the controller according to the instructions. Discuss the open control loop with and without controller. What amplitude reserve and phase reserve are available?
      (Note: Helpful functions: margin(G(s)), feedback(G(s),1))

      ii. Discuss the closed control loop with regard to dynamics. In particular, discuss the overshoot and the settling time. Give the boundary conditions for the settling time and define them.
      (Note: Helpful function: stepinfo(G(s)))

2. Experimental section 2 - Implementation in Matlab/Simulink

   a) Check the flux-torque-model. Implement the continuous controller for the inner control loop in the simulation and compare it with the analysis above in terms of dynamics.

   b) How do you recognize a coupling? Minimize the effect with the help of a decoupling network.

   c) Implement and check the anti-windup for the inner control loop. Are there any effects on the dynamics?

   d) Implement the flux and speed control with manipulated variable limitation and evaluate the dynamics of the speed controller on the basis of the simulation. Implement an anti-windup structure here as well.

3. Experimental section 3 - Controller discretization (time discretization)
   a) Transform the plant and the controllers into the z-domian. Compare the controller in the continuous case with the discontinuous case for the three given discretization methods. Use the Bode diagram (frequency response diagram) to do this. What do you find? What influcence has the sampling frequency? (Note: Helpful functions: c2d())
   b) Implement a discrete variant in the simulation. What problems exist when using time-discrete implementation?
   Hints: Use triggered subsystems that are controlled accordingly and sample the necessary signals with zero-order-hold.
4. Experimental section 4 - Controller programming into individual S-Functions
   a) Implement the S-functions created during preparation for the speed and flux controller including anti-windup, the current controller including anti-windup, the decoupling network and the flux model.
      i. Start with the current controller and make sure step by step that the individual functions are correct.
      ii. Then implement the speed and flux controller.
      iii. Finish this task with the implementation of the flux-torque-model.
      iv. Make sure that your control, based on S-Functions, is properly functioning. To do this, compare the results with the discrete-time control model.
5. Experimental section 5 - Controller programming in a single S-function
   a) Create a model that uses a single S-function. This single function should contain the ABC/DQ transformation, the flux-torque-model, the controllers with anti-windup and the decoupling network. Furthermore, the function should output a space vector (magnitude and phase), which is necessary for the PWM.
   b) Record the step responses in the simulation for current, flux and speed control in order to compare them with the laboratory measured values in experimental part III.
   c) Copy your finished functions to implement them in the laboratory.

   Note for experimental section 4:
   For the pointers in your S-Functions, use only pointers that point to a single element, not an array. This is necessary for implementation on the microcontroller in the laboratory.

6. Appendix

6.1 Machine parameters

Designation	Parameter	Value
Stator resistance	R_S	$3.9\,\Omega$
Stator leakage inductance	$L_{\mathrm{S}\sigma}$	$9.05\,\mathrm{mH}$
Related rotor resistance	R'_R	$1.6\,\Omega$
Related rotor leakage inductance	$L'_{\mathrm{R}\sigma}$	$9.05\,\mathrm{mH}$
Main inductance	L_h	$404\,\mathrm{mH}$
Number of pole pairs	p	
Inertia	Θ	$0.0018\,\mathrm{kgm}^2$
Rated power	P_rated	$2.2\,\mathrm{kW}$
Power factor	$\cos(\varphi)$	0.85
Efficiency	η	0.859
Stator voltage	$\tilde{U}_\mathrm{S,rated,ll}$	$400\,\mathrm{V}$
Stator current	$\underline{I}_\mathrm{S,rated}$	
Idle current	$\underline{I}_\mathrm{S,0}$	$1.87\,\mathrm{Ae}^{-\mathrm{j}74^\circ}$
Stator frequency	f_N	$50\,\mathrm{Hz}$
Idle speed	N_0	$3000\,\mathrm{r/min}$
Rated speed	N_rated	$2895\,\mathrm{r/min}$
Rated slip	s_rated	
DC link voltage	U_dc	$566\,\mathrm{V}$
Max. permissible current	i_max	$6\,\mathrm{A}$
Rated flux	$\hat{\Psi}_\mathrm{rated}$	$0.98\,\mathrm{Vs}$
Rated torque	M_rated	
Switching frequency converter	f_sw	$5\,\mathrm{kHz}$
Sampling frequency	f_A	$5\,\mathrm{kHz}$

Tab. 6.2: Machine parameters.

6.2 Example: Usage S-Function

6.2.1 S-Function: Communication with MATLAB-Simulink

```
/*-----------------------------------------------------------------------+
Filename: s_abc_dq_transformation.c
By:
-----------------------------------------------------------------------+
Purpose:  C-mex s-function
          KOS-Transformations

-----------------------------------------------------------------------+
 *               Channel        Value     [Dimension]     Explanation
 * Inputs:        1             abc          [3]
 *                2             gamma        [1]         Transformation angle (Park)

Block parameter input:
 *               1              fsample      [1]         Sampling frequency
 *               2              xy               [2]         Example


Output:         Channel        Value
 *              1               y_dq         [2]         Output variable in the dq-KOS
 *              2               debug        [3]         debug


Compile:    mex s_abc_dq_transformation.c transformations.c
-----------------------------------------------------------------------+
Revision history:::
Date:    ID:   Description:

----------------------------------------------------------------------*/


#define S_FUNCTION_NAME  s_abc_dq_transformation
#define S_FUNCTION_LEVEL 2
```

```
#define NO_OF_PARAMETERS 1
#define NO_OF_INPUTS 2
#define NO_OF_INPUTS_PORT0   3    // u (abc)
#define NO_OF_INPUTS_PORT1   1    // gamma
#define NO_OF_OUTPUTS 2
#define NO_OF_OUTPUTS_PORT0  2   // y (dq)
#define NO_OF_OUTPUTS_PORT1  3   // debug

// Loading block parameters
#define fsample     *(mxGetPr(ssGetSFcnParam(S,0)))         // Sampling frequency

// Define constants

#include <stdio.h>
#include <math.h>
#include "simstruc.h"

// ----------------------- Integrate your own functions ----------------------
#include "transformations.h"

// ----------------------- Initialization  ----------------------
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumSFcnParams(S, NO_OF_PARAMETERS);
   if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
   {
      return; /* Parameter mismatch will be reported by Simulink */
   }

   ssSetNumContStates(S, 0);
   ssSetNumDiscStates(S, 0);

   if (!ssSetNumInputPorts(S, NO_OF_INPUTS)) return;
   ssSetInputPortWidth(S, 0, NO_OF_INPUTS_PORT0);       // for each input channel
   ssSetInputPortWidth(S, 1, NO_OF_INPUTS_PORT1);


   ssSetInputPortDirectFeedThrough(S, 0, 1);            // for each input channel
```

```
    ssSetInputPortDirectFeedThrough(S, 1, 1);


    if (!ssSetNumOutputPorts(S, NO_OF_OUTPUTS)) return;
    ssSetOutputPortWidth(S, 0, NO_OF_OUTPUTS_PORT0);
    ssSetOutputPortWidth(S, 1, NO_OF_OUTPUTS_PORT1);     // for each output channel



    ssSetNumSampleTimes(S, 1);

                             /* Take care when specifying exception free code - see sfuntmpl.doc */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);

    // user defined variables
    init_var(fsample); //Aufruf der Initialisierungsdatei für Variablen (in der h-Datei definiert)
}

// Define discrete execution time
static void mdlInitializeSampleTimes(SimStruct *S)
{
    // Sample time
    ssSetSampleTime(S, 0, (1/fsample));
    ssSetOffsetTime(S, 0, 0*(1/fsample)/2);
}


static void mdlOutputs(SimStruct *S, int_T tid)
{
    // Definition of outputs    // for each input channel
    real_T  *output     = ssGetOutputPortRealSignal(S,0);  //  Output pointer, y (dq)
    real_T  *debug      = ssGetOutputPortRealSignal(S,1);  //  Output pointer, debug
    // Definition Eingänge   // für jeden Output-Channel
    InputRealPtrsType input       = ssGetInputPortRealSignalPtrs(S, 0); // Input signal
    InputRealPtrsType gamma      = ssGetInputPortRealSignalPtrs(S, 1); // Transformation angle

    real_T in_a = *input[0];
    real_T in_b = *input[1];
    real_T in_c = *input[2];
    real_T out_d;
    real_T out_q;


    //real_T Is_B = *Is_AB1[1];

    //Transformation
    abc_dq_trans(in_a,  in_b,  in_c,  *gamma[0],  &out_d, &out_q);
    output[0]   = out_d;        // Output
    output[1]   = out_q;
    //Debug Channel
    debug[0]     =0;
    debug[1]     =0;
}


/* Function: mdlTerminate =================================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
}


#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

6.2.2 Function abc_dq_trans

```
/**********************************/
/* C-Functions, integrated in Corpus s_transformations.c */
/**********************************/

#include "transformations.h"
// transformations.h contains
//      - Constants
//      - Integration of further standard functions (math.h...)
//      - Initialization of functions


//  ----------------------- Variables -----------------------------------
float   in_alpha, in_beta;                // States: Induced voltage Us-RsIs
float    cosGamma,sinGamma;           // Variable for calculation Cos / Sin (gamma)
//  -------------------------------------------------------------------

// Initializations
// is called automatically at the start of simulation.
// Parameter:
void init_var( float fsa) //The hand-over of fsa and the initialization of the variables in this file only as an example
{
    // reset old states
    in_alpha      = 0;
    in_beta       = 0;

}

//  -------------------------------------------------------------------

// abc_dq_transformation
// Paramter: - keine

void abc_dq_trans(float in_a, float in_b, float in_c, float gamma, double *out_d, double *out_q ){

      // calc Alpha- Beta-Components (Step 1)
           in_alpha = TWOTHIRD*(in_a - 0.5*(in_b+in_c));
```

```
            in_beta   = SQRT3RECIP*(in_b-in_c);
        // calc dq-Components (Step 2)
            cosGamma = cos(gamma);
            sinGamma = sin(gamma);
            *out_d = in_alpha * cosGamma + in_beta  * sinGamma;
            *out_q = in_beta  * cosGamma - in_alpha * sinGamma;
}
```

6.2.3 Required header file

```
// HEADER FILE
#include <math.h>
#define TWOTHIRD   0.666666666666667
#define SQRT3RECIP 0.577350269189626
// ------------------------ Prototypes -------------------------------------

// Initializations
// is called automatically at the start of simulation
// Parameter:
void init_var(float fsa);
//
//
// Is called with the frequency set in the Simulink mask.
// Parameter:
void abc_dq_trans(float in_a, float in_b, float in_c, float gamma, double *out_d, double *out_q);
```

7. References

[1] FUCHS, F. W.: *Skript: Regelung elektrischer Antriebe*, 2011. Lehrstuhl für Leistungselektronik und Elektrische Antriebe, Christian-Albrechts-Universität zu Kiel.

[2] K. KÜPFMÜLLER, W. MATHIS and A. REIBIGER: *Theoretische Elektrotechnik - Eine Einführung*, volume 18. Springer-Verlag, 2008.

[3] SCHRÖDER, D.: *Elektrische Antriebe - Regelung von Antriebssystemen*. Springer, 3 edition, 2009.

[4] G. F. FRANKLIN, J. D. POWELL and M. WORKMAN: *Digital Control of Dynamic Systems*, volume 3. Ellis-Kagle Press, 2006.

[5] LUTZ, H. and W. WENDT: *Taschenbuch der Regelungstechnik*, volume 7. Verlag Harri Deutsch, 2007.

[6] THE MATHWORKS, INC.: *Simulink - Writing S-Functions, Version 3*. The MathWorks, Inc., 1998.

[7] THE MATHWORKS, INC.: *Simulink - Developing S-Functions*. The MathWorks, Inc., 2012.

[8] BOHN, C. and D.P. ATHERTON: *An analysis package comparing PID anti-windup strategies*. Control Systems, IEEE, 15(2):34 –40, Apr. 1995.