CTU

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Department of Measurement

Master's thesis

Sensorless Field Oriented Control of PMSM Based on STSPIN32G4

Bc. Kristián Šlehofer

May 2024

Supervisor: Ing. Jan Stejskal

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Šlehofer Kristián** Personal ID number: **483516**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Open Informatics**

Specialisation: **Computer Engineering**

II. Master’s thesis details

Master’s thesis title in English:

Sensorless Field Oriented Control of PMSM Based on STSPIN32G4

Master’s thesis title in Czech:

Bezsenzorové vektorové řízení PMSM pomocí STSPIN32G4

Guidelines:

1) Develop a mathematical model of the PMSM.
2) Perform a study of sensorless algorithms for PMSM control.
3) Build a simulation model of the selected sensorless FOC algorithm for PMSM control.
4) Implement the algorithm on the EVSPIN32G4 evaluation board.

Bibliography / sources:

[1] R. Krishnan, 'Permanent Magnet Synchronous and Brushless DC Motor Drives,' 1st edition, CRC Press, 616 p., 2010, ISBN 978-0-8247-5384-9
[2] G. F. Franklin, 'Feedback Control of Dynamic Systems,' 8th Edition, Pearson, 2018, ISBN-13: 978-0134685717
[3] H. K. Khalil, 'Nonlinear Systems,' Third edition, Prentice Hall, 2002, ISBN 0-13-067389-7
[4] STMicroelectronics, 'High performance 3-phase motor controller with embedded STM32G4 MCU,' DS13630 Datasheet [on-line]

Name and workplace of master’s thesis supervisor:

Ing. Jan Stejskal Department of Electric Drives and Traction FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **14.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until:
by the end of summer semester 2024/2025

Ing. Jan Stejskal Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

. | Date of assignment receipt

Student’s signature

Declaration

I declare that the presented work was developed independently and that I have listed all sources of the information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 24, 2024

. . . . . .
Bc. Kristián Šlehofer

Acknowledgements

I would like to express my deepest appreciation to my thesis supervisor Ing. Jan Stejskal for his advices, friendly approach and sharing of valuable experiences, without which I would not have been able to solve many problems. Furthermore, I would like to thank STMicroelectronics for providing me with the necessary hardware and test instruments. Last but not least, I am also very grateful mainly to my family and all my friends for supporting me and keeping me sane during my studies.

Abstract

This thesis presents a sensorless field-oriented control (FOC) algorithm for permanent magnet synchronous motors (PMSMs) and its implementation on the STSPIN32G4-based evaluation board. The objective is to develop an efficient control method without relying on position sensors. The study begins with a mathematical model of the PMSM, followed by an investigation of various sensorless control algorithms. A simulation model of the selected *Model Reference Adaptive System* (MRAS) method was created in MATLAB to validate its performance. The algorithm was then implemented on the *EVSPIN32G4* evaluation board, which was modified to improve current sensing resolution. Experimental results demonstrated that the sensorless FOC algorithm is stable and exhibits a significant degree of robustness to input parameter error. Furthermore, the whole control logic runs on a microcontroller, making it a viable option for cost-effective applications.

Keywords: PMSM, sensorless FOC, observer, embedded system, STSPIN32G4

Abstrakt

Tato práce prezentuje algoritmus bezsenzorového vektorového řízení (FOC) pro synchronní motory s permanentními magnety (PMSM) a jeho implementaci na evaluační desce založené na *STSPIN32G4*. Cílem je vyvinout účinnou metodu řízení bez nutnosti spoléhat se na snímače polohy. Studie začíná matematickým modelem PMSM, po jehož odvození následuje přehled různých algoritmů bezsenzorového řízení. V prostředí MATLAB byl vytvořen simulační model zvolené metody *Model Reference Adaptive System* (MRAS) a ověřena jeho funkčnost. Algoritmus byl poté implementován na evaluační desce *EVSPIN32G4*, která byla upravena pro zlepšení rozlišení snímání proudu. Na základě experimentů bylo prokázáno, že algoritmus je stabilní a vykazuje značnou míru odolnosti vůči chybě vstupních parametrů. Celá řídicí logika navíc běží na mikrokontroléru, což z ní činí výhodnou variantu pro aplikace zaměřené na nízkou cenu.

Klíčová slova: PMSM, bezsenzorové vektorové řízení, pozorovatel, vestavěný systém, STSPIN32G4

Překlad názvu: Bezsenzorové vektorové řízení PMSM pomocí STSPIN32G4

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Meaning
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
DMA	Direct Memory Access
DSP	Digital Signal Processing
EMF	Electromotive force
FOC	Field-Oriented Control
GPIO	General-Purpose Input/Output
IC	Integrated Circuit
IDE	Integrated Development Environment
I^2C	Inter-Integrated Circuit
MCU	Microcontroller Unit
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MRAS	Model Reference Adaptive System
PID	Proportional-Integral-Derivative
PMSM	Permanent Magnet Synchronous Motor
PWM	Pulse Width Modulation

List of Used Notation

Quantity	Meaning	Unit
$_{a,b,c}$	Subscript - quantity in abc reference frame	
$_{\alpha,\beta}$	Subscript - quantity in $\alpha\beta$ reference frame	
$_{d,q}$	Subscript - quantity in dq reference frame	
θ_m	Mechanical angle	rad
θ_e	Electrical angle	rad
ω_m	Mechanical angular velocity	s^{-1}
ω_e	Electrical angular velocity	s^{-1}
R	Stator resistance	Ω
L	Stator inductance	H
Ψ_a, Ψ_b, Ψ_c	Total flux linking each stator winding	Wb
$\Psi_{a\mathrm{m}}, \Psi_{b\mathrm{m}}, \Psi_{c\mathrm{m}}$	Permanent magnet fluxes linking stator windings	Wb
Ψ_PM	Permanent magnet flux linkage	Wb
Ψ_q	q-axis flux linkage	Wb
p_p	Number of motor pole pairs	-
J	Moment of inertia	$\mathrm{kg\,m^2}$
P_gap	Air gap power	W
T_em	Motor electromechanical torque	$\mathrm{N\,m}$
T_load	Load torque	$\mathrm{N\,m}$
e	Back-EMF	V
u_a, u_b, u_c	Voltages in *abc* reference frame	V
i_a, i_b, i_c	Phase currents in *abc* reference frame	A
u_α, u_β	Voltages in $\alpha\beta$ reference frame	V
i_α, i_β	Phase currents in $\alpha\beta$ reference frame	A
u_d, u_q	Voltages in *dq* reference frame	V
i_d, i_q	Phase currents in *dq* reference frame	A
k_v	Back-EMF constant	$\mathrm{rpm\,V^{-1}}$
k_e	Back-EMF constant in SI units	$\mathrm{V\,s\,rad^{-1}}$
f_sw	Switching frequency	Hz

Chapter 1
Introduction

Thanks to continuous industrialization, automation, and the recent massive boom in electromobility, Permanent Magnet Synchronous Motors (PMSM) are widely popular and used in various segments. PMSMs are known for their high performance and high power density.

However, knowing the exact position of the rotor is essential for proper control and thus achieving high efficiency. Position information is commonly obtained from position sensors, such as encoders or resolvers, that are mounted on the rotor shaft. Not only do these sensors increase the cost and size, but they also reduce the reliability and immunity to noise.

To overcome this handicap of PMSM, a great effort has been made to develop sensorless control techniques. Several approaches have been proposed, each with its own advantages and disadvantages.

This thesis focuses on the development of one such sensorless field-oriented control algorithm and its implementation on real hardware. Based on the research, an algorithm using the Model Reference Adaptive System (MRAS) control scheme was selected.

First, a mathematical model of the PMSM is derived. Then, research and comparison of available sensorless techniques is performed, MRAS-based position estimation is developed, and the whole sensorless solution is tested in simulations. Consequently, the control software is implemented on the real hardware based on the *STSTPIN32G4* by STMicroelectronics and tested with a motor from Maxon. Finally, the last section summarizes the results and presents a proposal for possible future work.

Chapter 2
Mathematical model

In order to analyze and simulate the dynamics of a three-phase Permanent Magnet Synchronous Motor (PMSM), a mathematical model is required. To derive the model, the following assumptions are made:

- Windings are assumed to be identical and sinusoidally distributed.
- Resistance and inductance are assumed to be constant.
- Change in inductance due to the rotor position is neglected.
- Hysteresis losses and eddy current losses are neglected.

2.1 Initial description

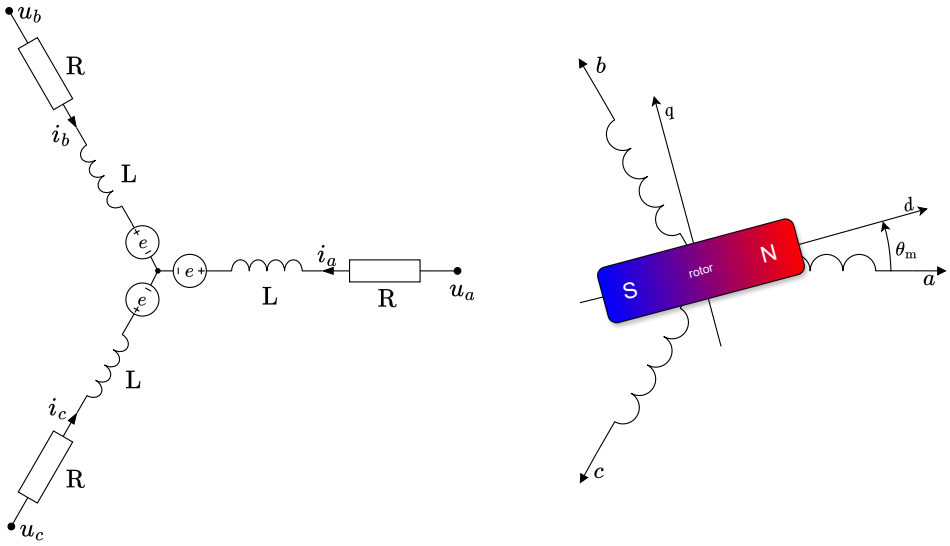

Figure 1: Analytical model of PMSM and rotor

Figure 1 shows the simplified electrical circuit of the wye-wound stator. Windings are assumed to be identical and sinusoidally distributed. They define a three-phase coordinate system abc in the two-dimensional space with axes shifted by 120 degrees.

Mechanical angle θ_m is defined as an angle between the a-phase magnetic axis and the d-axis of the rotor and refers to the angle of the rotor shaft with respect to the stator. The relation to the electrical angle θ_e is given by:

$$\theta_\mathrm{m} = p_\mathrm{p}\, \theta_\mathrm{e}\ , \tag{1}$$

where p_p is the number of pole pairs.

Since the wye-wound PMSM in a fault-free condition is a balanced system, both stator currents and stator voltages are in equilibrium:

$$i_a + i_b + i_c = 0 \tag{2}$$
$$u_a + u_b + u_c = 0 \tag{3}$$

Voltage equations for particular phases are:

$$u_a = Ri_a + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_a \tag{4}$$
$$u_b = Ri_b + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_b \tag{5}$$
$$u_c = Ri_c + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_c \tag{6}$$

The following applies to the flux equations:

$$\Psi_a = Li_a + \Psi_{a\mathrm{m}} \tag{7}$$
$$\Psi_b = Li_b + \Psi_{b\mathrm{m}} \tag{8}$$
$$\Psi_c = Li_c + \Psi_{c\mathrm{m}} \tag{9}$$

The permanent magnet flux linking a particular phase is maximum when the rotor d-axis aligns with the phase magnetic axis and zero when perpendicular. Therefore, the following holds for the linked motor flux:

$$\Psi_{a\mathrm{m}} = \Psi_\mathrm{PM} \cos\left(\theta_\mathrm{e}\right) \tag{10}$$
$$\Psi_{b\mathrm{m}} = \Psi_\mathrm{PM} \cos\left(\theta_\mathrm{e} - \frac{2\pi}{3}\right) \tag{11}$$
$$\Psi_{c\mathrm{m}} = \Psi_\mathrm{PM} \cos\left(\theta_\mathrm{e} + \frac{2\pi}{3}\right) \tag{12}$$

where Ψ_PM is permanent magnet flux linkage.

Motor manufacturers often do not provide the value of permanent magnet flux linkage in their datasheets. The induced back-electromotive force (back-EMF) is equal to the rate of flux linkages [1]. Therefore, it is possible to use an alternative flux linkage parametrization using the back-EMF constant, which is proportional to motor pole pairs and permanent magnet flux linkage and is usually provided in the datasheet.

If the constant is in SI units [$\mathrm{V\,s\,rad^{-1}}$], the following applies:

$$k_\mathrm{e} = p_\mathrm{p} \frac{\mathrm{d}}{\mathrm{d}t}\left(\Psi_\mathrm{PM}\right) \tag{13}$$

To complete the model, the electromechanical relation is required. Assuming that the moment of inertia is constant, mechanical dynamics of the PMSM can be expressed as

$$\sum T_\mathrm{tot} = T_\mathrm{em} - T_\mathrm{load} = J\frac{\mathrm{d}}{\mathrm{d}t}\omega_\mathrm{m}\ , \tag{14}$$

where T_em is the motor electromechanical torque and T_load is the load torque, including the friction torque caused by the load.

2.1.1 Model in *abc* reference frame

By substitution of equations (7) - (12) into the voltage equations (4) - (6), we obtain the complete electrical model of the PMSM:

$$u_a = Ri_a + L\frac{\mathrm{d}}{\mathrm{d}t}i_a + \frac{\mathrm{d}}{\mathrm{d}t}\left(\Psi_{\mathrm{PM}}\cos\left(\theta_{\mathrm{e}}\right)\right) \tag{15}$$

$$u_b = Ri_b + L\frac{\mathrm{d}}{\mathrm{d}t}i_b + \frac{\mathrm{d}}{\mathrm{d}t}\left(\Psi_{\mathrm{PM}}\cos\left(\theta_{\mathrm{e}} - \frac{2\pi}{3}\right)\right) \tag{16}$$

$$u_c = Ri_c + L\frac{\mathrm{d}}{\mathrm{d}t}i_c + \frac{\mathrm{d}}{\mathrm{d}t}\left(\Psi_{\mathrm{PM}}\cos\left(\theta_{\mathrm{e}} + \frac{2\pi}{3}\right)\right) \tag{17}$$

After differentiating the last term representing the back-EMF in equations (15) - (17), we can substitute the equation (13) and obtain the alternative model:

$$u_a = Ri_a + L\frac{\mathrm{d}}{\mathrm{d}t}i_a - \frac{\mathrm{d}}{\mathrm{d}t}\left(\Psi_{\mathrm{PM}}\right)\cdot\omega_{\mathrm{e}}\sin\left(\theta_{\mathrm{e}}\right) = Ri_a + L\frac{\mathrm{d}}{\mathrm{d}t}i_a - k_{\mathrm{e}}\omega_{\mathrm{m}}\sin\left(\theta_{\mathrm{e}}\right) \tag{18}$$

$$u_b = Ri_b + L\frac{\mathrm{d}}{\mathrm{d}t}i_b - \frac{\mathrm{d}}{\mathrm{d}t}\left(\Psi_{\mathrm{PM}}\right)\cdot\omega_{\mathrm{e}}\sin\left(\theta_{\mathrm{e}} - \frac{2\pi}{3}\right) = Ri_b + L\frac{\mathrm{d}}{\mathrm{d}t}i_b - k_{\mathrm{e}}\omega_{\mathrm{m}}\sin\left(\theta_{\mathrm{e}} - \frac{2\pi}{3}\right) \tag{19}$$

$$u_c = Ri_c + L\frac{\mathrm{d}}{\mathrm{d}t}i_c - \frac{\mathrm{d}}{\mathrm{d}t}\left(\Psi_{\mathrm{PM}}\right)\cdot\omega_{\mathrm{e}}\sin\left(\theta_{\mathrm{e}} + \frac{2\pi}{3}\right) = Ri_c + L\frac{\mathrm{d}}{\mathrm{d}t}i_c - k_{\mathrm{e}}\omega_{\mathrm{m}}\sin\left(\theta_{\mathrm{e}} + \frac{2\pi}{3}\right) \tag{20}$$

Hence, the back-EMF for one phase is:

$$e = -k_{\mathrm{e}}\omega_{\mathrm{m}} \tag{21}$$

The electromechanical torque T_{em} is defined as the power transmitted through the air gap divided by the mechanical angular velocity of the rotor. We can write:

$$T_{\mathrm{em}} = \frac{P_{\mathrm{gap}}}{\omega_{\mathrm{m}}} = \frac{e_a i_a + e_b i_b + e_c i_c}{\omega_{\mathrm{m}}} \tag{22}$$

2.2 Clarke transformation

In electrical engineering, Clarke transformation (or the alpha-beta transformation) is a mathematical transformation utilized to analyze three-phase systems. It transforms the three-phase reference frame (*abc*) into a two-phase orthogonal stationary reference frame ($\alpha\beta$), reducing the number of equations and, thus, the analysis complexity of the system.

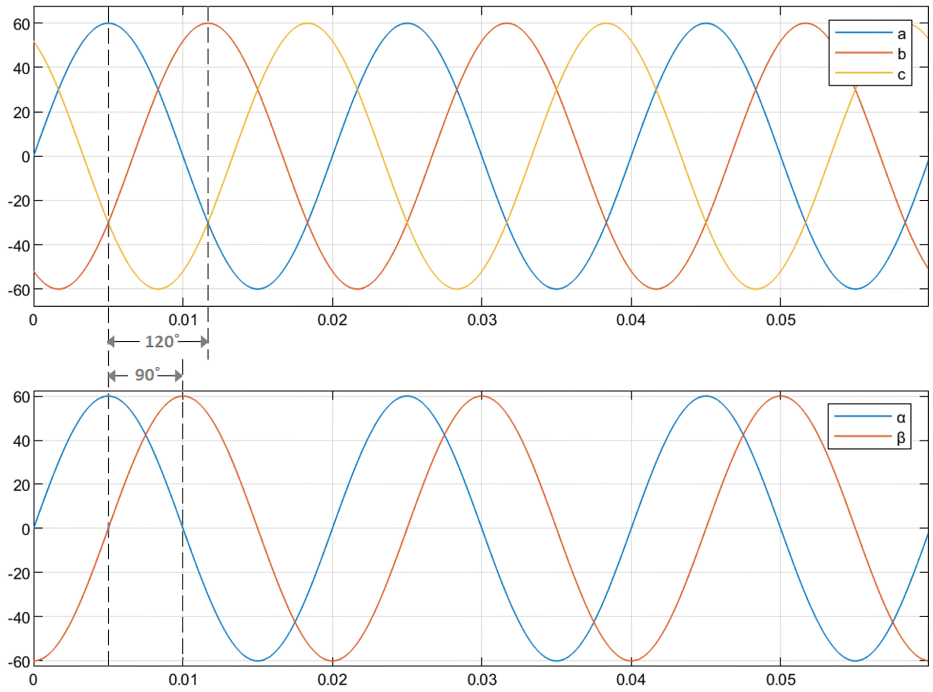

Figure 2: Clarke transformation [2]

The alpha-beta transformation is defined as:

$$\begin{bmatrix} x_\alpha \\ x_\beta \\ x_0 \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} \tag{23}$$

where zero component x_0 equals zero in balanced systems.

The inverse Clark transformation is used to convert a two-phase reference frame back to a three-phase reference frame:

$$\begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 1 \end{bmatrix} \begin{bmatrix} x_\alpha \\ x_\beta \\ x_0 \end{bmatrix} \tag{24}$$

This version of the Clarke transform is amplitude-invariant, which means it preserves the amplitude of the electrical variables to which it is applied. The Clarke transform also exists in a power-invariant version, which, on the other hand, preserves the active and reactive powers, but current amplitudes are not the same as those in the *abc* reference frame. The power-invariant version is similar to the amplitude-invariant and is defined as:

$$\begin{bmatrix} x_\alpha \\ x_\beta \\ x_0 \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} \tag{25}$$

2.2.1 Model in $\alpha\beta$ reference frame

Applying the transformation (25) on voltage equations (18) - (20) and electromechanical torque equation T_{em}, we obtain the PMSM model in the $\alpha\beta$ coordinate system:

$$u_\alpha = Ri_\alpha + L\frac{\mathrm{d}}{\mathrm{d}t}i_\alpha - k_\text{e}\omega_\text{m}\sin(\theta_\text{e}) \tag{26}$$

$$u_\beta = Ri_\beta + L\frac{\mathrm{d}}{\mathrm{d}t}i_\beta + k_\text{e}\omega_\text{m}\cos(\theta_\text{e}) \tag{27}$$

$$T_\text{em} = \frac{3}{2}k_\text{e}\left[\cos(\theta_\text{e})i_\beta - \sin(\theta_\text{e})i_\alpha\right] \tag{28}$$

2.3 Park transformation

Another crucial mathematical transformation used in electrical engineering is the Park transformation. It serves to simplify the analysis and subsequent control by transforming the two-phase stationary $\alpha\beta$ reference frame into a two-phase rotating dq reference frame. The transformed quantities appear stationary if the reference frame rotates at the same speed as the rotor.

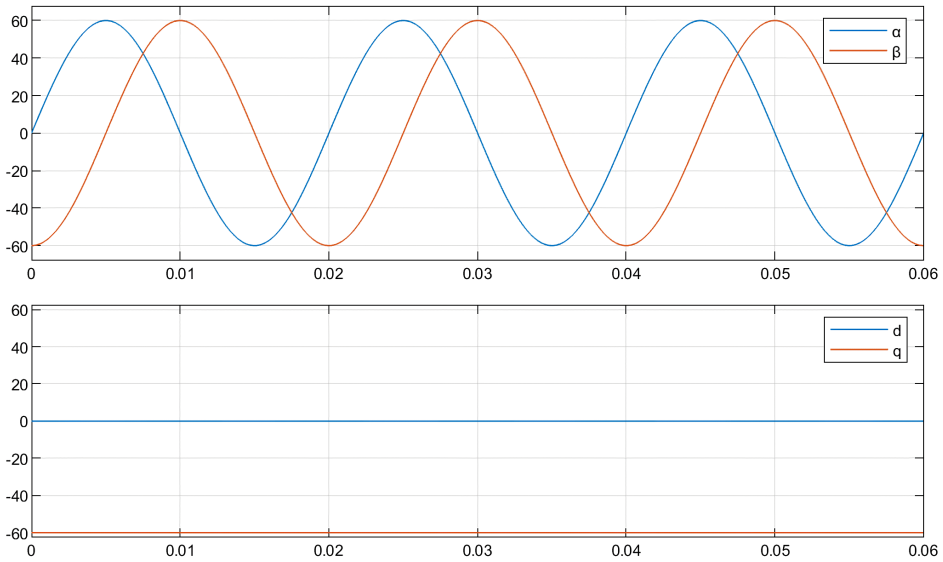

Figure 3: Park transformation [3]

The transformation equations are given by:

$$\begin{bmatrix} x_d \\ x_q \end{bmatrix} = \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix} \tag{29}$$

where ϕ is an angle between the α- and d-axes.

The inverse Park transformation is used to convert the rotating reference frame back to the stationary reference frame:

$$\begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} x_d \\ x_q \end{bmatrix} \tag{30}$$

2.3.1 Model in *dq* reference frame

Applying the transformation (29) on voltage equations (26), (27) and electromechanical torque equation (28) in $\alpha\beta$ coordinate system, we obtain the PMSM model in the *dq* coordinate system rotating with the same speed as the rotor:

$$u_d = Ri_d + L\frac{\mathrm{d}}{\mathrm{d}t}i_d - L\omega_\mathrm{e}i_q \tag{31}$$

$$u_q = Ri_q + L\frac{\mathrm{d}}{\mathrm{d}t}i_q + L\omega_\mathrm{e}i_d + k_\mathrm{e}\omega_\mathrm{m} \tag{32}$$

$$T_\mathrm{em} = \frac{3}{2}\left(k_\mathrm{e}i_q - p_\mathrm{p}\Psi_q i_d\right) \tag{33}$$

Chapter 3
Sensorless FOC

3.1 Overview

High efficiency, reliability, and high linearity between current and torque made PMSMs favored and widely used in many applications. However, these characteristics greatly depend on the correct control method and its accuracy.

Among the most straightforward control techniques, it is possible to use a simple six-step commutation scheme that is used primarily to control three-phase brushless DC (BLDC) motors. This method energizes two legs of the inverter, and the third one is left floating to measure the back-EMF and find the commutation point to advance to the next step. However, the simplicity of the method is redeemed by the introduced torque and current ripple, which can be unsuitable for applications requiring precise control. As the name suggests, this control scheme uses only six voltage vectors to create the rotating field. Therefore, the control achieves the torque angle of 90 degrees but with a deviation of 30 degrees, resulting in a significant torque ripple.

Field-Oriented Control (FOC), also called Vector control, solves this problem by continuously controlling the rotating voltage vector. Nevertheless, achieving precise control of PMSM requires accurate knowledge of the rotor position. Traditionally, this information is obtained through position sensors such as encoders or resolvers. However, the integration of such sensors increases system complexity, cost, and susceptibility to wear over time [4].

Sensorless FOC aims to eliminate the physical position sensor and deduce the rotor position without it, allowing for a more robust control strategy. Various control methods have been proposed to eliminate speed and position sensors. They can be generally classified into two types: signal injection methods and methods using the fundamental components of voltage and current signals.

3.1.1 Signal injection methods

Methods based on signal injection exploit the effects of PMSM magnetic saturation or rotor saliency. In the high-frequency injection method, a high-frequency voltage or current vector signal is superimposed on motor fundamental excitation. The corresponding high-frequency current (or voltage) signal contains rotor position information and is analyzed to track spatial saliencies and estimate the rotor or flux position [5].

Methods based on signal injection are effective at a standstill and in low-speed ranges because the amplitude of high-frequency signals used for position estimation does not depend on rotating velocity [6]. However, additional superimposed test signals can introduce unwanted audible noises, motor micromovements, or twitching, and the performance is limited at high speeds. It is mainly because the stator resistance voltage and the rotating voltage cannot be ignored at high speed, resulting

in the simplified model of high-frequency signal injection can no longer be applied. Furthermore, these methods do not work well with surface-mounted PMSM with low or zero magnetic saliency and require complex signal processing [7].

The high-frequency injection is utilized to detect rotor position at low speeds, for example, in an open-source VESC project [8].

Even though the injection usually uses high-frequency signals, some methods use low-frequency injection to estimate the rotor position. The advantage of the low-frequency signal injection is the reduction of loss and harsh acoustic noise. However, it creates new problems, and the filter for signal separation could lead to instability and degradation of the current control and position estimation performance [9].

3.1.2 Methods using the fundamental components

Latter methods based on fundamental components of control signals mostly employ the PMSM model and motor parameters to estimate the position. These methods perform best at medium and high speeds and do not generate any ripple or audible noises due to the injection of additional signals.

However, model-based methods tend to be sensitive to parameter variations since the motor parameters are required for the PMSM model. These parameters depend on the operating conditions, and this, in turn, degrades the control performance and causes an estimation position error. The voltage references are usually used for voltage information, but the reference and actual voltages differ because of the dead-time of the pulsewidth-modulated inverter and the voltage drop in the switching devices [10]. In the low-speed region, the estimate is erroneous and usually cannot be used for control. Regardless, in most industrial applications (fans, pumps, compressors, etc.), the motor is in the low-speed region only temporarily during the start, and thus, this disadvantage can be overcome by a suitable startup method.

Many fundamental component approaches were presented; some are based on the estimation of the back-EMF or flux linkage due to permanent magnets containing position information. For this reason, a Luenberger state observer, sliding-mode observer, or various Kalman filters are used [11][12][13]. These methods usually utilize the PMSM model in the $\alpha\beta$ coordinate system and extract the position using goniometric functions. Some employ the more accurate Extended EMF model with fewer approximations, notably benefiting interior PMSMs. The Extended EMF includes position information from not only the traditionally defined EMF but also from the stator inductance [14]. Kalman filter-based methods require complex computational operations to obtain proper accuracy, which always causes problems in real time operation [15].

For instance, STMicroelectronics' Motor Control SDK employs the Luenberger observer as a sensorless solution [16].

Other methods are based on the voltage or current error between the measured and calculated variables from the motor model. This information is either directly used or processed by an observer or an appropriate adaptation mechanism [17][18]. A linear feedback control technique where it is desirable to obtain a closed-loop system whose behavior is described by a reference model is called adaptive control [19]. One such method utilizes a Model Reference Adaptive System (MRAS) control scheme, which works with a reference model (the motor) and an adaptive model. Motor input voltages are also supplied into the adaptive model with PMSM equations. The current response from both

systems is then compared, and using an adaptation law based on Popov superstability, the angle and speed are estimated. The estimate has good convergence, the computational complexity is moderate, and it is convenient for engineering realization [20]. Also, it offers robustness against parameter variations and has a good speed response ranging from low to high speed [21].

Due to its simplicity and potential, the MRAS observer technique was selected for further investigation.

3.2 MRAS

Model Reference Adaptive System (MRAS) is a control method that compares outputs of two systems to make an uncertain controlled system track the behavior of a given reference plant model. The MRAS algorithm is well-known in the sensorless control of an induction motor, and has been proved to be effective and physically clear [22][23].

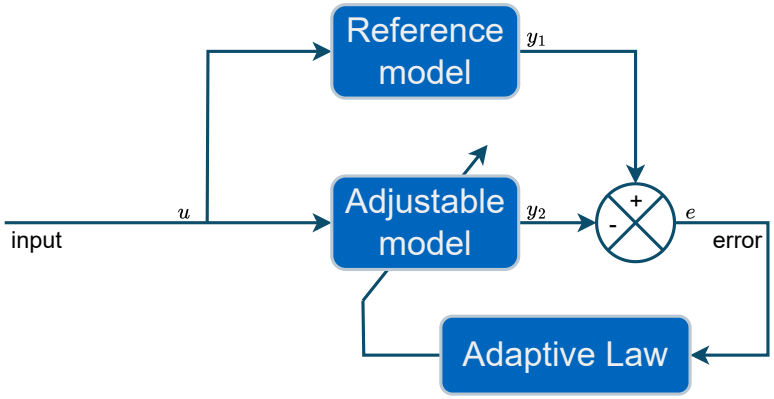

Figure 4: Block diagram of MRAS

The Model Reference Adaptive System structure consists of a reference model, an adaptive model with the same physical meaning, and a suitable adaptive law. The structure is visualized in Figure 4. The reference and adaptive models are connected in parallel and excited with the same input signals. The error between the estimated quantities obtained by the two models is used to drive a suitable adaptation mechanism that generates the estimated rotor speed.

The controlled PMSM itself is selected as a reference model. The adaptive model is derived from current equations in (31) and (32). First, the equations are rewritten into a matrix representation as follows:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_d \\ i_q \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & \omega_\mathrm{e} \\ -\omega_\mathrm{e} & -\frac{R}{L} \end{bmatrix}\begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & \frac{1}{L} \end{bmatrix}\begin{bmatrix} u_d \\ u_q \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{L}\frac{k_\mathrm{e}}{p_\mathrm{p}}\omega_\mathrm{e} \end{bmatrix} \tag{34}$$

This representation can be further simplified to resemble the standard form [24]:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_d + \frac{1}{L}\frac{k_\mathrm{e}}{p_\mathrm{p}} \\ i_q \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & \omega_\mathrm{e} \\ -\omega_\mathrm{e} & -\frac{R}{L} \end{bmatrix}\begin{bmatrix} i_d + \frac{1}{L}\frac{k_\mathrm{e}}{p_\mathrm{p}} \\ i_q \end{bmatrix} + \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & \frac{1}{L} \end{bmatrix}\begin{bmatrix} u_d + \frac{R}{L}\frac{k_\mathrm{e}}{p_\mathrm{p}} \\ u_q \end{bmatrix} \tag{35}$$

Now define new state variables as:

$$i'_d = i_d + \frac{1}{L}\frac{k_\mathrm{e}}{p_\mathrm{p}} \tag{36}$$

$$i'_q = i_q \tag{37}$$

$$u'_d = u_d + \frac{R}{L}\frac{k_\mathrm{e}}{p_\mathrm{p}} \tag{38}$$

$$u'_q = u_q \tag{39}$$

Then, the model can be expressed as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i'_d \\ i'_q \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & \omega_\mathrm{e} \\ -\omega_\mathrm{e} & -\frac{R}{L} \end{bmatrix}\begin{bmatrix} i'_d \\ i'_q \end{bmatrix} + \frac{1}{L}\begin{bmatrix} u'_d \\ u'_q \end{bmatrix} \tag{40}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} i' = Ai' + Bu' \tag{41}$$

Because the reference model and the adjustable model outputs have the same physical meaning, the adjustable model can be expressed using the estimated values:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \hat{i}'_d \\ \hat{i}'_q \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & \hat{\omega}_\mathrm{e} \\ -\hat{\omega}_\mathrm{e} & -\frac{R}{L} \end{bmatrix}\begin{bmatrix} \hat{i}'_d \\ \hat{i}'_q \end{bmatrix} + \frac{1}{L}\begin{bmatrix} u'_d \\ u'_q \end{bmatrix} \tag{42}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} \hat{i}' = \hat{A}\hat{i}' + Bu' \tag{43}$$

In order to design a robust adaptation law, the Popov superstability theory is used. According to the theory of Popov [25], if the feedback system is stable, it must meet the following two aspects:

1. The nonlinear time-varying feedback link satisfies the Popov integral inequality:

$$\eta(0,t) = \int_0^t v^T \omega \,\mathrm{d}\tau \geq -\gamma^2 \,, \qquad \forall t \geq 0 \,, \tag{44}$$

   where γ^2 can be any finite positive real number.

2. Transfer matrix $H(s) = C(sI - A)^{-1}$ is strictly positive definite matrix.

The Popov integral inequality is solved by the reverse procedure. At the initial state, the error is close to zero ($\lim_{t\to 0} e(t) = 0$), and the estimated speed is convergent to the actual value, so the MRAS is asymptotically stable. The specific process of reverse solution is as follows:

$$v = e_\mathrm{i} = i' - \hat{i}' \tag{45}$$

$$\omega = (\hat{\omega}_\mathrm{e} - \omega_\mathrm{e})\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} \hat{i}'_d \\ \hat{i}'_q \end{bmatrix} = J(\hat{\omega}_\mathrm{e} - \omega_\mathrm{e})\hat{i}' \tag{46}$$

Substitute (45) and (46) into Popov integral inequality equation (44):

$$\eta(0,t) = \int_0^t e_\mathrm{i}^T J\,(\hat{\omega}_\mathrm{e} - \omega_\mathrm{e})\,\hat{i}' \,\mathrm{d}\tau \geq -\gamma^2 \tag{47}$$

According to the PI control strategy of MRAS, the adaptive law is obtained as follows:

$$\hat{\omega}_\mathrm{e} = \int_0^t F_1(v,t,\tau)\,\mathrm{d}\tau + F_2(v,t) + \hat{\omega}_\mathrm{e}(0) \,, \tag{48}$$

where $\hat{\omega}_{\mathrm{e}}(0)$ is the initial value.

After deformation and calculation, the adaptation law is obtained:

$$\hat{\omega}_{\mathrm{e}} = \int_0^t k_{\mathrm{i}} \left[i_d \hat{i}_q - \hat{i}_d i_q - \frac{1}{L} \frac{k_{\mathrm{e}}}{p_{\mathrm{p}}} \left(i_q - \hat{i}_q \right) \right] \mathrm{d}\tau + k_{\mathrm{p}} \left[i_d \hat{i}_q - \hat{i}_d i_q - \frac{1}{L} \frac{k_{\mathrm{e}}}{p_{\mathrm{p}}} \left(i_q - \hat{i}_q \right) \right] + \hat{\omega}_{\mathrm{e}}(0) \, , \tag{49}$$

where k_{p} and k_{i} are the PI controller's proportional and integral term coefficients. Then, the estimated rotor position is:

$$\hat{\theta}_e = \int \hat{\omega}_{\mathrm{e}} \, d\tau \tag{50}$$

The complete structure of the MRAS-based speed estimator is in Figure 5, where err is the error term given by

$$err = i_d \hat{i}_q - \hat{i}_d i_q - \frac{1}{L} \frac{k_{\mathrm{e}}}{p_{\mathrm{p}}} (i_q - \hat{i}_q) \, . \tag{51}$$

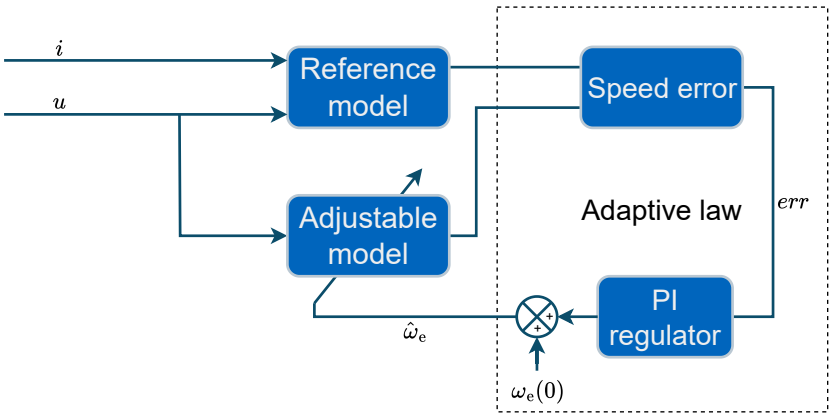

Figure 5: MRAS-based speed estimator

Chapter 4
Simulation

4.1 Environment

MATLAB is a programming and numeric computing platform used by millions of engineers and scientists to analyze data, develop algorithms, and create models [26]. It is also widely used in academia regarding signal and image processing, filter design, control design, robotics, and various simulations and computations. The most powerful aspect of MATLAB is its almost infinite extensibility with additional packages, toolkits, or environments, such as the MATLAB Simulink.

Simulink is a block diagram environment used to design systems with multidomain models, simulate before moving to hardware, and deploy without writing code [27]. It is a multi-purpose graphical tool that can be used for model-based design and simulation, ranging from simple problems to complex simulation models. Simulink shares the same degree of extensibility with the entire MATLAB suite.

One such mentionable additional package is Simscape, namely Simscape Electrical. Simscape enables the rapid creation of models of physical systems within the Simulink environment. It is possible to model systems such as electric motors, bridge rectifiers, hydraulic actuators, and refrigeration systems, by assembling fundamental components into a schematic. Simscape add-on products provide more complex components and analysis capabilities [28]. Simscape Electrical provides prepared electrical blocks and many examples, which could be used to test the derived models and compare the correctness and accuracy of the solution in the early phases of development.

4.2 PMSM model

In order to simulate the derived PMSM model in MATLAB Simulink, it is first necessary to rewrite the differential equations to the standard form. Both the $\alpha\beta$ and *dq* models were implemented and verified against the Simscape PMSM block. It is also possible to form the model in the *abc* reference frame. However, the complexity of the model is unnecessarily higher and, therefore, is unsuitable for implementation in a microcontroller.

To retrieve equations suitable to implement the model in the $\alpha\beta$ reference frame, rewrite equations (26) - (28) and (14) as:

$$\frac{\mathrm{d}}{\mathrm{d}t}i_\alpha = \dot{i_\alpha} = -\frac{R}{L}i_\alpha + \frac{1}{L}k_\mathrm{e}\omega_\mathrm{m}\sin(\theta_\mathrm{e}) + \frac{1}{L}u_\alpha \tag{52}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}i_\beta = \dot{i_\beta} = -\frac{R}{L}i_\beta - \frac{1}{L}k_\mathrm{e}\omega_\mathrm{m}\cos(\theta_\mathrm{e}) + \frac{1}{L}u_\beta \tag{53}$$

$$\dot{\omega}_\mathrm{m} = \frac{1}{J}\frac{3}{2}k_\mathrm{e}\left[\cos(\theta_\mathrm{e})i_\beta - \sin(\theta_\mathrm{e})i_\alpha\right] + \frac{1}{J}T_\mathrm{load} \tag{54}$$

$$\dot{\theta}_\mathrm{m} = \omega_\mathrm{m} \tag{55}$$

4.2 PMSM model

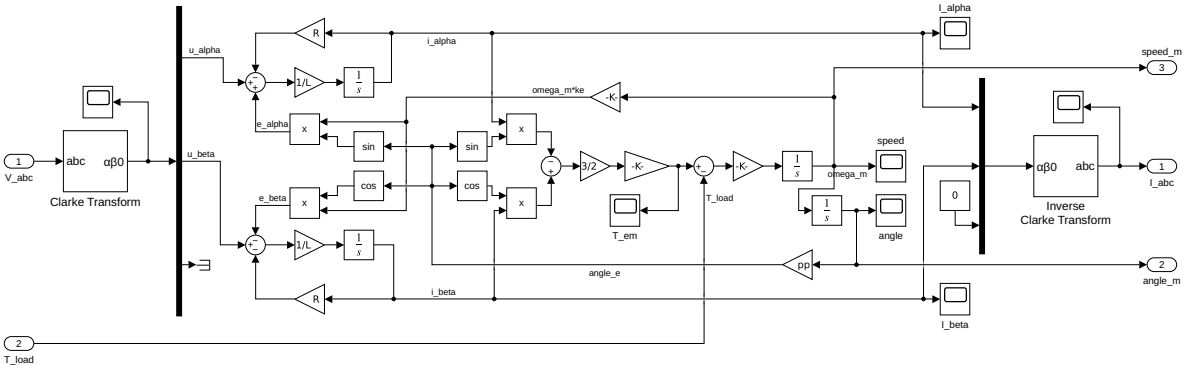

Figure 6: Simulink model in $\alpha\beta$ reference frame

To implement the model in the *dq* reference frame, the procedure is similar - rewrite the equations (31) and (32) as:

$$\frac{\mathrm{d}}{\mathrm{d}t} i_d = \dot{i_d} = -\frac{R}{L} i_d + \omega_\mathrm{e} i_q + \frac{1}{L} u_d \tag{56}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} i_q = \dot{i_q} = -\frac{R}{L} i_q - \omega_\mathrm{e} i_d - \frac{1}{L} k_\mathrm{e} \omega_\mathrm{m} + \frac{1}{L} u_q \tag{57}$$

$$\tag{58}$$

Then take (33) and (14) and assuming that $i_d = 0$, we can write:

$$\dot{\omega}_\mathrm{m} = \frac{1}{J} \frac{3}{2} k_\mathrm{e} i_q + \frac{1}{J} T_\mathrm{load} \tag{59}$$

$$\dot{\theta}_\mathrm{m} = \omega_\mathrm{m} \tag{60}$$

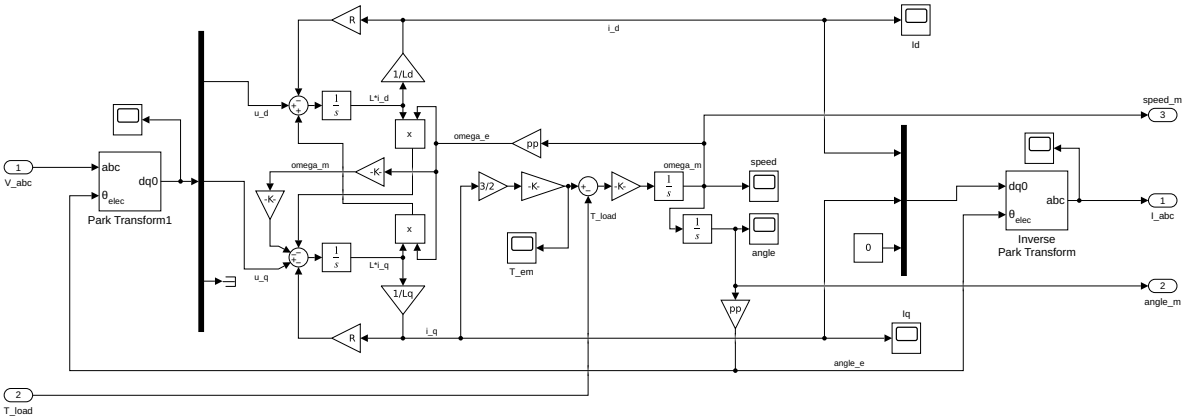

Figure 7: Simulink model in *dq* reference frame

Figures 6 and 7 show the $\alpha\beta$ and *dq* models. Inputs to both models are voltages u_α and u_β, respectively u_d and u_q; outputs are currents i_α and i_β, respectively i_d, i_q, and angular velocity ω_m. The angle is retrieved by integrating ω_m:

$$\theta_\mathrm{m} = \int \omega_\mathrm{m} \, \mathrm{d}t + \theta_\mathrm{m0} \, , \tag{61}$$

where θ_m0 is an initial angle.

4.3 Control

The complete sensorless control scheme was implemented in the MATLAB Simulink environment based on the derived equations. The block diagram is shown in Figure 8.

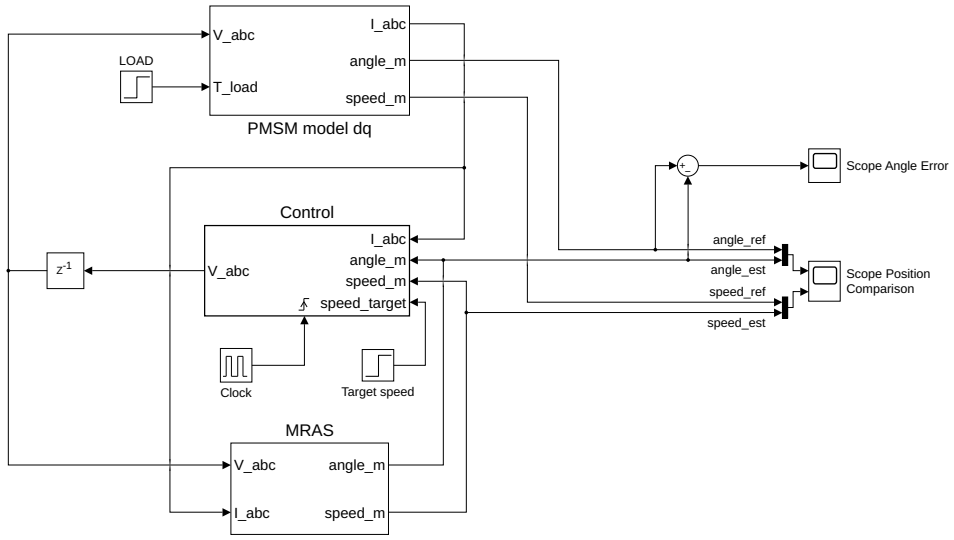

Figure 8: Block diagram of the sensorless FOC

The implementation of the field-oriented control is in Figure 9. It is realized as a triggered subsystem with switching frequency $f_{\mathrm{sw}} = 20\,000\,\mathrm{Hz}$. The control uses strategy, where the current i_d is controlled to zero.

In the simulation process, the difficulty of speed estimation is to select the appropriate PI parameters to obtain good dynamics and reduce the current noise of the whole system. In general, for different parameters of the motor, the adaptive law PI parameters are usually obtained by the trial and error method, first adjusting the proportional term and then adjusting the integral term [29].

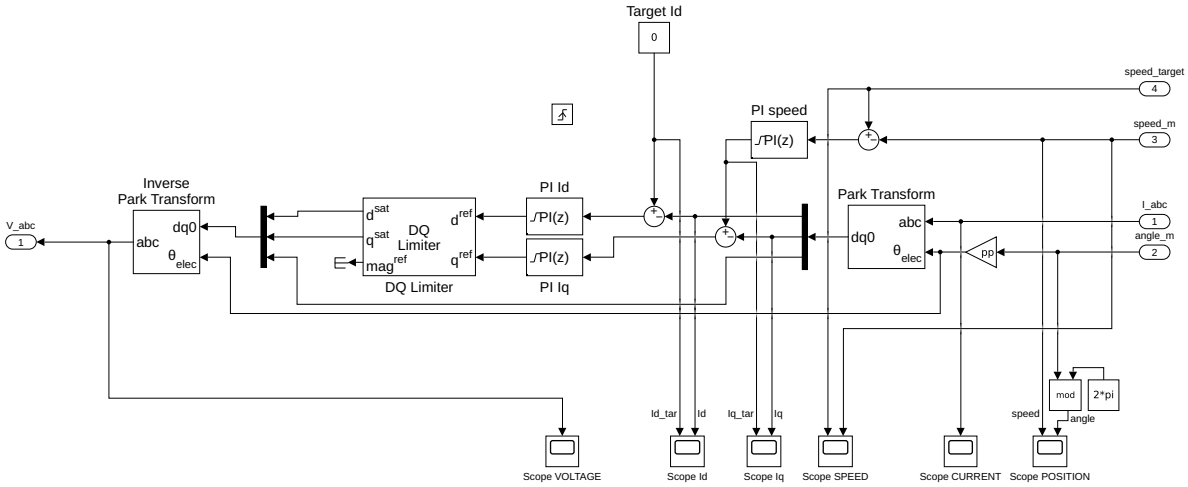

Figure 9: Field-oriented control subsystem

Figure 10 shows the implementation of the MRAS subsystem according to Section 3.2, with the

slight difference that the adaptive system is the current PMSM model in the $\alpha\beta$ reference frame given by equations (52) and (53). Currents are transformed into the *dq* reference frame afterwards.

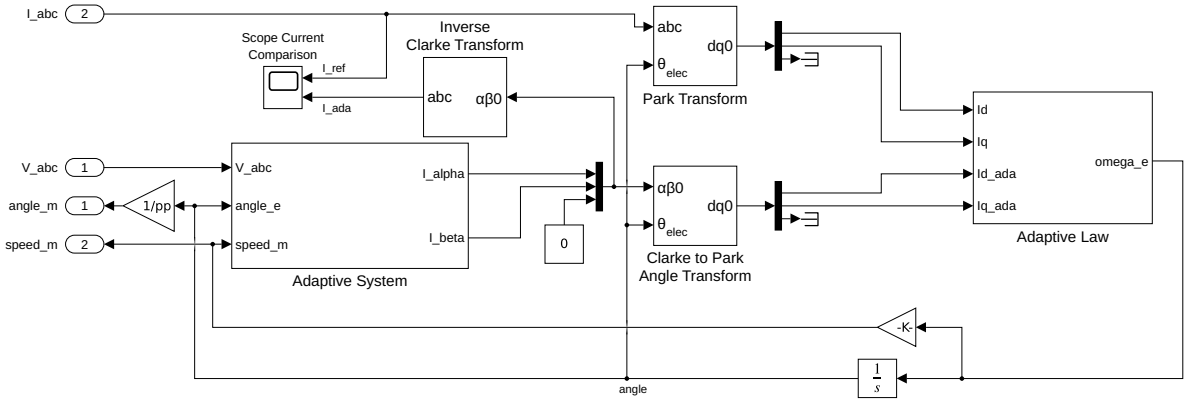

Figure 10: MRAS subsystem

The parameters used for the simulation are in the table 1. Parameters are slightly altered from the motor Maxon *EC-i 40* specification sheet, which was later used for the implementation [30].

Table 1: PMSM parameters used in simulation

Quantity	Meaning	Value
U	Nominal voltage	$48\,\mathrm{V}$
I	Nominal current	$3\,\mathrm{A}$
R	Terminal resistance	$1.01\,\Omega$
L	Terminal inductance	$0.955\,\mathrm{mH}$
p_p	Number of pole-pairs	7
k_v	Back-EMF constant	$105\,\mathrm{rpm V^{-1}}$
J	Rotor and load combined inertia	$0.05\,\mathrm{kg\,m^2}$
T_load	Load torque	$0.15\,\mathrm{N\,m}$

4.4 Simulation profile

The simulation profile was designed to show the motor start and dynamic behavior when the external load is suddenly applied. The profile can be divided into two phases:

First phase

In the first phase, at simulation time $t = 0.1\,\mathrm{s}$, the motor starts from a standstill with no load. The motor accelerates until the reference speed of $100\,\mathrm{rpm}$ is reached. Due to the high inertia configured in simulation parameters, the acceleration is relatively slow. The motor reaches the reference speed after approximately 1 second.

Second phase

The second phase executes at simulation time $t = 1.2\,\mathrm{s}$. Execution time is set to occur after the motor reaches a steady speed. At this point, an external load of $0.15\,\mathrm{N\,m}$ is applied to the rotor.

4.5 Simulation results

Figure 11 shows the simulation results of speed tracking, Figure 12 shows current outputs from the reference model (*'i_ref'*) and adaptive model (*'i_ada'*).

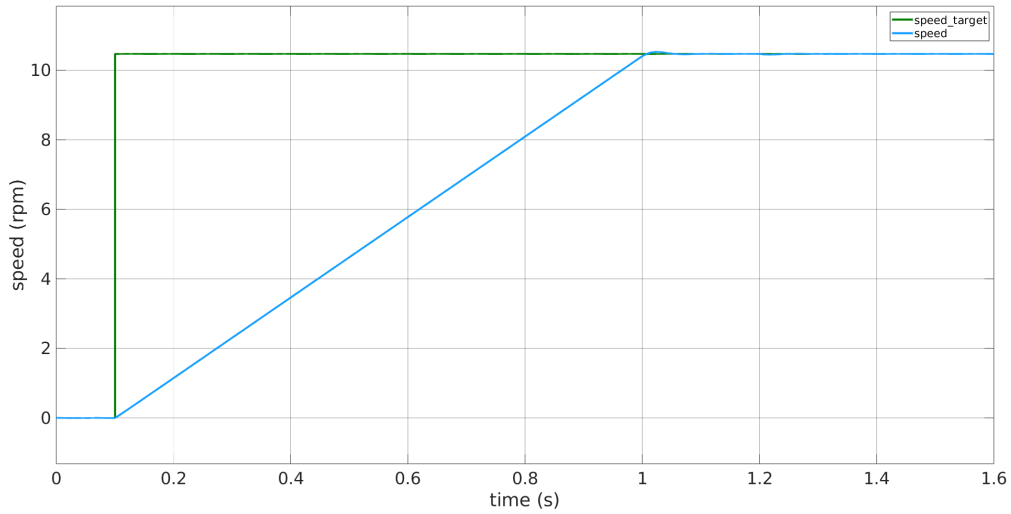

Figure 11: Speed tracking

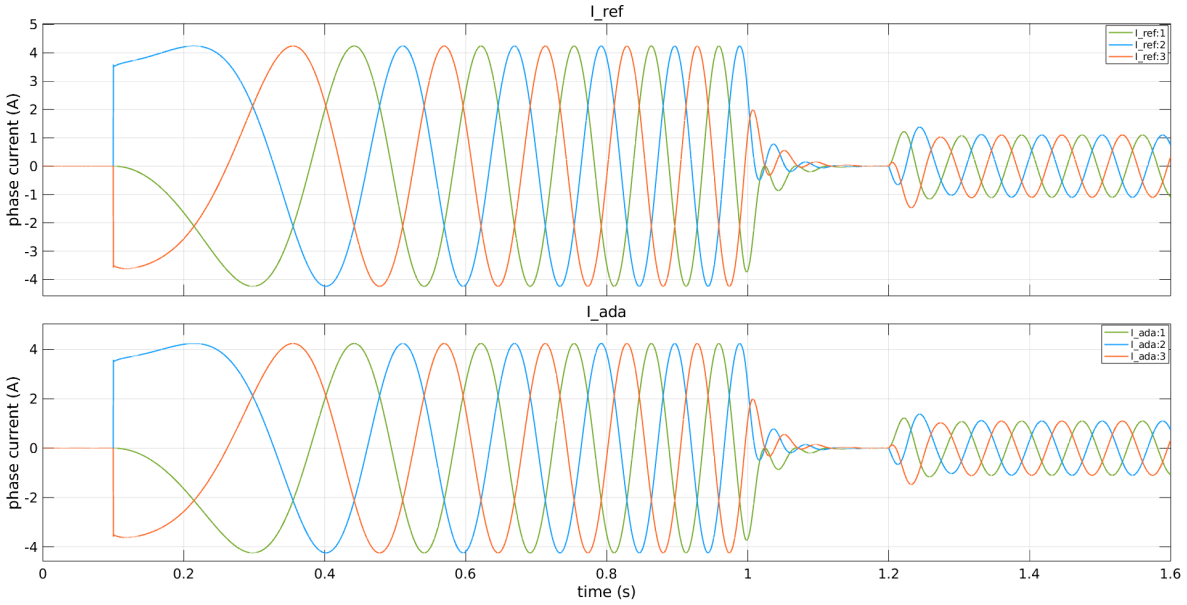

Figure 12: Comparison of output currents

4.5 Simulation results

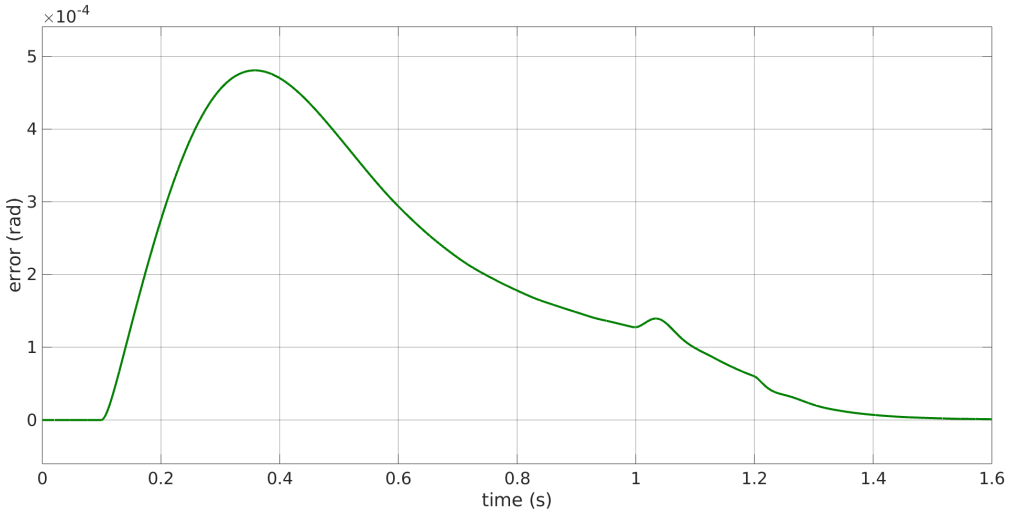

Figure 13: Angle error

The results confirmed the applicability and robustness of the method. Current outputs are almost the same; hence, the error is near zero. Figure 13 presents the angle difference, which confirms the near-zero error.

Chapter 5
Implementation

5.1 Hardware

The sensorless control algorithm is implemented on the evaluation board *EVSPIN32G4* by STMicroelectronics [31]. Experimental testing was performed with the Maxon *EC-i 40* motor equipped with a 10-bit *ENX 16 EASY* encoder [30][32].

Parameters of the motor can be found in Table 2. Resistance and inductance values were divided by two because they are listed as ”phase to phase” values in the datasheet and the model expects phase values.

Table 2: Maxon *EC-i 40* motor parameters

Symbol	Parameter	Value
U	Nominal voltage	$48\,\mathrm{V}$
	No load speed	$5000\,\mathrm{rpm}$
	No load current	$150\,\mathrm{mA}$
	Nominal speed	$4390\,\mathrm{rpm}$
I	Nominal current	$2.39\,\mathrm{A}$
R	Terminal resistance	$0.505\,\Omega$
L	Terminal inductance	$0.4775\,\mathrm{mH}$
p_p	Number of pole-pairs	7
k_v	Back-EMF constant	$105\,\mathrm{rpmV}^{-1}$

The *EVSPIN32G4* is a fully assembled and easy-to-deploy tool for assessing motor control algorithms. This board is built around the *STSPIN32G4*, a high-performance 3-phase motor controller with an integrated STM32G4 MCU. The *STSPIN32G4* is a fully self-supplied device with high degree of integration. It embeds a triple half-bridge gate driver with integrated bootstrap diodes, power management structures, a complete set of protections (thermal shutdown, short-circuit, undervoltage lockout), and an extended temperature range [33]. Figure 14 shows a block diagram of the gate driver.

Power management consists of a high-precision, low-dropout (LDO) linear regulator and a programmable buck regulator. The linear regulator generates a $3.3\,\mathrm{V}$ supply for both the gate driver logic and the microcontroller. An embedded programmable buck regulator powers the gate drivers. It can generate four different voltages: $8\,\mathrm{V}$, $10\,\mathrm{V}$, $12\,\mathrm{V}$, and $15\,\mathrm{V}$. Additionally, there is a standby regulator with a very low quiescent current for power saving. Power supplies are visualized at the top of the diagram.

The gate driver is internally connected to the MCU using the signals on the left side of the diagram. Inputs INH1-3 are intended for control signals of high-side switches, inputs INL1-3 are intended for

control signals of low-side switches, SCL and SDA are clock and data signals of the communication I^2C bus, and finally, WAKE, READY, and nFAULT are control and status signals, respectively. The right side of the diagram displays outputs for the power stage.

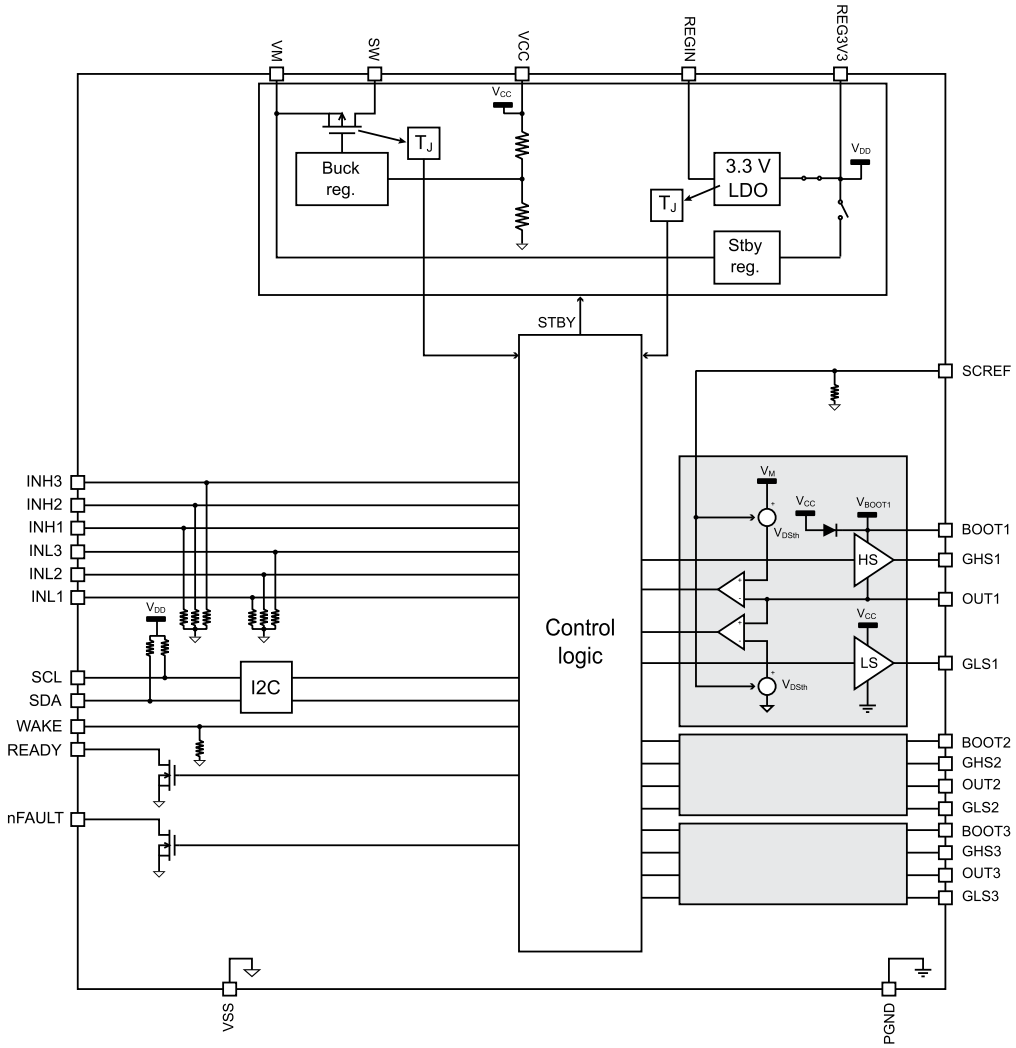

Figure 14: STSPIN32G4 gate driver IC block diagram [33]

The power stage on the board consists of a triple half-bridge inverter with *STL110N10F7* N-channel power MOSFETs. The board provides three-shunt topology low-side current sensing using the integrated operational amplifiers of the *STSPIN32G4* in differential amplifier configuration for better rejection of common mode signal.

Figure 15 shows the half-bridge for the first phase along with the amplifier network for current sensing. Resistors R41 and R47 realize voltage bias. If conditions

$$\begin{aligned} \mathrm{R44} &= \mathrm{R50} \\ \mathrm{R41}||\mathrm{R47} &= \mathrm{R53} \end{aligned}$$

hold, where R41||R47 is a parallel combination of aforementioned resistors, then the amplifier gain is given by resistors R50 and R53:

$$G = \frac{\mathrm{R53}}{\mathrm{R50}} \doteq 7.33\,. \tag{62}$$

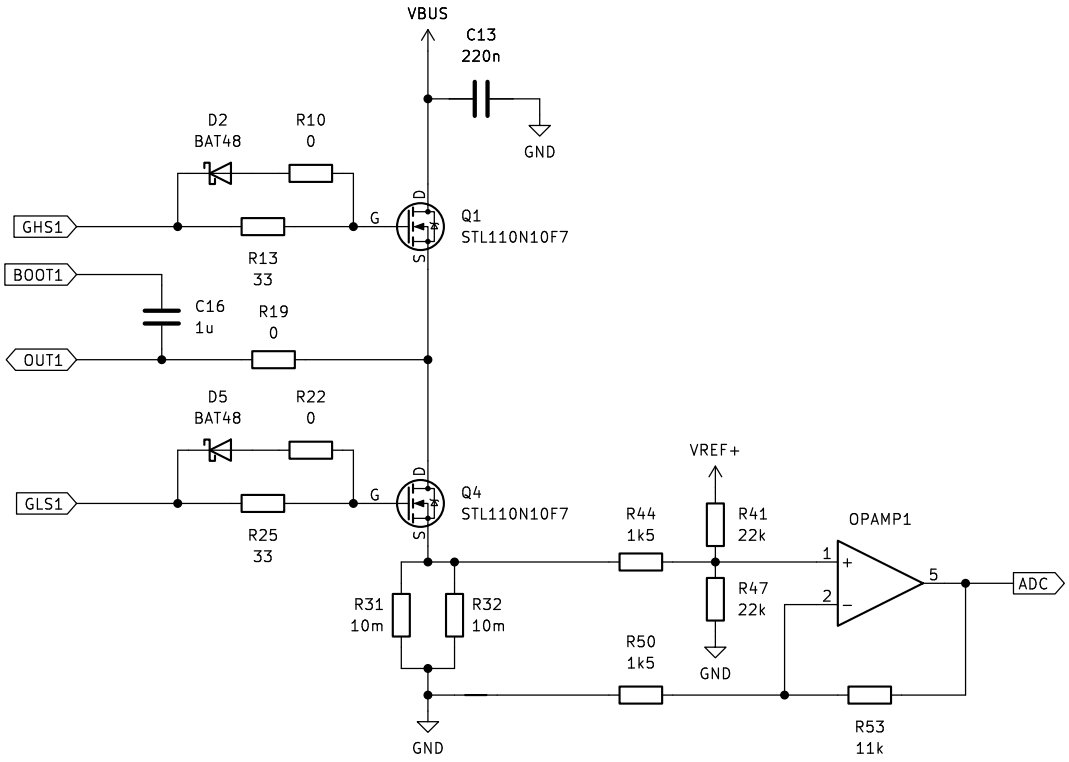

Figure 15: One of the three half-bridges

5.1.1 Board modifications

The board was initially equipped with two $10\,\mathrm{m\Omega}$ shunt resistors in parallel, allowing for a continuous current rating of $20\,\mathrm{A}$. Due to the 12-bit ADC resolution and current sensing network with a gain of 7.33, this configuration resulted in the smallest measurable current value of $22\,\mathrm{mA}$.

$$\text{resolution} = \frac{V_{\text{REF+}}}{2^{12}} \frac{1}{R_{\text{shunt}} \cdot G} \doteq 21.98\,\mathrm{mA}\,, \tag{63}$$

where R_{shunt} is the total shunt resistance, G is amplifier gain and $V_{\text{REF+}}$ is the ADC reference voltage.

While sufficient for high-power applications, this resolution is inappropriate for applications with low phase currents, as is the case with the Maxon motor used for testing, which is rated for maximal continuous current of $2.39\,\mathrm{A}$. Phase currents were, accordingly, expected to be low when operating under no or low load, and imprecise current measurements would degrade the algorithm's performance. Consequently, parallel shunt resistors were replaced with a single $20\,\mathrm{m\Omega}$ shunt resistor, thereby improving the resolution by a factor of 4.

5.2 Software and libraries

The control software was developed in C language with a mixture of STM32 HAL (Hardware Abstraction Layer) and LL (Low Layer) libraries. The HAL driver layer offers high-level and feature-oriented APIs (application programming interfaces) to interact with the application and hides the peripheral complexity from the end-user. On the other hand, the LL driver layer offers low-level APIs at the

register level, with better optimization but less portability. It offers hardware services based on the available features of the STM32 peripherals, which exactly reflect the hardware capabilities. These require deep knowledge of the MCU and peripheral specifications [34].

The application leverages the best of both: HAL for simple configuration and most of the peripheral servicing and LL primarily for time-critical parts and interrupt handling. The application also uses the ARM CMSIS-DSP library with implementations of transformations used in motor control and PID controllers. CMSIS-DSP is a standalone signal processing library for embedded systems, which provides optimized compute kernels for Cortex-M targets [35].

STM32CubeMX was used to set up the project files and generate primary code for peripheral and clock configuration. *STM32CubeMX* is a graphical tool that provides a convenient user interface to configure the microcontroller. It allows a simple configuration of pin assignments, peripherals, or the clock tree, and the corresponding C initialization code generation [36]. Generated configurations were customized afterward and implemented into the control library to ensure the correct settings based on the user parameters in `settings.h` file.

To develop, build, and mainly debug the software, *STM32CubeIDE* was employed. *STM32CubeIDE* is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors. *STM32CubeIDE* also includes standard and advanced debugging features including views of CPU core registers, memories, and peripheral registers, as well as live variable watch, Serial Wire Viewer interface, or fault analyzer [37].

A *git* version control system was used to track the software development. The *git* made it easier to navigate changes and test new features without the risk of breaking the code altogether and being unable to revert. *Git* is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency [38].

5.3 Peripheral configuration

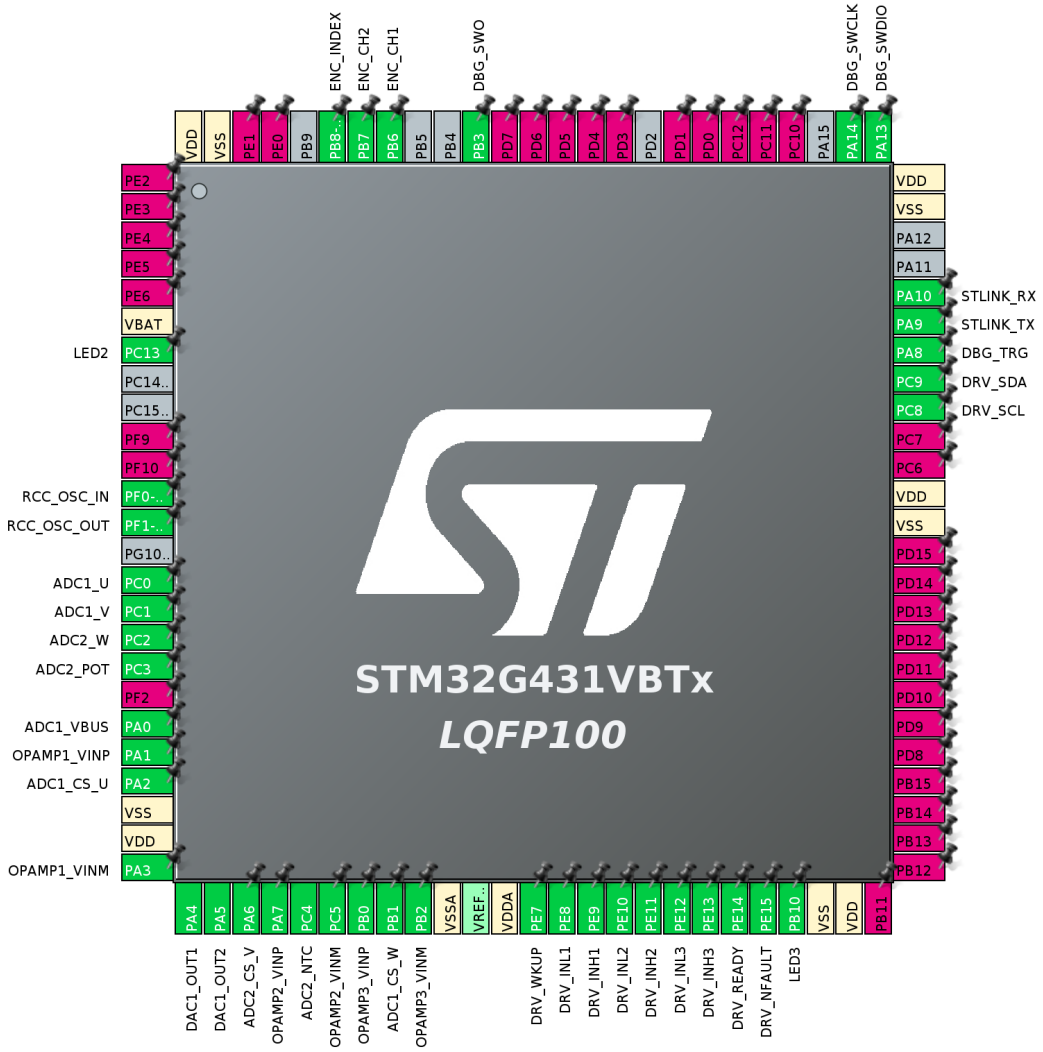

Figure 16: Pin assignments

The microcontroller is clocked at the highest frequency of 170 MHz with a clock source set to a 24 MHz external crystal oscillator mounted on the *EVSPIN32G4* board.

Figure 16 shows the microcontroller pin assignments. The red-pink-colored pins are not routed outside of the package. Pins prepended with "DRV_" are internally connected to the gate driver IC; their names principally correspond with datasheet names in Figure 14.

An I^2C bus is used to communicate with the gate driver. The bus is configured for the highest possible communication speed that the gate driver can handle and is accordingly set to the "Fast Mode Plus" with a bus frequency of 1000 kHz.

A timer *TIM1* responsible for the PWM generation is configured in center-aligned mode in "PWM mode 1", generating three complementary outputs with hardware dead-time insertion to prevent "shoot-through" (effectively short-circuiting the power supply). In center-aligned mode, the counter counts

from 0 to the auto-reload value (content of the TIMx_ARR register) – 1, generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event. Then it restarts counting from 0 [39]. The timer frequency is twice the motor switching frequency, preload registers are activated, and the repetition counter is set to 1.

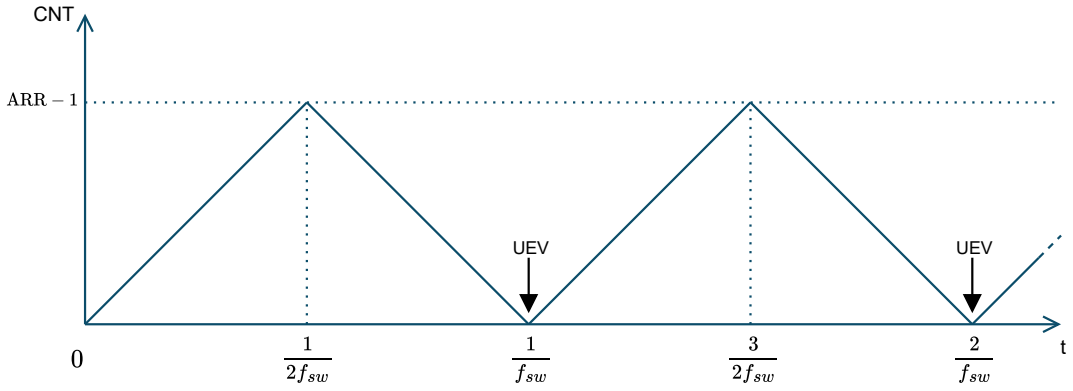

Figure 17: Illustration of the timer behavior

This configuration ensures that new duty-cycle values are transferred from the preload registers on the update event just before the counter starts counting up again. Without the repetition counter, the update event would occur even in the middle of the PWM period, which is unnecessary. ”PWM mode 1” guarantees that all low-side switches are turned ON when the update event arrives.

Figure 17 shows the timer’s behavior and the update event’s occurrence (UEV). The axis ”CNT” indicates the counter value.

A fourth capture-compare channel without the output is configured to internally trigger the ADC measurements at exact moments synchronized with the PWM. Synchronized triggering is illustrated in Figure 18 in Section 5.5. The gate-driver nFAULT signal is internally connected to the timer break input to shut down the PWM outputs immediately in case of gate-driver-related problems.

The system window watchdog feature is activated to ensure fault-free control software operation, and it resets the whole board if the control software unexpectedly hangs.

An ADC utilizes both regular and injected channels. Regular channels are configured in 12-bit resolution and continuous conversion mode, and they monitor board supply voltage, potentiometer output voltage, and MCU internal voltage reference. The channel measuring board supply voltage has the analog window watchdog feature activated to implement voltage protection. Section 5.4 provides more information about protections. Injected channels measure the outputs of operational amplifiers connected to shunt resistors and are triggered by the PWM timer.

The Direct Memory Access (DMA) manages data transfers from regular channels, and the data are later processed in the low-frequency task. The MCU processes data from injected channels in the interrupt handler. Each injected channel has its specialized data register to hold the converted value; therefore, DMA usage is not beneficial.

All three built-in operational amplifiers are activated in the default ”Standalone” mode with no additional special settings.

An incremental encoder with quadrature output provides a reference angle and velocity for com-

parison with estimated values. The output processing is simplified by exploiting the encoder interface mode of the timer peripheral. In this mode, the counter register value corresponds to the encoder position, and the rotation direction can be directly obtained by reading a counting direction bit in a register. To virtually double the encoder's resolution, the timer peripheral is configured in "x2 counting mode", where the counter is updated on both rising and falling edges of the signal. The incremental encoder is connected to the timer *TIM4*.

The software uses Serial Wire Output (SWO) mainly to output debugging strings via printing. It is also possible to output internal variables using SWO, but care must be taken not to overwhelm the internal trace buffers of *STM32CubeIDE* Serial Wire Viewer. High bandwidth data overloads them very quickly, causing SWO packet loss and making the IDE hardly controllable.

To overcome the bandwidth bottleneck, the control software utilizes a Digital-to-Analog Converter (DAC) as a debug feature to output fast changing internal variables and inspect them on an oscilloscope. Both channels of *DAC1* are enabled. To make the debug feature as non-intrusive as possible, values to be outputted by the DAC are transferred by two separate DMA channels that are triggered independently by timers *TIM6* and *TIM7* configured for 1 µs period. However, using a DAC to output internal variables introduces a small error in comparison results because the output buffer is not perfectly accurate.

5.4 Protections

Control software realizes several protections of the involved hardware. The motor is immediately turned off if the protection triggers.

Overvoltage and undervoltage protections are implemented using the analog window watchdog feature of the ADC. The board supply voltage is continuously measured and automatically compared against settable thresholds. An interrupt is triggered, and the motor stops if the supply voltage is not in the configured voltage range.

The software overcurrent protection is evaluated after every current measurement. Raw ADC values are first converted to current in the respective interrupt handler and straightaway compared against the settable threshold in both polarities. If out of range, the motor is stopped.

Gate driver offers two sources of thermal protection. Both the internal switching regulator supplying the voltage to the gate driver and the linear low-dropout (LDO) regulator have their own thermal protection. If the temperature exceeds the threshold, the corresponding regulator is switched off until the temperature returns to the normal operating range.

Gate driver also implements "VDS monitoring" protection. Drain-source voltages of power switches are monitored and compared against a reference threshold. This mechanism detects anomalous conditions, such as power switch failure resulting in short to either ground or supply voltage, and in that case, it forces all the gate driver outputs low. The protection is latched and has to be cleared by the software.

In the case of an unexpected program failure, the system window watchdog provides another way of protecting the involved hardware. Suppose the software does not reload the watchdog counter periodically before the set time threshold. In that case, the watchdog generates a reset of the microcontroller, resulting in a reset of the whole board. The counter is reloaded in the low-frequency task,

and the threshold is set to approximately 1.5 ms. This setup assures that the hardware is reset to the safe state after one missed low-frequency task iteration at maximum.

5.5 Current sensing

To correctly perform the field-oriented control, it is mandatory to measure the motor current every switching period. For the sensorless operation, this is the only feedback between the motor and the control software, and thus, it is mandatory to measure precisely and accurately.

As the board implements three-shunt topology low-side current sensing, the only time window when the motor phase current flows through the shunt resistor is if the low-side transistor of the same leg is turned on. Therefore, measuring the current flowing through the motor phases is possible only if all the low-side switches are turned on. Due to this, care must be taken to synchronize the ADC trigger with the PWM precisely.

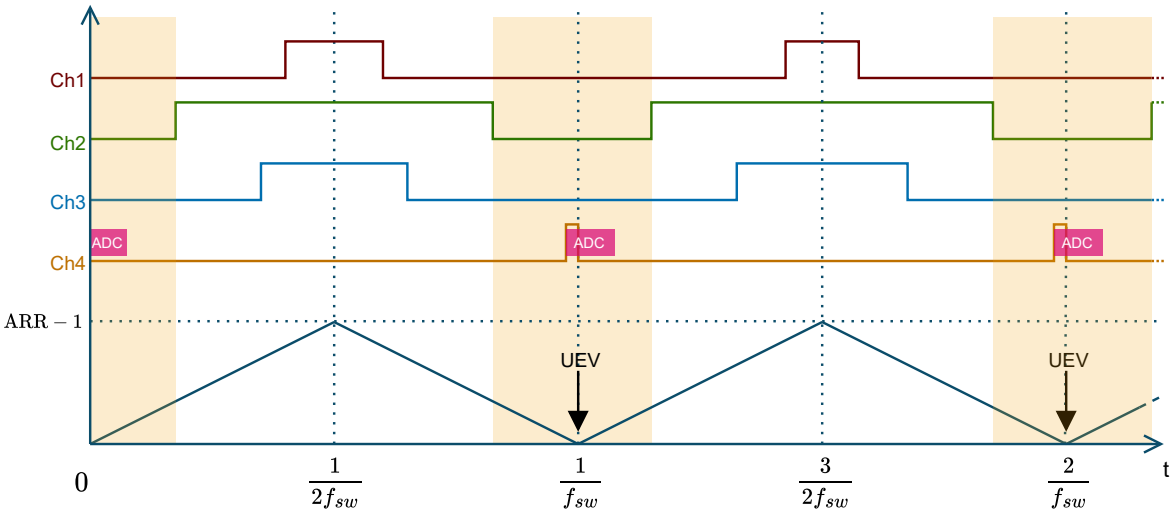

Figure 18: PWM timing diagram and ADC trigger

The internal trigger output of the fourth channel of the PWM timer ensures exact ADC triggering. By the nature of center-aligned PWM, all low-side switches are turned on when the update event arrives. Therefore, the trigger is set to be right before the update event.

Figure 18 displays an example of two PWM periods. Ch1 - Ch3 are timer output signals for individual high-side switches, and Ch4 is the synchronized internal ADC trigger. Inverted signals for low-side switches are not included in the figure. The measurement is possible in transparent orange sections, where the condition about low-side switches is fulfilled. Pink rectangles illustrate the ADC current measurement.

The ADC can only convert a value between 0 V and its reference voltage 3.3 V. To enable the measurement of current in both directions, the output of the shunt resistor is biased to half of ADC reference voltage by two equal-size resistors, and after that, it is amplified. Schematic showing the biasing can be found in Figure 15 in Section 5.1.

Even though equal-size resistors are used, the real resistance always differs and creates an offset.

Thus, assuming the bias voltage is strictly half of the reference voltage and, therefore, half of the maximum ADC value is erroneous. Instead, the magnitude of the offset is measured and calibrated before every start of the motor prior to the charging phase of the bootstrap capacitor. With all outputs disabled, the ADC performs several measurements of the offset, and an average value is stored for further current conversions.

With the increasing duty cycle of the PWM, the ON period of the low-side switch decreases, but the ADC conversion time is fixed. If it decreases to the point where the ADC conversion takes longer, then the ADC measures incorrectly, and current readings are unreliable. The control software leverages the current identity

$$i_a + i_b + i_c = 0 \tag{64}$$

to overcome this issue. If the highest phase duty cycle is higher than 75%, then the current for that phase is computed using the identity.

5.6 Software architecture

The control software is interrupt-driven and does not depend on any real-time operating system (RTOS). Instead, the context execution relies on hardware interrupts with assigned priorities and internal software state machine. A diagram of the state machine is displayed in Figure 19.

The program starts with the basic initialization of the hardware and advances to the idle state. If the program receives a start command, it leaves the idle state, configures the gate driver and concerned peripherals, and advances to successive execution states. After that, the program flow can be divided into two ”tasks” based on the execution frequency. Both tasks execute relevant procedures according to the state of the program.

High-frequency task

The high-frequency task is a set of vital procedures executed synchronously every switching period during the highest priority ADC interrupt routine after the conversion of injected channels is finished. It collects necessary information about the angle and speed and performs the main control loop.

Low-frequency task

The low-frequency task is based on the *SysTick* timer and executes a speed regulation loop and auxiliary functions with lower priority or lower execution frequency requirements.

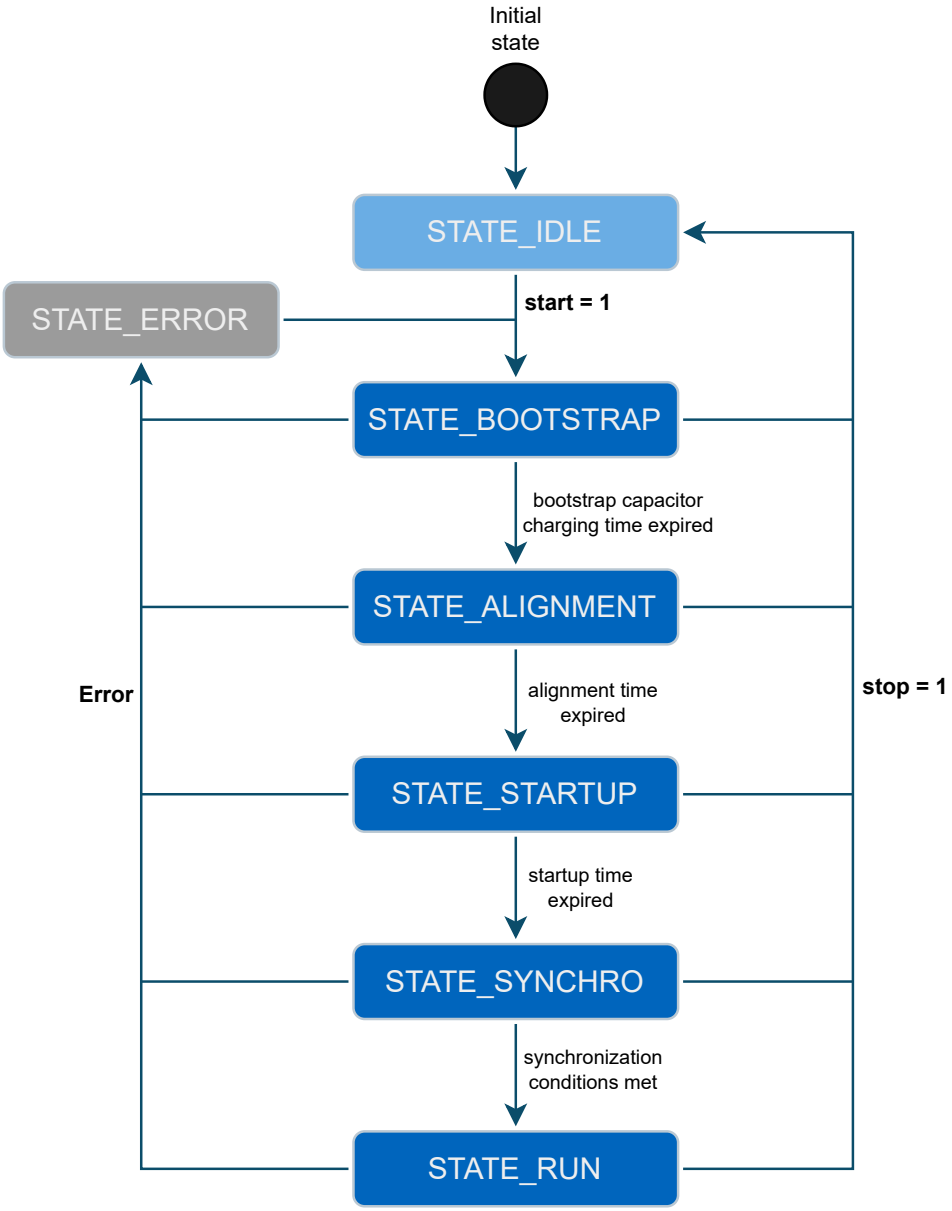

Figure 19: Program state machine

5.7 Main control loop

The main control loop follows a standard field-oriented control scheme, the diagram of which is shown in Figure 20. The i_d current component is controlled to zero, and a speed PI controller controls the i_q component. The implementation relies on optimized CMSIS DSP functions for computing the sine and cosine of an angle, Clarke and Park transformations, and PID controller.

First, the `arm_sin_cos_f32()` function is utilized to calculate the sine and cosine of the current electric angle of the motor, which is used for the Park and inverse Park transformations. Then, the three-phase current is first transformed to the $\alpha\beta$ system by the Clarke transform and then to the *dq* system by the Park transform (`arm_clarke_f32()`, `arm_park_f32()`).

Acquired direct and quadrature-axis currents are fed into two PI controllers to obtain the control voltages. PI controllers are based on the CMSIS DSP PID implementation (`arm_pid_f32()`), which is very simple but, for the sake of simplicity, does not include any saturation of the output or integral anti-windup technique. These features were implemented to keep the regulator simple by inspecting the control output value and limiting it and the PID's internal sum variable.

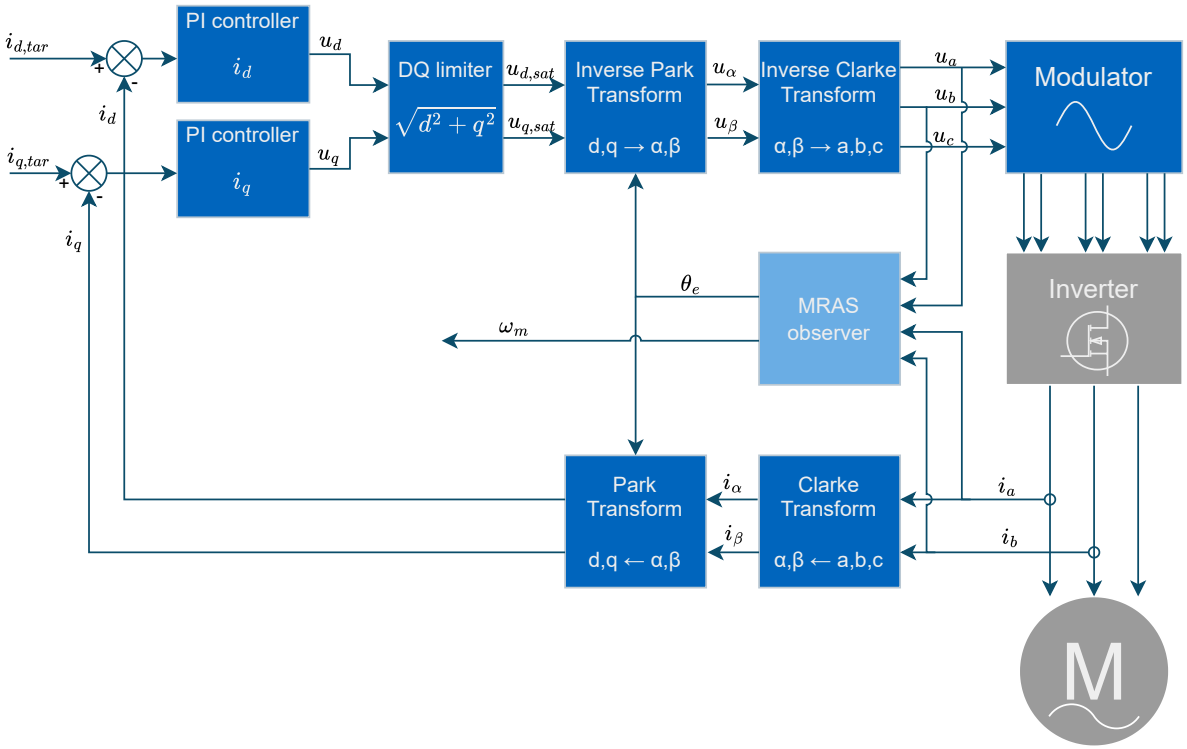

Figure 20: Block diagram of the field-oriented control

To avoid the distortion of current waveforms when using duty cycles close to zero or to the maximum counter value, where the inverter cannot produce the desired voltage due to the dead-time or a switching element unable to switch with sufficient speed, a mechanism to limit control *dq* voltages is employed. The magnitude of the resulting voltage vector is compared against a configurable limit value derived from the supply voltage. If it exceeds the maximum, *dq* voltages are saturated with prioritization on the d-axis voltage. In equations:

$$mag = \sqrt{u_d^2 + u_q^2} \tag{65}$$

When $mag > u_{\text{max}}$:

$$u_d^{\text{sat}} = \begin{cases} u_{\text{max}}, & \text{if } u_d \geq u_{\text{max}} \\ u_d\,, & \text{if } -u_{\text{max}} < u_d < u_{\text{max}} \\ -u_{\text{max}}, & \text{if } u_d \leq -u_{\text{max}} \end{cases} \tag{66}$$

$$u_q^{\text{sat}} = \begin{cases} \sqrt{u_{\text{max}}^2 - (u_d^{\text{sat}})^2} \cdot \text{sign}(u_q), & \text{if } |u_q| \geq \sqrt{u_{\text{max}}^2 - (u_d^{\text{sat}})^2} \\ u_q, & \text{if } |u_q| < \sqrt{u_{\text{max}}^2 - (u_d^{\text{sat}})^2} \end{cases} \tag{67}$$

When $mag \leq u_{\text{max}}$:

$$u_d^{\text{sat}} = u_d \tag{68}$$
$$u_q^{\text{sat}} = u_q \tag{69}$$

where u_{max} is the saturation voltage.

Afterward, the saturated control voltages are transformed back to the $\alpha\beta$ system by the inverse Park transform (`arm_inv_park_f32()`) and eventually to the *abc* coordinate system by the inverse Clarke transform (`arm_inv_clarke_f32()`). Finally, with respect to the maximum counter value, the control voltages are converted to values that are set in the timer capture/compare register.

5.8 MRAS observer

The observer implementation follows the block diagram of the implementation in the simulation environment in Figure 10. That means the adjustable model is implemented in the $\alpha\beta$ reference frame using equations 52 and 53, and the current outputs are eventually transformed into the *dq* reference frame. The only difference from schematics in Figures 10 and 20 is the reference frame of accepted input voltages. The voltage inputs for the MRAS are the same as those for the modulator, which are initially from PI controllers of field-oriented control in the *dq* reference frame and are consecutively transformed back to the *abc* reference frame. The MRAS subsystem accepts input voltages in the $\alpha\beta$ reference frame instead of the *abc* reference frame to save one inverse and one forward Clark transform.

The angle and speed estimation based on the MRAS observer is implemented using the equations derived in section 3.2. Implementing differential equations directly in the program is impossible; it is mandatory to utilize numerical methods. The control software offers two methods to choose from depending on the desired accuracy or speed of calculation.

The faster available method is a simple Euler method, also called the forward Euler method. The Euler method is a first-order method and, therefore, the most straightforward technique to solve differential equations with initial conditions numerically. At each step, the function is approximated by the tangent, assuming that the error will be small if the step size is small. Due to this fact, this approach is the fastest, but the error increases with larger step sizes and more fluctuating waveforms and accumulates over time. Equations after application of the simple Euler method have form

$$\hat{i}_\alpha[n+1] = \hat{i}_\alpha[n] + h \cdot \left(-\frac{R}{L}\hat{i}_\alpha[n] + \frac{1}{L}k_{\text{e}}\hat{\omega}_{\text{m}}[n]\sin\left(\hat{\theta}_{\text{e}}[n]\right) + \frac{1}{L}u_\alpha[n] \right) \tag{70}$$
$$\hat{i}_\beta[n+1] = \hat{i}_\beta[n] + h \cdot \left(-\frac{R}{L}\hat{i}_\beta[n] + \frac{1}{L}k_{\text{e}}\hat{\omega}_{\text{m}}[n]\cos\left(\hat{\theta}_{\text{e}}[n]\right) + \frac{1}{L}u_\beta[n] \right) , \tag{71}$$

where $h = \frac{1}{f_{\text{sw}}}$ is a time step given by the switching frequency of the motor.

The more accurate method is a modified Euler method, also known as Heun's method or explicit trapezoidal rule. It is a two-stage, second-order procedure to solve differential equations. It uses a simple Euler method as a foundation. First, the intermediate approximation of the slope at t_i is computed, and then the slope at the next step $t_{i+1} = t_i + h$ with recently calculated intermediate. Finally, both slopes are averaged to find a better solution approximation at the end of the time step.

Equations after application of the modified Euler method have form

$$\hat{i}_{\alpha,1} = -\frac{R}{L}\hat{i}_\alpha[n] + \frac{1}{L}k_\mathrm{e}\hat{\omega}_\mathrm{m}[n]\sin\left(\hat{\theta}_\mathrm{e}[n]\right) + \frac{1}{L}u_\alpha[n] \tag{72}$$

$$\hat{i}_{\beta,1} = -\frac{R}{L}\hat{i}_\beta[n] + \frac{1}{L}k_\mathrm{e}\hat{\omega}_\mathrm{m}[n]\cos\left(\hat{\theta}_\mathrm{e}[n]\right) + \frac{1}{L}u_\beta[n] \tag{73}$$

$$\hat{i}_{\alpha,\mathrm{pred}} = \hat{i}_\alpha[n] + h \cdot \hat{i}_{\alpha,1} \tag{74}$$

$$\hat{i}_{\beta,\mathrm{pred}} = \hat{i}_\beta[n] + h \cdot \hat{i}_{\beta,1} \tag{75}$$

$$\hat{i}_{\alpha,2} = -\frac{R}{L}\hat{i}_{\alpha,\mathrm{pred}} + \frac{1}{L}k_\mathrm{e}\hat{\omega}_\mathrm{m}[n]\sin\left(\hat{\theta}_\mathrm{e}[n]\right) + \frac{1}{L}u_\alpha[n] \tag{76}$$

$$\hat{i}_{\beta,2} = -\frac{R}{L}\hat{i}_{\beta,\mathrm{pred}} + \frac{1}{L}k_\mathrm{e}\hat{\omega}_\mathrm{m}[n]\cos\left(\hat{\theta}_\mathrm{e}[n]\right) + \frac{1}{L}u_\beta[n] \tag{77}$$

$$\hat{i}_\alpha[n+1] = \hat{i}_\alpha[n] + \frac{h}{2} \cdot \left(\hat{i}_{\alpha,1} + \hat{i}_{\alpha,2}\right) \tag{78}$$

$$\hat{i}_\beta[n+1] = \hat{i}_\beta[n] + \frac{h}{2} \cdot \left(\hat{i}_{\beta,1} + \hat{i}_{\beta,2}\right) \ , \tag{79}$$

where $h = \frac{1}{f_\mathrm{sw}}$ is a time step given by the switching frequency of the motor, and $\hat{i}_{\alpha/\beta,1/2}, \hat{i}_{\alpha/\beta,\mathrm{pred}}$ are intermediate values.

After acquiring the estimated currents from the model, they are processed using the adaptation law equation 51, and the resulting error is fed to the PI controller to get the estimated speed. According to the equation 50, the angle is obtained by integration of the speed. The numerical integration is realized using the forward Euler method, which approximates the integral by

$$\hat{\theta}_\mathrm{e} = \int_t^{t+\tau} \hat{\omega}_\mathrm{e}\,\mathrm{d}\tau \qquad \rightarrow \qquad \hat{\theta}_\mathrm{e}[n+1] = \hat{\theta}_\mathrm{e}[n] + \tau \cdot \hat{\omega}_\mathrm{e}[n] \ , \tag{80}$$

where $\tau = \frac{1}{f_\mathrm{sw}}$ is a time step given by the switching frequency of the motor.

5.9 Startup procedure

Sensorless algorithms generally suffer when it comes to low-speed operation and motor startup. Depending on the particular algorithm, this may happen due to, e.g., the low back-EMF, low current, or inaccurate motor parameters. In order to smoothly bring the motor to operation from zero speed, the control software adapts an open-loop startup procedure with closed-loop current control based on the I-f startup method in [7]. The method utilizes current controllers from the field-oriented control and operates the motor with a predefined current, thus preventing the risk of an overcurrent condition. The procedure can be divided into several phases: alignment phase, open-loop startup, and synchronization phase. An illustration of the startup procedure is in Figure 21.

In the alignment phase, a current vector with a preconfigured angle and amplitude is applied to the motor, forcing the rotor flux to align with the generated stator flux, which is achieved by requesting a d-axis current that is sufficiently large to overcome the load and inertia. The alignment current is ramped up from zero by a linear ramp to prevent oscillations or unwanted twitches caused by sudden current application. After the end of the alignment phase duration, it is assumed that the rotor electrical angle corresponds to the demanded alignment angle, and the procedure advances to the next phase.

The open-loop startup further leverages the effect of stator and rotor flux alignment. Startup is accomplished by creating a virtual position sensor. The motor speed is linearly ramped up, and the angle

of the virtual position sensor is computed according to the actual speed until the speed reaches the configured startup speed. Acceleration rate is defined by startup time and desired startup speed. The angle of the virtual position sensor is the input to the field-oriented control, generating accelerating rotating stator flux with preconfigured amplitude. The current amplitude should be, once more, sufficiently large to overcome the load and inertia and ensure the startup is successful. After the open-loop ramp duration ends, the procedure advances to the final phase.

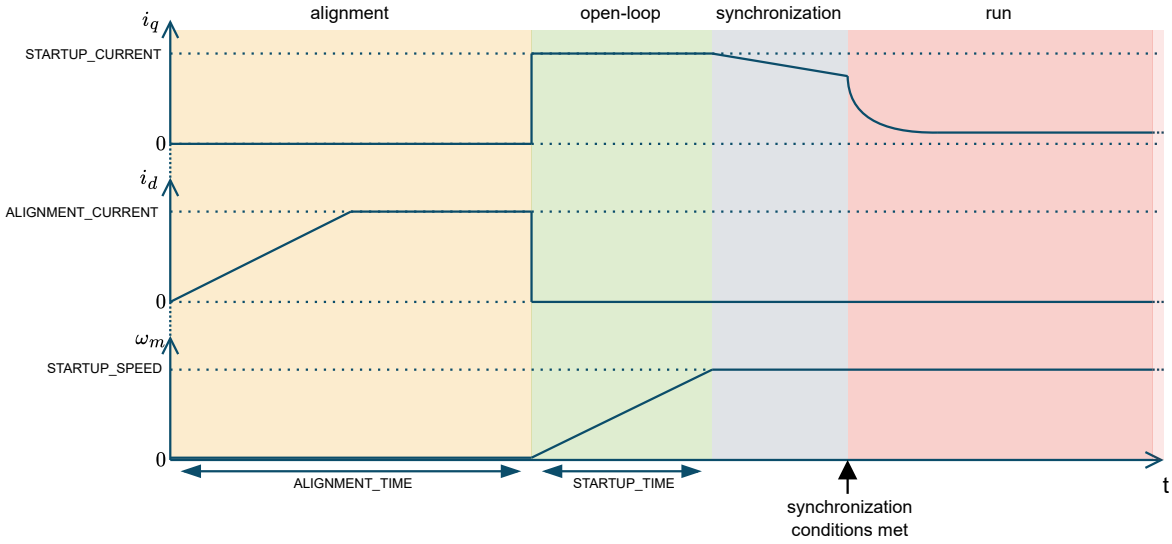

Figure 21: Startup procedure illustration

The synchronization phase ensures a smooth transition from the open-loop control to the closed-loop sensorless control. The virtual position sensor phase is still used in this phase to keep the motor in open-loop operation at a constant speed. Constant speed and decreasing of the q-axis current are to ensure better convergence of the observer. After the synchronization conditions are fulfilled (the difference between the startup angle and the estimated angle and the synchronization speed and the estimated speed), the control is entirely handed over to the sensorless algorithm. If the conditions are not met before the synchronization phase is over, the motor is stopped, and the control program enters the idle state.

5.10 Code structure

The control software was implemented as a library with a set of header and source files.

A library for interfacing with the internal gate driver is in folder *STSPIN32G4*. It contains all register definitions and necessary functions to configure the gate driver, implemented according to the datasheet [33].

All control software settings, e.g., motor parameters, duration of particular startup phases, or special library configurations, are in the header file `settings.h`.

5.11 Configuration and usage

From the user's perspective, the library has two important files: `settings.h` and `typedefs.h`.

All control software settings can be found in the `settings.h` file, as already mentioned in Section 5.6. The `settings.h` file contains settings as constants that cannot be changed during runtime and one helper macro to convert the back-EMF constant from $\mathrm{rpm\,V^{-1}}$ to SI units $\mathrm{V\,s\,rad^{-1}}$ fundamental for the PMSM model. There, the user configures parameters such as the switching frequency, dead-time length, motor inductance, resistance and back-EMF constant, duration and current amplitudes of particular startup phases, and settings of PI controllers. In addition, there are constants with resistance values of voltage dividers, resistors used in the current sensing network, and shunt resistors in the event of hardware customization.

Because it was necessary to implement the processing of the incremental encoder to allow comparison of the estimated values with reference values, the control software can be switched to sensor mode and run the motor with the position feedback from the encoder. Besides sensor mode, it is also possible to configure the library to use only the simple Euler method to solve PMSM equations in MRAS observer and disable the open-loop startup. Especially in sensor mode, the open-loop startup

with the associated synchronization phase may be omitted.

To operate the control software, the board needs to be connected to a computer, and the software launched in debug mode. Afterward, the software is controlled using two variables and a control structure. For that purpose, the *STM32CubeIDE* live variable watch feature can be used to inspect and set the variables during runtime. Figure 22 shows the live variable watch feature example. Setting variable `start` to `1` runs the motor, setting variable `stop` to `1` obviously stops the motor.

The variable `evspin` is an extensive control structure with variables used throughout the control software. The file `typedefs.h` defines all control structure members and types. From the user's perspective, inside the control structure are a couple of mentionable variables and substructures that are positioned at the top of the structure.

The first member variable `state` is an enum type used in the state machine holding the current program execution state.

The second member variable, `base`, is a structure that provides slightly more information about the execution state of the program than the state variable. It contains boolean flags representing each state, plus flags indicating whether the DQ limiter or the speed ramp is active.

The third member variable `run` is used to control the speed in the run phase after successful synchronization. It offers information about the actual speed and speed target and contains variables to set and execute a linear speed ramp. To set the new target speed, enter the value to variable `ramp_final`, desired duration of the speed change to `ramp_duration`, and execute the ramp by setting the variable `activate_ramp` to `true`.

In the fourth member variable `foc`, we can find variables essential for field-oriented control, especially currents in all three reference frames.

Substructure `adc` contains everything related to the ADC, most notably variables `vbus` and `vdda` hold information about the power supply.

The rest of the control structure consists of substructures with mostly uninteresting variables essential for various parts of the algorithm execution.

The thesis includes a complete project in *STM32CubeIDE* with all the necessary settings to build the library and launch files to run a debug session to control the software on the fly. The project is also available on GitHub[1].

[1] `https://github.com/lunakiller/EVSPIN32G4_FOC`

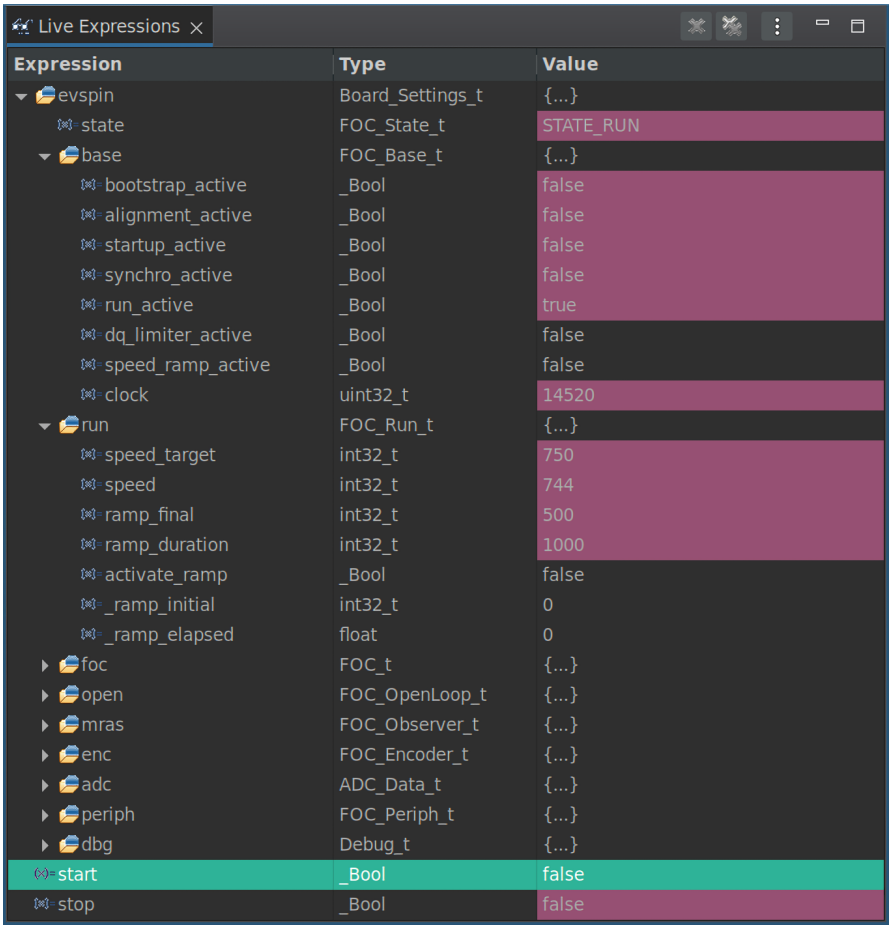

Figure 22: Example of the live variable watch feature

Chapter 6
Results

6.1 Experimental setup

Figure 23 shows the experimental setup. Except for the *EVSPIN32G4* board and Maxon *EC-i 40* motor, the setup consists of a Tektronix *MSO58B* oscilloscope, TTI *EX354T* triple power supply, and a laptop running *STM32CubeIDE* and the control software in debug mode. Phase current waveforms were measured using Tektronix *TCP0030A* current probes. Measurements of other quantities were realized using voltage probes connected to the DAC outputs and one GPIO pin.

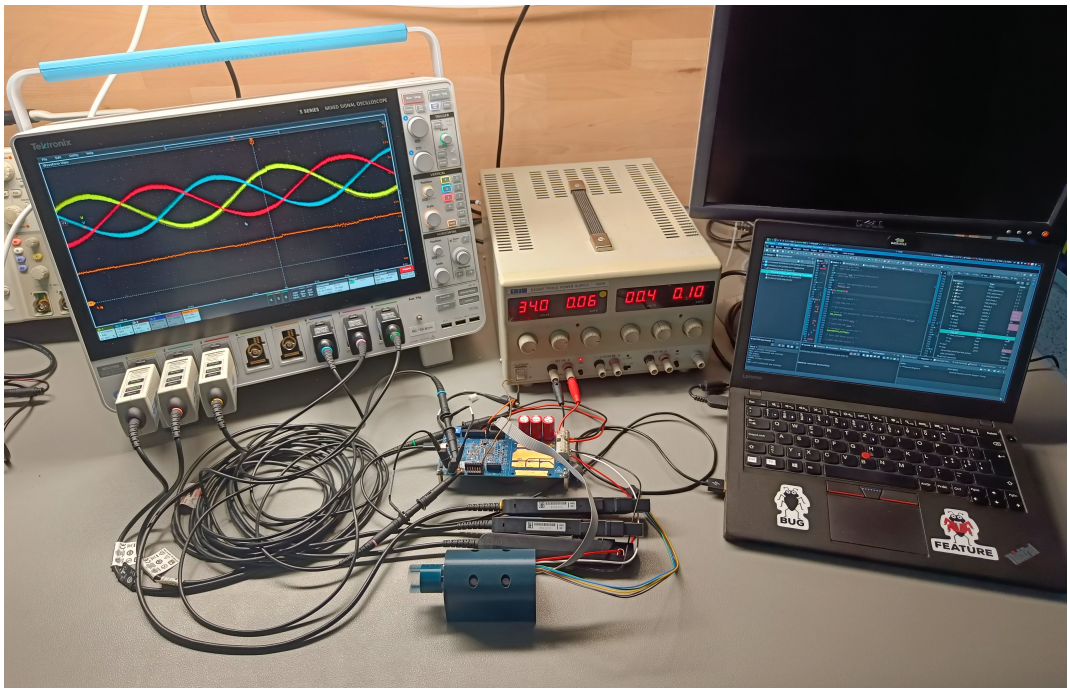

Figure 23: Experimental setup

The parameters of the motor used for experiments correspond to the datasheet parameters already presented in Table 2. Only the supply voltage was chosen to be 34 V, mainly due to the power supply limitation.

Control software settings are in Table 3.

As in the simulation, all PI constants were also obtained by the trial and error method. Although the controller implementation is a complete three-term PID controller, the derivative control is not used, and in all cases, the D constant is kept at zero. With the assistance of an oscilloscope, the constants were tuned until current waveforms were pure sinusoidal.

Table 3: Settings of the control software

Setting	Value
Switching frequency	30 kHz
Dead-time	300 ns
Bootstrap phase duration	250 ms
Alignment phase duration	1000 ms
Alignment phase i_d	1500 mA
Alignment angle	60 deg
Open-loop startup duration	300 ms
Open-loop startup i_q	1200 mA
Open-loop startup final speed	750 rpm
Max synchronization phase duration	2000 ms

6.2 *dq* current PI controller

Figure 24 shows the step response and behavior of d-axis current controller on significantly changing setpoints. Small current oscillations after application of 1 A i_d are caused by the rotor oscillations due to the sudden application of the current.

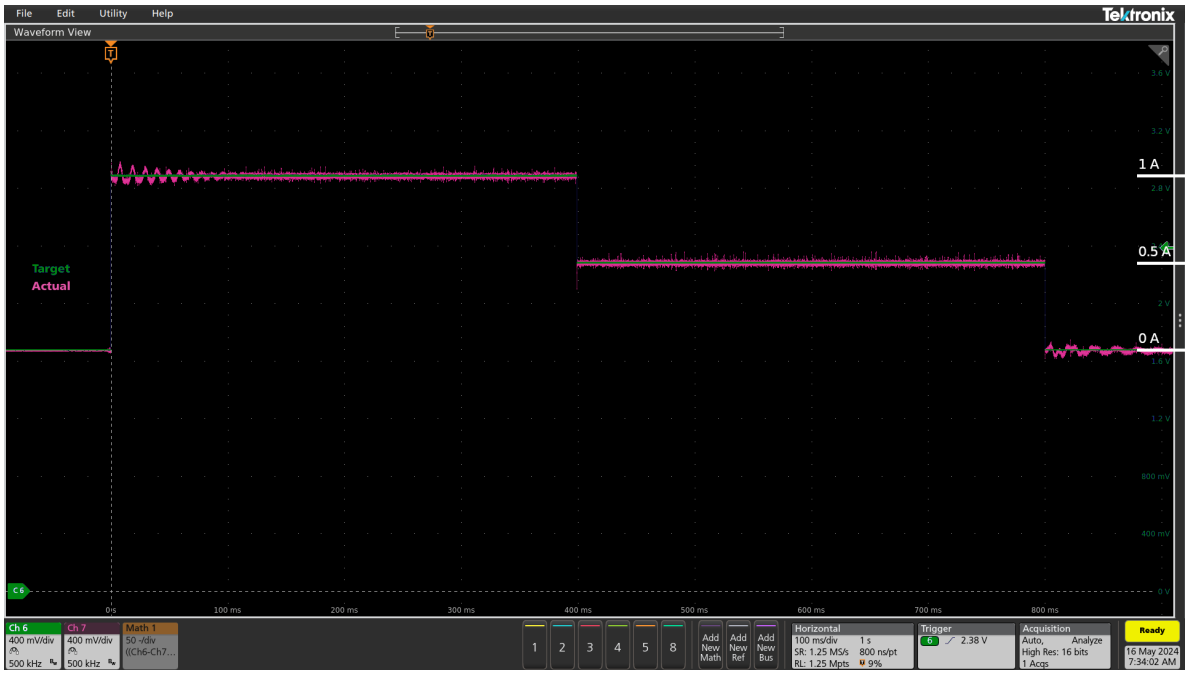

Figure 24: i_d current PI controller response

Current PI controller constants are the same for both d- and q-axis currents; therefore, the q-axis current controller shares the same dynamics.

6.3 No load start

The first test the proposed sensorless algorithm underwent was a simple start of the motor with no load. The motor accelerated from a standstill to $750\,\mathrm{rpm}$ using the I-f startup method explained in Section 5.9. During tests, the estimated angle and speed were compared with the reference angle and speed from the encoder.

Due to only two available DAC output channels, it was not possible to compare both estimated values with the reference values simultaneously. Instead, the tests had to be performed one at a time. This fact further confirms the functionality of the method and its repeatability.

Figure 25 shows a comparison of the estimated angle and angle from the encoder. Angles are electrical angles in range $[-p_\mathrm{p}180°; p_\mathrm{p}180°]$. Channels 6 and 7 of the oscilloscope are DAC outputs, channel 8 is used concurrently as a trigger and as an indicator of the synchronization phase, which begins with the rising edge and ends with the falling edge. The math channel shows the difference and converts it from the voltage back to the original values that were input to the DAC. Therefore, in this case, the math channel corresponds to the difference between the estimated value and the value from the encoder in degrees.

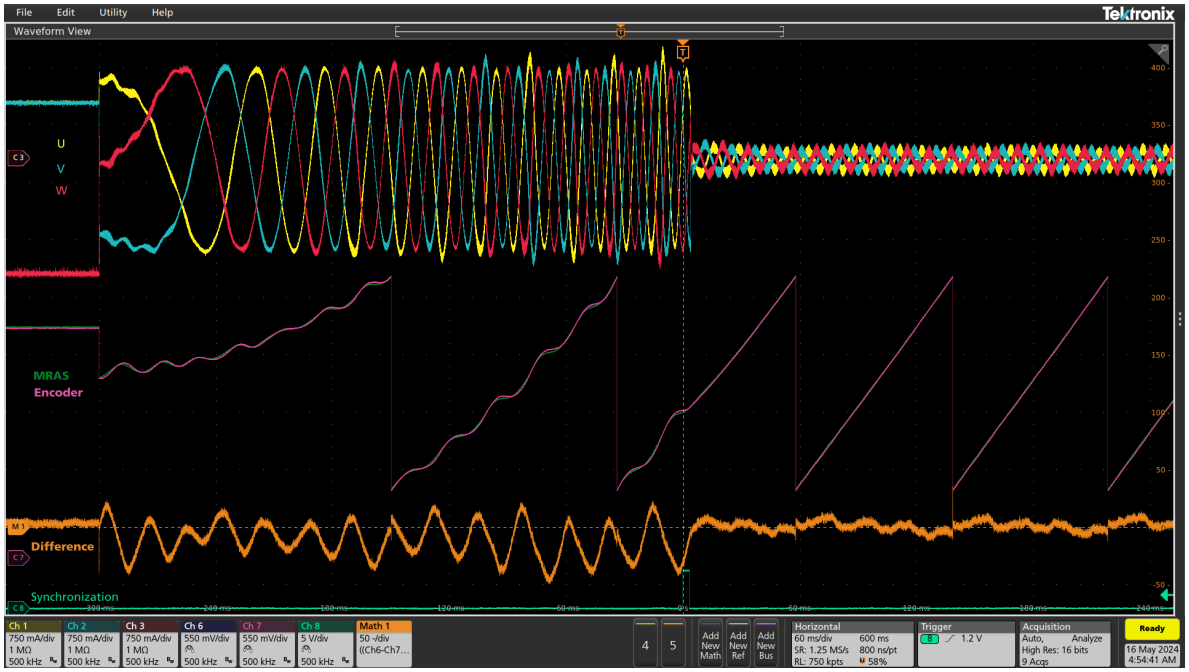

Figure 25: Angle comparison

The figure shows that the rotor position difference oscillates as a result of open-loop startup. However, after the synchronization phase, when the control is completely handed over to the sensorless algorithm, the angular difference remains close to zero. Peaks in the angle difference caused by the angle wrap-around at different moments were mostly filtered out.

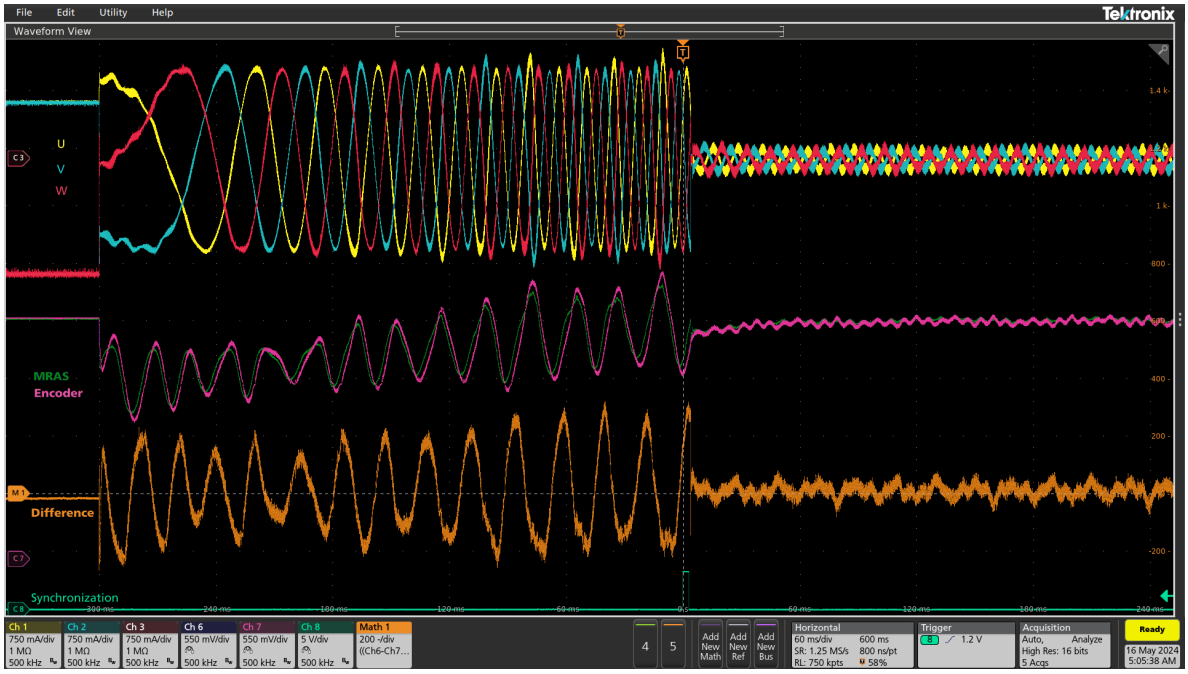

Figure 26: Speed comparison

Figure 26 displays the speed comparison. In this test, the math channel corresponds to the difference between the estimated value and the value from the encoder in rotations per minute. The speed also oscillates during the startup, and the error is small after synchronization. Also, fluctuations in the speed signal from the encoder after the synchronization can be noticed, and the speed estimate is much more stable.

An important aspect to consider when inspecting the results is the low resolution of the used encoder (only 10-bit), whose output has to be filtered to be usable. It is highly likely that the poor resolution of the encoder and filtering is the cause of the difference between the estimated and reference value and that the actual error is close to zero.

6.4 Run under load

Consequent figures demonstrate the motor operation under load. The motor was loaded with approximately $136\,\mathrm{mN\,m}$ load torque, resulting in q-axis current $i_q \doteq 1000\,\mathrm{mA}$. Figure 27 shows operation at $750\,\mathrm{rpm}$ and Figure 28 shows operation at $1500\,\mathrm{rpm}$.

6.4 Run under load

Figure 27: Run under load, $\omega_\mathrm{m} = 750\,\mathrm{rpm}$

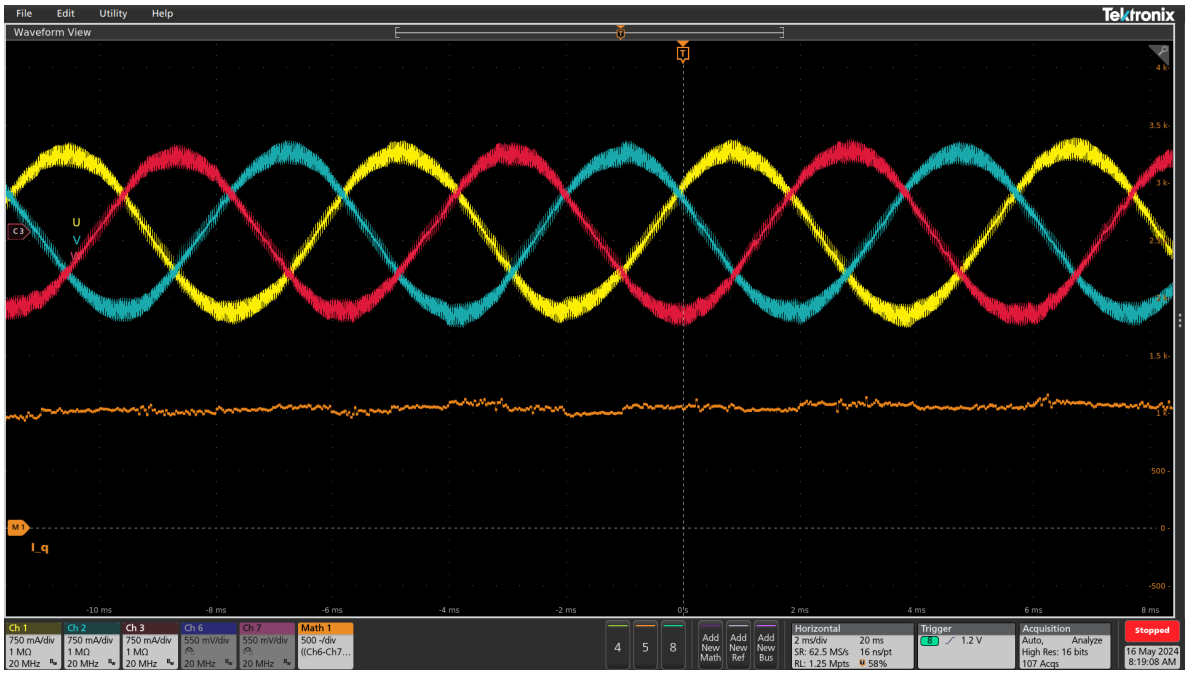

Figure 28: Run under load, $\omega_\mathrm{m} = 1500\,\mathrm{rpm}$

In both cases, we can see sinusoidal current waveforms that are undistorted by higher harmonics.

6.5 Startup with erroneous parameters

Experiments were also conducted to demonstrate the robustness of the control method to erroneous or changing motor parameters. For example, the parameters can change during motor runtime due to increasing temperature. Besides, the measurement of motor parameters tends to be a problematic task, and it is often the case that the parameters measured by the LCR meter do not fully correspond to datasheet parameters, regardless of the measurement frequency set on the LCR meter.

Because motor parameters are defined in the program as constants, it was not possible to change the inductance and resistance values on the fly without interfering with the control library. Instead, it was tested during the startup with wrongly configured motor parameters.

First, six start test cases were run: half the resistance value, half the inductance value, half the value of both, and then the same but with twice the resistance and inductance values. All other parameters were unchanged during the tests.

Table 4: Results of average angle differences

Configuration	**Average difference**
$0.5 \cdot R$	$0.5°$
$0.5 \cdot L$	$10.2°$
$0.5 \cdot R, 0.5 \cdot L$	$3.22°$
$2 \cdot R$	$-1.6°$
$2 \cdot L$	$-4.8°$
$2 \cdot R, 2 \cdot L$	$-3.7°$

Results proved that the method can start, synchronize, and run the motor without significant issues. It was possible to change speed and load the motor without the loss of control. The angle during open-loop startup oscillated more, and the synchronization phase took longer. Even though the algorithm was able to synchronize, there was a steady-state error between the estimated and reference angles, resulting in likely degraded performance and efficiency. The Table 4 contains the average angle differences in the run phase of the motor. The method seems more prone to error in the inductance parameter, which is apparent from the results.

After this observation, additional tests were conducted to verify the limits of the method's tolerance limit to the significantly erroneous resistance parameter. During tests with ten times the datasheet resistance value, the method was unable to synchronize. The synchronization failed either due to maximal synchronization phase duration expiration or overcurrent caused by the sudden divergence of the sensorless observer.

Subsequently, the resistance was decreased to five times the datasheet value. Even with this high error, the observer could converge and synchronize.

6.5 Startup with erroneous parameters

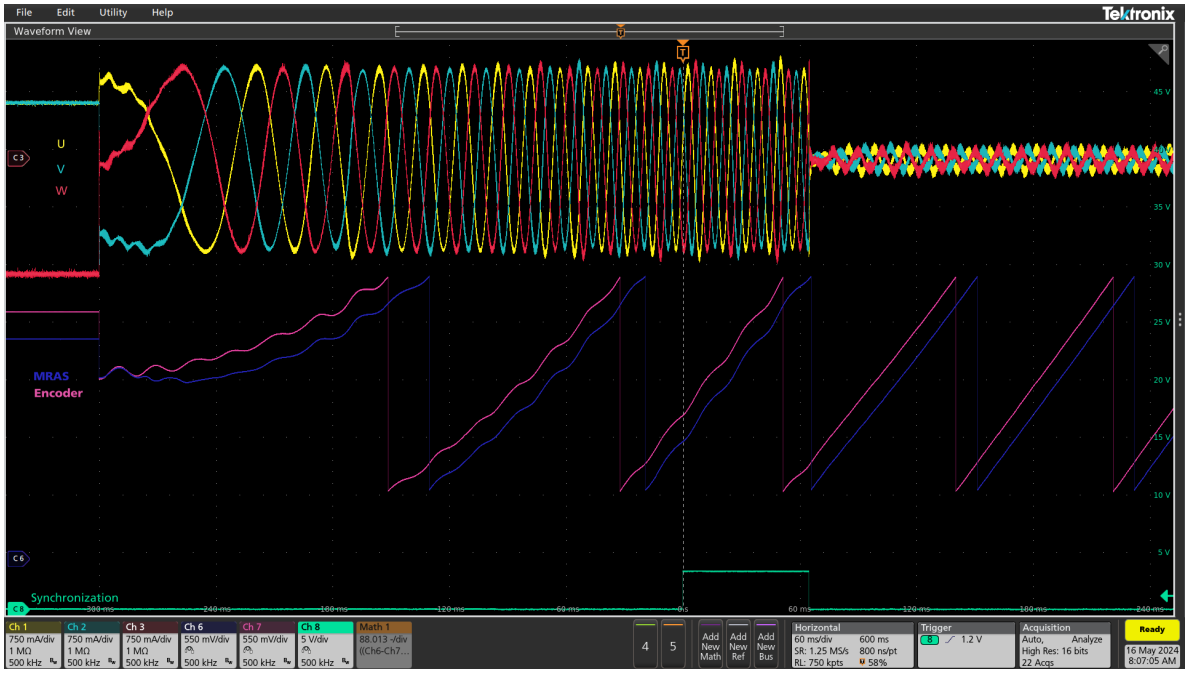

Figure 29: Worse startup with $5 \cdot R$, angle comparison

Figure 29 shows one of the worse starts with five times the resistance value. The difference channel is hidden due to the angle shift of one electrical rotation between the estimated and reference value. It is clear that the convergence performance has deteriorated, and the overall performance has visibly degraded.

Although the motor successfully reached the running phase, the algorithm diverged as the load was gradually applied, the estimate suddenly changed, and the control failed due to the overcurrent. Five times the resistance error is too high, causing the control to be unstable under dynamically changing loads. However, the change in resistance due to temperature does not reach such values.

Chapter 7
Conclusion

The thesis aimed at the problematics of field-oriented control of PMSMs without using a position sensor. Such sensors are unwanted due to increased production and maintenance costs and a greater risk of malfunction.

In order to simulate a three-phase PMSM and corresponding control algorithms, the mathematical model was derived, and commonly used Clarke and Park transformations were introduced to transform the model to a form more suitable for control. Typically, the PMSM model uses flux-linkage in the back-EMF term; however, the manufacturer often does not specify a flux-linkage parameter in its datasheet, especially for low-cost motors. Therefore, the model uses an alternative parametrization using the back-EMF constant and rotational speed.

Most often used sensorless algorithms were briefly introduced and compared. Based on referenced resources, the control method based on MRAS was selected, as it offered the most satisfactory performance among others, especially over a wide speed range and under varying load operation. The method was further analyzed, and all necessary parts were explained.

First, PMSM models were verified in a simulation in MATLAB Simulink environment against the model available in the Simscape add-on library to check the accuracy and correctness. Then, the field-oriented control was implemented along with the MRAS observer. The developed sensorless solution was verified in simulations, yielding very satisfactory results.

Afterward, the solution was implemented on the *EVSPIN32G4* board, resulting in complete sensorless software for controlling PMSM motors. The software is configurable to satisfy the usage of motors with different parameters. The control software is interrupt-based to guarantee the execution of individual time-critical phases at precise moments. It takes advantage of many of the hardware features available in the *STSPIN32G4*, such as DMA and internal peripheral triggering, to meet strict timing requirements and reduce the load on the processor.

Finally, the control software was tested with the Maxon *EC-i 40* motor. Test results demonstrated the capabilities and the accuracy of the method and functionality of the developed control software. The control is stable with and without the load, and the resulting current waveforms are purely sinusoidal. Additionally, the method proved to be reasonably tolerant to erroneous motor parameters. Even when the parameters were significantly wrong, the method managed to start and run the motor without any visible impact on the accuracy and quality of the control.

Future work could include extending the control software to operate over the entire speed range. As with other methods based on fundamental excitation, the proposed method has degraded performance at very low speeds, and thus, the open-loop startup method had to be employed. Although it was possible to start the motor from a standstill and skip the open-loop start, the control in very low-speed ranges was unreliable, especially at high loads. For this purpose, combining the proposed method with a method based on signal injection would be beneficial in allowing proper control during startup and low speeds. After the start-up is overcome, the control would switch to the current solution.

References

[1] Ramu Krishnan. *Permanent Magnet Synchronous and Brushless DC Motor Drives*. CRC Press, 1st edition, 2010. doi: 10.1201/9781420014235.

[2] Implement ab to alpha-beta transformation - Simulink. [online] `https://www.mathworks.com/help/mcb/ref/clarketransform.html`. Accessed: 2024-01-21.

[3] Implement alpha-beta to dq transformation - Simulink. [online] `https://www.mathworks.com/help/mcb/ref/parktransform.html`. Accessed: 2024-01-21.

[4] Ryoji Mizutani, Takaharu Takeshita, and Nobuyuki Matsui. Current model-based sensorless drives of salient-pole pmsm at low speed and standstill. *IEEE Transactions on Industry Applications*, 34(4):841–846, 1998. doi: 10.1109/28.703990.

[5] Hao Zhu, Xi Xiao, and Yongdong Li. A simplified high frequency injection method for pmsm sensorless control. In *2009 IEEE 6th International Power Electronics and Motion Control Conference*, pages 401–405, 2009. doi: 10.1109/IPEMC.2009.5157420.

[6] Shinji Ichikawa, Mutuwo Tomita, Shinji Doki, and Shigeru Okuma. Sensorless control of permanent-magnet synchronous motors using online parameter identification based on system identification theory. *IEEE Transactions on Industrial Electronics*, 53(2):363–372, 2006. doi: 10.1109/TIE.2006.870875.

[7] Lei Wang, Yifan Zhang, Linhai Zhao, and Guozhu Chen. An improved 3-step startup method based on sensorless vector control of pmsm. In *2019 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, pages 1–5, 2019. doi: 10.1109/APPEEC45492.2019.8994460.

[8] VESC Project. [online] `https://vesc-project.com/`. Accessed: 2024-05-14.

[9] Tomohiro Nimura, Shinji Doki, and Masami Fujitsuna. Position sensorless control of pmsm with a low-frequency signal injection. In *2014 International Power Electronics Conference (IPEC-Hiroshima 2014 - ECCE ASIA)*, pages 3079–3084, 2014. doi: 10.1109/IPEC.2014.6870124.

[10] Yukinori Inoue, Koji Yamada, Shigeo Morimoto, and Masayuki Sanada. Effectiveness of voltage error compensation and parameter identification for model-based sensorless control of ipmsm. *IEEE Transactions on Industry Applications*, 45(1):213–221, 2009. doi: 10.1109/TIA.2008.2009617.

[11] Hongryel Kim, Jubum Son, and Jangmyung Lee. A high-speed sliding-mode observer for the sensorless speed control of a pmsm. *IEEE Transactions on Industrial Electronics*, 58(9):4069–4077, 2011. doi: 10.1109/TIE.2010.2098357.

[12] Ahmed Lagrioui and Hassan Mahmoudi. Speed and current control for the pmsm using a luenberger observer. In *2011 International Conference on Multimedia Computing and Systems*, pages 1–6, 2011. doi: 10.1109/ICMCS.2011.5945721.

[13] Mohamad Syakir Termizi, Jurifa Mat Lazi, Zulkifilie Ibrahim, Md Hairul Nizam Talib, Mohd Junaidi Abdul Aziz, and Shahrin Md Ayob. Sensorless pmsm drives using extended kalman filter (ekf). In *2017 IEEE Conference on Energy Conversion (CENCON)*, pages 145–150, 2017. doi: 10.1109/CENCON.2017.8262474.

[14] Zhiqian Chen, Mutuwo Tomita, Shinji Ichikawa, Shinji Doki, and Shigeru Okuma. Sensorless control of interior permanent magnet synchronous motor by estimation of an extended electromotive force. In *Conference Record of the 2000 IEEE Industry Applications Conference. Thirty-Fifth IAS Annual Meeting and World Conference on Industrial Applications of Electrical Energy (Cat. No.00CH37129)*, volume 3, pages 1814–1819 vol.3, 2000. doi: 10.1109/IAS.2000.882126.

[15] Konrad Urbanski. Sensorless control of pmsm high dynamic drive at low speed range. In *2011 IEEE International Symposium on Industrial Electronics*, pages 728–732, 2011. doi: 10.1109/ISIE.2011.5984247.

[16] X-CUBE-MCSDK - STM32 Motor Control Software Development Kit (MCSDK) - STMicroelectronics. [online] `https://www.st.com/en/embedded-software/x-cube-mcsdk.html`. Accessed: 2024-05-14.

[17] Nobuyuki Matsui. Sensorless pm brushless dc motor drives. *IEEE Transactions on Industrial Electronics*, 43(2):300–308, 1996. doi: 10.1109/41.491354.

[18] Lennart Harnefors and Hans-Peter Nee. A general algorithm for speed and position estimation of ac motors. *IEEE Transactions on Industrial Electronics*, 47(1):77–83, 2000. doi: 10.1109/41.824128.

[19] Hassan K Khalil. *Nonlinear systems; 3rd ed.* Prentice-Hall, Upper Saddle River, NJ, 2002. ISBN 9780130673893.

[20] Yusheng Hu, Liyi Li, Weilin Guo, and Yan Li. Vector control of permanent magnet synchronous motor based on new mras. In *2021 24th International Conference on Electrical Machines and Systems (ICEMS)*, pages 1983–1987, 2021. doi: 10.23919/ICEMS52562.2021.9634634.

[21] Young Sam Kim, Sang Kyoon Kim, and Young Ahn Kwon. Mras based sensorless control of permanent magnet synchronous motor. In *SICE 2003 Annual Conference (IEEE Cat. No.03TH8734)*, volume 2, pages 1632–1637 Vol.2, 2003.

[22] Fang-Zheng Peng and Tadashi Fukao. Robust speed identification for speed-sensorless vector control of induction motors. *IEEE Transactions on Industry Applications*, 30(5):1234–1240, 1994. doi: 10.1109/28.315234.

[23] Young Ahn Kwon and Dae Won Jin. A novel mras based speed sensorless control of induction motor. In *IECON'99. Conference Proceedings. 25th Annual Conference of the IEEE Industrial Electronics Society (Cat. No.99CH37029)*, volume 2, pages 933–938 vol.2, 1999. doi: 10.1109/IECON.1999.816537.

[24] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems (8th Edition)*. Pearson, 8th edition, 2018. ISBN 0134685717.

[25] Colin Schauder. Adaptive speed identification for vector control of induction motors without rotational transducers. In *Conference Record of the IEEE Industry Applications Society Annual Meeting,*, pages 493–499 vol.1, 1989. doi: 10.1109/IAS.1989.96696.

[26] MATLAB. [online] `https://www.mathworks.com/products/matlab.html`. Accessed: 2024-05-21.

[27] Simulink - Simulation and Model-Based Design - MATLAB. [online] https://www.mathworks.com/products/simulink.html, . Accessed: 2024-05-21.

[28] Simscape - MATLAB. [online] https://www.mathworks.com/products/simscape.html, . Accessed: 2024-05-21.

[29] Li Yujie and Cao Shaozhong. Model reference adaptive control system simulation of permanent magnet synchronous motor. In *2015 IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 498–502, 2015. doi: 10.1109/IAEAC.2015.7428603.

[30] Maxon: EC-i 40 Catalog Page. [online] https://www.maxongroup.com/medias/sys_master/root/8882557943838/EN-21-276.pdf, . Accessed: 2024-01-21.

[31] STMicroelectronics. UM2850: Getting started with the EVSPIN32G4, EVSPIN32G4NH. [online] https://www.st.com/resource/en/user_manual/um2850-getting-started-with-the-evspin32g4-evspin32g4nh-stmicroelectronics.pdf, 2021. Accessed: 2024-05-21.

[32] Maxon: Encoder 16 EASY XT Catalog Page. [online] https://www.maxongroup.com/medias/sys_master/root/8883962478622/EN-21-466-467.pdf, . Accessed: 2024-05-21.

[33] STMicroelectronics. DS13630: STSPIN32G4 Datasheet. [online] https://www.st.com/resource/en/datasheet/stspin32g4.pdf, 2022. Accessed: 2024-05-22.

[34] STMicroelectronics. UM2570: Description of STM32G4 HAL and low-layer drivers. [online] https://www.st.com/resource/en/user_manual/um2570-description-of-stm32g4-hal-and-lowlayer-drivers--stmicroelectronics.pdf, 2020. Accessed: 2024-05-22.

[35] CMSIS-DSP: CMSIS DSP Software Library. [online] https://arm-software.github.io/CMSIS-DSP/latest/index.html. Accessed: 2024-05-22.

[36] STMicroelectronics. UM1718: STM32CubeMX for STM32 configuration and initialization C code generation. [online] https://www.st.com/resource/en/user_manual/um1718-stm32cubemx-for-stm32-configuration-and-initialization-c-code-generation-stmicroelectronics.pdf, 2024. Accessed: 2024-05-22.

[37] STM32CubeIDE - Integrated Development Environment for STM32 - STMicroelectronics. [online] https://www.st.com/en/development-tools/stm32cubeide.html. Accessed: 2024-05-19.

[38] Git. [online] https://git-scm.com/. Accessed: 2024-05-22.

[39] STMicroelectronics. RM0440: STM32G4 series advanced Arm®-based 32-bit MCUs. [online] https://www.st.com/resource/en/reference_manual/rm0440-stm32g4-series-advanced-armbased-32bit-mcus-stmicroelectronics.pdf, 2024. Accessed: 2024-05-21.