Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Camilla Marie Greve Hartviksen

Improvements on the motor control of a Formula Student racecar

Master's thesis in Cybernetics and Robotics
Supervisor: Geir Mathisen
June 2022

NTNU
Norwegian University of Science and Technology

revolve NTNU

Camilla Marie Greve Hartviksen

Improvements on the motor control of a Formula Student racecar

Master's thesis in Cybernetics and Robotics
Supervisor: Geir Mathisen
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

MASTER THESIS DESCRIPTION

Candidate:	**Camilla Greve Hartviksen**
Course:	**TTK4900 Engineering Cybernetics**
Thesis title (Norwegian)	**Forbedringer av en motorkontroller i en Formula Student racerbil**
Thesis title (English):	**Improvements on the motor control of a Formula Student racecar**

Thesis description: The racing car of Revolve at NTNU has electric propulsion. To get the desired characteristics, the current motor controller is developed by Revolve themselves. This motor controller is incrementally upgraded each year, and the development of the motor controller is the theme for this master thesis.

The main aims for this work are:

- To develop an improved motor control algorithm for the new racecar, within the consteraints given by the existing system of the car.
- To structure the software of the motor controller and to improve the process of developing motor controller software.
- Improve the calibration technique used to calibrate the encoders on the motor.

The objectives will be:

1. Conduct a literature review of motor control algorithms/methods applicable for controlling *internal permanent magnet synchronous motors* (IPMSMs).
2. Propose algorithms/systems meeting the main aims of this thesis.
3. As far as time permits, implement the suggestions from point 2 above.

Start date: 3rd of January, 2022
Due date: 6th of June, 2022

Thesis performed at: Department of Engineering Cybernetics
Supervisor: Professor Geir Mathisen, Dept. of Eng. Cybernetics

Sammendrag

Denne oppgaven ser på forbedringer av motorstyringsalgoritmen som brukes i vekselretteren til Revolve NTNU's Formula Student racerbil. Et litteraturstudie på mulige forbedringer ble gjort, og det ble foreslått et forbedret design. Forbedringene er gjort med henhold til struktur og kjøretid, som ble oppnådd ved å bruke en kodegenerert motorkontroller laget i Simulink i kodebasen til vekselretteren. Etter resultater fra litteraturstudiet ble det også laget en implementasjon av denne motorkontrolleren med en i_{d}-τ_{ref} oppslagstabell i Simulink. Dette skiftet ut de tidligere tilnærmingene til strømmen i_{d}, som før var beregnet fra et referansemoment τ_{ref} gjort med flere iterasjoner av Newtons metode.

Implementasjonen av forbedringene på motorkontrollerne ble gjennomført, men valideringen av dem ble ikke utført på grunn av mangel på utstyr og tid. Begrensningene er beskrevet mer detaljert i Section 1.2.

Det ble også gjort forbedringer på kalibreringsprosedyren til enkoderne på bilen. En prosedyre ble implementert som automatisk fant statorforskyvningsvinkelen. Dette ble validert til å ha en betydelig forbedring sammenlignet med den forrige metoden som finner offset manuelt. Tiden det tok å fullføre kalibreringen på én enkoder på én motor ble redusert fra 10 timer til 15 minutter. Denne forbedringen vil være til nytte for enkoderkalibreringsprosessen i framtidige år i Revolve NTNU.

Abstract

This report covers the research done on the improvements of the motor control algorithm used in the DC-AC inverter of Revolve NTNU's Formula Student racing car. A literature review of the improvements on motor control is done, and new designs and implementations of the improvements are proposed. This includes structural improvements and improvements in runtime by using a code-generated motor controller made in a Simulink model in the inverter. Further, after the literature review, an implementation of this motor controller with an i_{d}-τ_{ref} Look-Up Table (LUT) was made in Simulink. This switched out the previous approximations of i_{d} calculated from a reference torque τ_{ref}, which was done with numerous iterations of the Newton-Raphson method.

Validation of the implemented motor controller improvements was not done due to lack of equipment and time. These limitations are described in more detail in Section 1.2.

In addition, improvements were successfully made to the calibration procedure of the motor encoders used on the car. A procedure was implemented that automatically found the stator offset angle. This was validated to have a significant improvement compared to the previous method of manually finding the offset. The improvement led to a reduction in time for the calibration procedure, from up to 10 hours to only 15 minutes needed to finish calibrating one encoder for one motor. This improved procedure will benefit the encoder calibration process for the following years in Revolve NTNU.

Preface

This thesis includes the work on the embedded systems that are built on the work done by previous members of Revolve NTNU. Specifically, the work done that is described in Section 3.1 and Section 3.2. Both the Formula Student racecar that is developed by Revolve NTNU for the season this thesis was written in, and the ones made in previous seasons, are all built on the base that alumni have created, and continue to create each year.

Acknowledgements

Going into Revolve NTNU was a difficult decision, and one based on my scarce practical knowledge in electrical and mechanical systems exiting my fourth year in my master degree. Throughout my years in Cybernetics and Robotics, I've gravitated more and more towards taking the harder, tougher route with more challenges and higher stakes, but also higher rewards. This was exactly what Revolve both offered and delivered through with. The learning curve was steep and difficult, and I still have much to learn within the field, but I feel that I have put all the theory I've learnt throughout my years in school into practice. Now, I feel much more confident going into the work force with the tools and knowledge I've obtained. I highly recommend joining this student organization for anyone with even just a slight curiosity and motivation for learning more within your field. It will be hard, but it will be so worth it.

I want to thank the members of Revolve NTNU, both current and past, who have contributed to this thesis. Håkon Skeie, Jan Ottar Seljebu Olsen, Simen August Tinderholt and Francesco Fanin are all alumni on the inverter that have provided immense help. They have always been glad to help, and have responded to my inquires in a moment's notice. Additionally, this thesis would not have been possible without the amazing help from Eskil Mogstad and William Karl Moriggi, that are current members that have been essential in the improvements on the calibration routine.

I also want to thank my thesis advisor, Geir Mathisen, whose meetings I've always looked forward to, and who have urged me to think outside the box and evaluate each decision I've wanted to make in regards to the thesis.

A special thanks goes out to my group leader and rich person, Niklas Strømsnes, and brother in-law and professional bolle, André Kapelrud, who both proofread this thesis. I'm sorry for the number of times you've had to read the word "thus" and "gotten".

I finally want to thank my family for supporting me. This includes my two awesome brothers and biggest idols growing up, who always have my back. My sister, who I look so much up to in all things, and her family who have made Trondheim feel like home. I've been so proud and happy to watch my niece and nephews grow up here. My brother and sister in-laws, who I have the joy and privilege to be an extra little sister to. My pap, who taught me to always work hard. Lastly, mijn sterke mam, who inspires me to take the tough route every day.

This year has been a year of change for me. Not only have I improved my practical knowledge and skills, but this year brought on some personal changes as well. Through it all, however, I've gained so many memories and I've made strong connections and friends for life.

Writing this got me thinking about the girl I was 5, 10 and 15 years ago, and I'm so hecking proud of how far I've gotten. Not only academically, but

personally as well. This is a result of all of tougher choices I've made, and the harder routes I've taken throughout the years that scared me, challenged me and made me feel little and stupid. Those choices are the ones that benefited me the most, and has made me into the awesome Camilla I am today.

So I say to you, reader, take the tough path. Run towards what scares you. Gønn på!

...and support young girls going into STEM!

Acronyms

- **APU** - Application Processor Unit
- **DV** - Driverless Vehicle
- **EV** - Electrical Vehicle
- **I19** - The first self-made generation of inverter, by Revolve NTNU
- **I21** - The latest self-made generation of inverter, by Revolve NTNU
- **LUT** - Look-Up Table
- **MPSoC** - Microprocessor System on Chip
- **PL** - Programmable Logic
- **PMU** - Platform Management Unit
- **R21** - The Revolve NTNU team for the 2020-2021 season
- **R22** - The Revolve NTNU team for the 2021-2022 season
- **RPU** - Real-time Processor Unit
- **SDK** - Software Development Kit
- **VCU** - Vehicle Control Unit

Contents

1 Introduction

This thesis was done in collaboration with Revolve NTNU. Revolve NTNU is a student organization run solely by student volunteers that — in parallel with full-time studies — design and produce the mechancial and electrical systems behind an electric racecar every year. Figure 1.1 shows the car made for the 2021-2022 season, *Aurora*.

The organization competes in the world's largest engineering competition, Formula Student, competing with over 114 universities from 38 countries. In eight months, the around 60 members participates in the design and production of the systems needed for a racecar. Revolve NTNU that has placed on top 3 in the biggest Formula Student competition, *Formula Student Germany* (FSG). The specific technical groups are as follows:

- **Electrical engineering** — responsible for the accumulator and for all electronics and embedded hardware and software, consisting of the groups
  - Embedded systems
  - Power systems
- **Mechanical engineering** — responsible for all technical systems ranging from complex composites to gears and motors, consisting of the groups
  - Aerodynamics
  - Chassis
  - Suspension and Drivetrain
  - Driver Interface
- **Software Engineering** — responsible for making both the *Electrical Vehicle* (EV) and the *Driverless Vehicle* (DV) run as smooth as possible, and for making Revolve's self-made analysis software; Revolve Analyze. It consists of the groups
  - Autonomous Systems
  - Control Systems
  - Software Development

In addition to these groups, Revolve also has its own racecar drivers, its own board, and a marketing group that deals with all the financials, contacting sponsors, and managing all events and social media that put Revolve NTNU on the map.

Revolve competes in both EV (Electrical Vehicle) and DV (Driverless Vehicle) competitions. This season, FSG merged the two competitions. Because this seasons team in Revolve, team R22, has a goal of placing top 3 in FSG, this will be the first year that Revolve makes a merged EV and DV vehicle. [1]

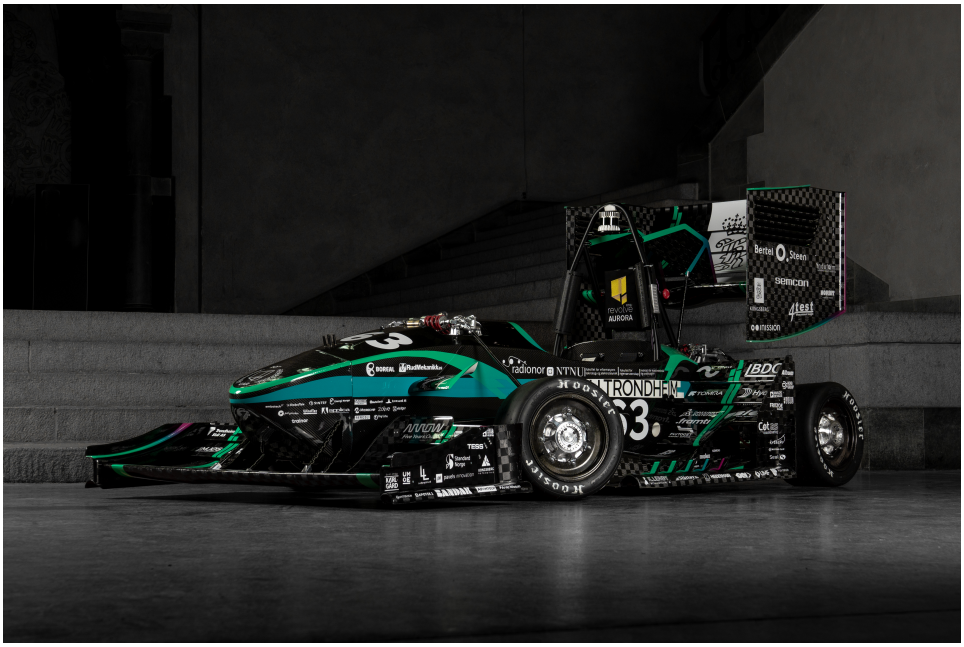


Figure 1.1: A photo of the R22 car, *Aurora*.

It took four years for Revolve NTNU to develop an in-house inverter implementation, with the first one, named I19, ready for use in 2019. Before this, OEM solutions have been used instead. The inverter is one of the the most complex piece of hardware and software on the car. During the '20/'21 season, development of the second generation of the inverter (I21) was started. The work described in this thesis is a further refinement of the development of the I21 that will be used in Revolve NTNU's four-wheel-driven electric vehicle.

1.1 Background

The first in-house made inverter inverter in Revolve NTNU was called I19, and took 4 years to make. It is a reliable inverter and has been used for multiple seasons, including this season; see a render of the placement of the I19 in the R22 car in Figure 1.2. Parallel to when I19 was finished in 2019, a combined simulator of the motor control and the whole mechanical system was made in Simulink. The motor controller of this simulator employed rotational speed in rpm as input, and through the use of a speed controller, a current controller and a pulse width modulator, outputted duty cycles meant as input to the transistors in the inverter. This system was only meant to be used as a model to implement, test and validate the mathematics behind the motor control algorithm. When the second generation of the in-house made inverter was developed on in 2021, called I21, its motor controller was written in C code based directly on the motor controller from the simulator. Having a sizeable motor control algorithm with a large amount of mathematic calculations in pure C code is not very structured, and can be hard to read and debug. Instead of using this solution, the motor controller can be distilled directly from the Simulink model, by means of Simulink's code generation capabilities. This generated C code could be optimized based on objectives like RAM-, ROM- or execution efficiency. These are all code generation objectives that could be easily selected and put in a prioritized order of importance in Simulink.

The motivation for improving the current implemented motor control algorithm is to make it more structured with the use of a modular block system made in Simulink, and faster through the use of optimization strategies that generated code offers. In the future, when further work is to be done on this motor controller, the simulator will be up to date with the newest system as well. Having a simulator of a mechanical system is highly beneficial in an industry setting, as it determines up front if design goals are met without having to risk electrical and mechanical parts for a test run. Simulations are also faster to perform than a mechanical run-through of the motor, which would have to include setup time. Simulations are therefore likely to be more cost efficient than testing mechanically. Simulations also allows analysing the performance and the energy usage of the model to see if the model parameters meets all the requirements before production is started. This includes parameters like power consumption, as well as current, torque and speed responses. Simulators that are up to date with the mechanical system are therefore beneficial to have. This is because simulating running the motor is an important and critical part of testing the motor control design, instead of risking money and safety by testing designs directly on the motor.

This will make it easier to work on and improve upon the motor control algorithm, because an interface in a Simulink diagram, which is a program made specifically for mathematical functions, is easier to work with for this purpose than a large C code base. For instance, Simulink offers finished solutions like built-in PID-blocks that would be cumbersome to write in C code. Moreover, a switch like this would ensure that the simulator is always up to date with the motor controller that is actually running on the car. This is not ensured at the moment, because the motor controller that is run on the car is often worked on independently of the simulator.

Thus, another goal is to use Simulink's code generating capabilities, which will generate a function with set inputs and outputs that correspond with the model. This function will then be merged into the existing I21 codebase.

Figure 1.2: The figure shows the systems inside the car, where the inverter casing is on top of the yellow accumulator package behind the seat.

Furthermore, this thesis investigates different possibilities for improving the motor control algorithm. Currently, a Newton-Raphson method is used to find an approximation of the current, which is calculated from the torque. This is time consuming and can be switched out with a pre-made *Look-Up Table* (LUT) of torque and current measurements. This will improve the runtime of the motor control algorithm, which will it possible to either reduce the switching time, and therefore increase the peak rotational speed in rpm, or spend the remaining time of the current switching period on other tasks. A suggestion of implementation of this is presented in the design chapter, but the actual implementation of this wasn't done because of lack of resources.

It is vital that the encoder sends accurately calibrated position sensor data back to the motor controller for the motor controller algorithm to run as efficient as possible. This is currently done through manual calibration. If the encoder isn't calibrated accurately, more current is required to reach the peak rotational speed of the motor. Additionally, the calibration has to be done all over again if the encoder is accidentally rotated at any point, which has happened in previous years in Revolve NTNU during periods where several people have worked in tight-knit settings in the workshop.

1.2 Limitations

Because the author could not acquire a torque sensor during the duration of the writing of this thesis, the *Look-Up Table* (LUT) of the q-current i_q and the torque reference τ_{ref} was not made. In addition to NTNU, several local companies were contacted to try to obtain this without any luck.

In addition, attempts were made to contact the motor company itself that Revolve NTNU has a sponsor contract with, Fischer Elektromotoren, to get a completed measured LUT for the specific motor that the organization uses. The requests were, however, not responded to.

After merging the code-generated Simulink motor controller model with the existing C code base on the I21, there were issues found with the existing code base. Because of unsolved repeated errors in the firmware Xilinx Vivado and Xilinx Software Development Kit, respectively used to generate bitstreams from the FPGA and compile the existing C code base on the I21 MPSoC, the code generated motor controller could not be run nor validated. These build errors were attempted to be debugged without success, and because of time limitations, the attempt to validate the implementation was discontinued.

When manually calibrating the encoders, the measured .csv-files of the measured data were not saved; only the screenshots from the runs were saved. This includes the figures Figure 6.2 and Figure 6.1.

2 Motor control theory

In this chapter the background theory necessary to understand the different methods of motor control will be presented. Some chapters are refinements from the project thesis [2] written in the fall of 2021.

2.1 Motor topology

The racecar made in the '21/'22 season consists of a 4WD electric powertrain, where each wheel has a hub-mounted *interior permanent magnet synchronous motor* (IPMSM) connected to gearboxes.

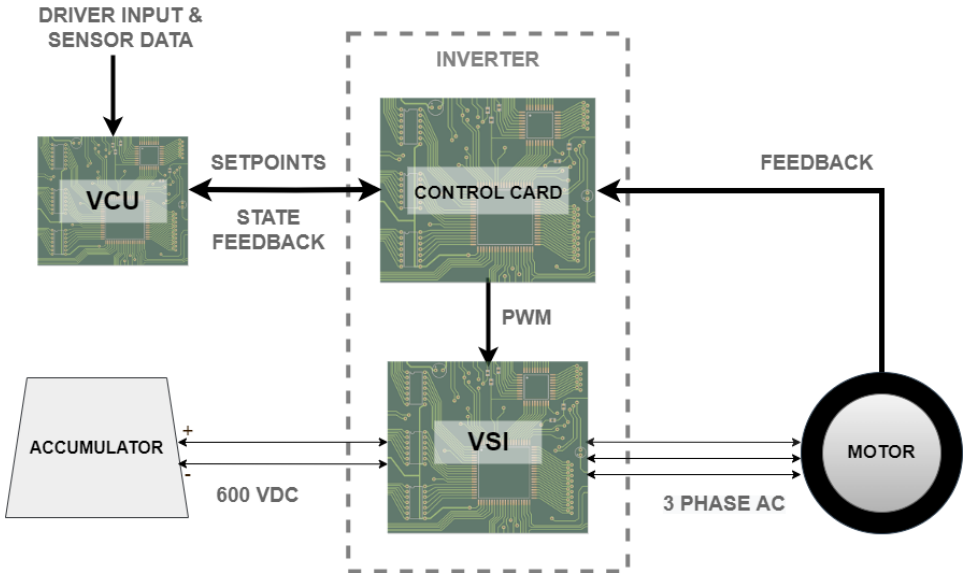


Figure 2.1: Image showing topology and connections between the inverter, the VCU, the motor and the accumulator.

What is called the *inverter* in the car consists of multiple cards; a control card and multiple *Voltage Source Inverters* (VSI). The control card acts as the brain of the inverter, see Section 2.6, and communicates with one other PCB on the car, called the *Vehicle Control Unit* (VCU). The VCU handles sensor data and driver inputs from the car. The control card receives these setpoints and uses them to compute control signals in the form of *Pulse Width Modulation* (PWM) signals to send to the motors via the VSI cards — more details on PWM are introduced in Section 2.9.4. The control card also sends back state feedback data back to the VCU.

A higher level topology of the motors can be seen in Figure 2.1, where the accumulator not only provides energy, but also receives energy by regeneration during deceleration. By having the opportunity to control each motor individually, the traction of each wheel can be maximized. Torque is also distributed between all four wheels depending on the state of the car, the tires, and the driver's inputs. This is done using a self-made *Torque Vectoring* (TV) algorithm.

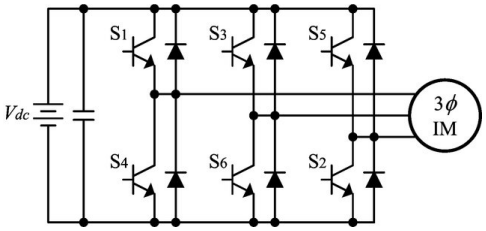


Figure 2.2: The figure shows a two-level, three-phase inverter with three half-bridges, where each S_n represent a transistor. Image gotten from [3]

Each VSI contains six transistors which make up three half-bridges that receive the PWM signals from the control card and send out a phase output to the motor, seen in Figure 2.2. In addition to this, the power stages have DC-link capacitors that act as load-balancing storage and prevent voltage spikes and electromagnetic interference.

2.2 Clarke- and Park-transform

The three phase currents going to the stator can be calculated from the stator current I_s and the electrical angular velocity ω_e as

$$\begin{bmatrix} i_{\mathrm{a}}(t) \\ i_{\mathrm{b}}(t) \\ i_{\mathrm{c}}(t) \end{bmatrix} = I_s \begin{bmatrix} \sin(\omega_{\mathrm{e}} t) \\ \sin(\omega_{\mathrm{e}} t - \frac{2\pi}{3}) \\ \sin(\omega_{\mathrm{e}} t - \frac{4\pi}{3}) \end{bmatrix}, \tag{1}$$

where ω_{e} is the product of the mechanical angular velocity ω_m and the number of pole pairs p in the motor.

When regulating the currents in the motor control algorithm, it is more desirable to work with two direct currents instead of sinusoidal currents. This is to prevent having to input values in a sinusoidal form. This can be achieved using the Clarke and Park transforms, also called the direct-quadrature-zero. [4]

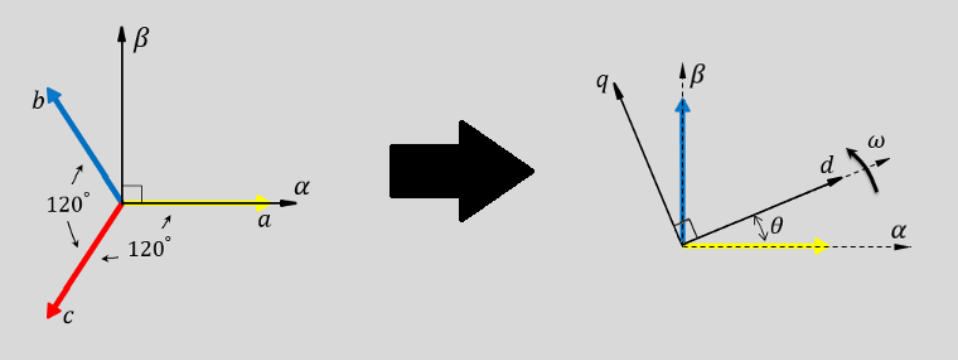


Figure 2.3: On the left, the Clarke-transform is first performed on the *abc*-system to turn it into a two-phase $\alpha\beta$-system. Following that, a Park-transform is performed on the $\alpha\beta$-system to turn it into a dq-system, which are two axes that are parallel and perpendicular with the rotor flux axes. [4] Figure made from figures from [5] and [6].

The Clark transform turns the three-phase currents i_{a}, i_{b} and i_{c} into two-dimensional currents i_α i_β, as in

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ 0 & \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_{\mathrm{a}} \\ i_{\mathrm{b}} \\ i_{\mathrm{c}} \end{bmatrix}. \tag{2}$$

The i_α current is parallel with the i_{a} axis, while the i_β current is perpendicular to the i_{a} axis, see Figure 2.3.

Furthermore, the $\alpha\beta$-system can be turned so that it is along the same axes that the rotor flux is in. The axis that is directly parallel with the primary rotor flux axis is known as the d-axis (direct axis). The axis that is perpendicular to the primary rotor flux axis is called the q-axis (quadrature axis). When the stator flux is along this axis, the motor creates maximum torque. [4]

This transformation from $\alpha\beta$ to dq is done through the Park transform, which is defined as

$$\begin{bmatrix} i_{\mathrm{d}} \\ i_{\mathrm{q}} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix}. \tag{3}$$

2.3 Motor equations

As derived in [7], the equations for a general three-phase synchronous motor can be defined as

$$u_{\mathrm{ij}} = R_{\mathrm{j}} I_{\mathrm{ij}} + \frac{\partial \psi}{\partial t} \qquad i \in \{a, b, c\}, \quad j \in \{s, r\}, \tag{4}$$

where the index j appoints to either the rotor or the stator coil windings and the i appoints to the three phases. The first term contributes to the voltage across the stator windings, and the latter term contributes to the counter-electromotive force. [7] ψ is the magnetic flux linkage in the system, defined as $\psi = N\phi$, which essentially represents the total flux in all N turns of the coil.[8] The variation of the flux linkage over time induces a voltage, as in

$$u(t) = \frac{\partial \psi}{\partial t}. \tag{5}$$

Further derivations made in [7] show that after applying the Clarke- and Park-transforms on Equation (4), the transformed voltage equations will be in the dq-frame as in

$$\begin{aligned} v_{\mathrm{d}} &= R_s I_{\mathrm{d}} + \frac{1}{\omega_{\mathrm{n}}} \frac{\partial \psi_{\mathrm{d}}}{\partial t} - p\psi_{\mathrm{q}} \\ v_{\mathrm{q}} &= R_s I_{\mathrm{q}} + \frac{1}{\omega_{\mathrm{n}}} \frac{\partial \psi_{\mathrm{q}}}{\partial t} + p\psi_{\mathrm{d}} \end{aligned} . \tag{6}$$

where the flux linkage ψ, the angular velocity ω and the reactances (inductances) x are defined as

$$\begin{aligned} \psi_{\mathrm{q}} &= x_{\mathrm{q}} i_{\mathrm{q}} \\ \psi_d &= x_{\mathrm{d}} i_{\mathrm{d}} + \psi_{\mathrm{m}} = x_{\mathrm{d}} i_{\mathrm{d}} + x_{\mathrm{m}} i_{\mathrm{f}} \\ \omega_{\mathrm{n}} &= \frac{1}{n} \frac{\partial \theta}{\partial t} \\ x_{\mathrm{d}} &= x_{\mathrm{md}} x_{\mathrm{s}\sigma} \\ x_{\mathrm{q}} &= x_{\mathrm{mq}} x_{\mathrm{s}\sigma} \end{aligned} . \tag{7}$$

In Equation (7), n is the speed in rpm per unit, θ is the angle between the rotor and stator flux, i_f is the field current per unit, and finally, x_m is the mechanical reactance, where the σ index appoints to a leakage reactance. [7]

Furthermore, the motor torque in the dq-frame is the difference between the flux linkage and current in the dq-frame, $\tau_e = \psi_\mathrm{d} i_\mathrm{q} - \psi_\mathrm{q} i_\mathrm{d}$. [7] Using the Equation (7), the flux linkages from the dq-frame can be expressed through the reactances from the same frame, thus the torque can be expressed as

$$\begin{aligned} \tau_e &= \psi_\mathrm{d} i_\mathrm{q} - \psi_\mathrm{q} i_\mathrm{d} \\ &= \psi_m i_\mathrm{q} - (x_\mathrm{q} - x_\mathrm{d}) i_\mathrm{d} i_\mathrm{q} \\ &= x_m i_f i_\mathrm{q} - (x_\mathrm{q} - x_\mathrm{d}) i_\mathrm{d} i_\mathrm{q}. \end{aligned} \tag{8}$$

The torque is thus made up of two terms, the first term stems from the electromagnetic torque generated from the stator, and the last term stems from the reluctance in the motor due to saliency, described in Section 2.4. To increase torque and benefit from the reluctance in the last term, i_d has to be negative.[7] This will be elaborated on in Section 2.9.2.

2.4 Motor specification

The motor used in Revolve is from Fischer Elektromotoren, and some important values from the motor's datasheet can be seen in Table 1. The data sheet for the motor is in the appendix, Section 9. The motor is a wye configuration, 4 pole motor — weighing in at 3.37 kg.

Data	Symbol	Value
Nominal Torque	T_Nominal	11.1 Nm
Nominal Current	I_Nominal	22.6 $\mathrm{A_{rms}}$
Nominal Speed	n_Nominal	13250 rpm
Nominal Power	P_Nominal	15404 W
Peak Torque	T_Peak	29.1 Nm
Peak Current	I_Peak	61 $\mathrm{A_{rms}}$
Peak Speed	n_Peak	20000 rpm
Peak Power	P_Peak	35366 W
Inductance, d-axis	L_d	0.27 mH
Inductance, q-axis	L_q	0.37 mH

Table 1: Key values from the motor data sheet of the motor Revolve NTNU uses, the Fischer Elektronik TI085-052-070-04B7S-07S04BE2. The data sheet for the motor is in the appendix, Section 9

The stator voltage and current are limited by their direct and quadrature parts as

$$\begin{aligned} i_\mathrm{peak}^\mathrm{stator} \geq i^\mathrm{stator} &= \sqrt{i_\mathrm{d}^2 + i_\mathrm{q}^2} \\ u_\mathrm{peak}^\mathrm{stator} \geq u^\mathrm{stator} &= \sqrt{u_\mathrm{d}^2 + u_\mathrm{q}^2}. \end{aligned} \tag{9}$$

The motor has some saliency that must be considered when calculating the torque equations. Saliency is a measure of the lack of symmetry between the rotor and stator in the motor that leads to a non-uniform air gap and can create a reluctance torque in the motor. The difference can be seen between the inductance parameters L_{d} and L_{q}, see Table 1. The relationship between the two is that $L_{\mathrm{q}} > L_{\mathrm{d}}$, which is traditional within IPMSM. This means that the flux along the d-axis is calculated along the path with higher reluctance than the q-axis that goes through the path with the lowest reluctance. [7]

2.5 Workings of an inverter

The inverter takes in DC power from the accumulator and turns it into three-phase AC power for use in the motor's stator. A controller can then modulate the amplitude, frequency and phase of these three sinusoidals to create a magnetic field vector in the stator perpendicular to the one created from the rotor magnetic field. When these two fields are 90° in relation to each other, this will generate maximum torque in the motor. Increasing amplitude would increase power to the motor, and increasing frequency would increase the rpm of the motor.

The inverter uses three half-bridges for this purpose; see Figure 2.2, thus it is considered three-phase. Each transistor in the half-bridge act as a switch, where the motor control decides which one of them should let current through at any moment. By switching which transistors the current can go through, commutation happens in the motor. The motor has four pole pairs, so commutation happens every 15°.

2.6 Specifications of the I21 inverter

Its control card is based on the Enclustra XU5, containing a Xilinx Ultrascale+ MPSoC, which features 4x A53 cores at 1.3 GHz and 2x R5 cores at 500 MHz including a powerful and compact FPGA board. Figure 2.4 is a render of the control card. [9]

I21 will be responsible for powering 4 35 kW IPMSMs at 35 kW each, see Section 2.4, developing a maximum torque of 29.1 Nm each, and reaching a maximum rotational velocity of 20 krpm. It will do this by converting direct current from a 600 VDC battery accumulator to three-phase AC with variable magnitude, phase, and frequency.

The I21's VSI is based around a 1.2kV SiCMOSFET six-pack module. It features shunt current measurement and several overcurrent protection mechanisms to prevent heat damage in the motor. This includes desaturation protection for the module and I^2t-based (current squared over time) motor overload protection, which keeps track of the RMS current in the motor windings.

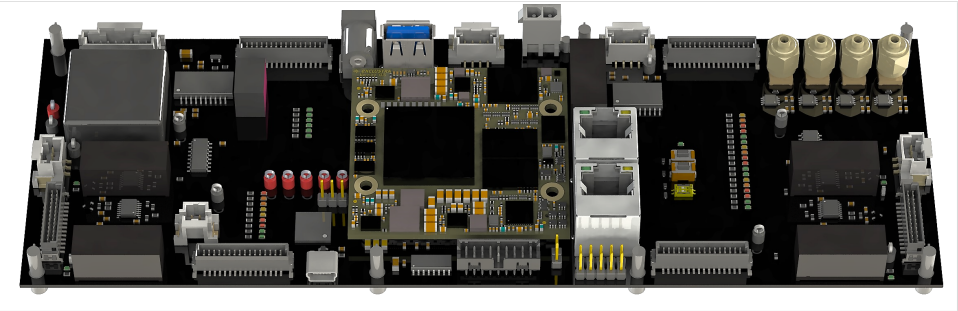


Figure 2.4: A render of the I21 control card, exported from the Altium software used to design Revolve's PCBs.

2.7 Motor encoder

The motor encoder that Revolve uses is mounted on each of the four motors on the car, and has been used in Revolve since 2016. The model comes from a German company called Heidenhain, see Figure 2.5. It is an absolute internal rotary encoder without integral bearing, which means that the way the encoder is mounted determines how well it performs and functions. This will be elaborated on later in this chapter in relation to the scanning gap of the encoder.

Figure 2.5: Revolve NTNU's motor encoder, the Heidenhain ECI1118.

The IPMSM used in this system requires the rotor position as soon as the car is turned on. This is not possible with an incremental encoder. They simply count an incremental amount of a particular pattern on the edges of the rotating unit at start-up from its start position. This is, however, possible with the absolute rotary encoder as used in this application, as each point on the edge of the disc has a specific binary code identifiable with light and dark patterns. This ensures that the rotor position is known at start-up, even after the loss of supply.

Accuracy in the reading of the rotor position is directly related to the motor control, and this means that the encoder needs to be in mutual alignment mechanically with the rotor in the motor. The datum shift is a value that is added to the physical position of an encoder for this purpose. A datum is a reference frame that is based on the zero point (origin), the shape and orientation of the encoder geometric disc, and several known control points. Thus, during mounting, it is necessary to physically align the encoder properly with the rotor and get a relation between the datum of the two. Then, when a datum shift is applied in the software of the encoder, the disparity between the two datums is so negligible that the encoder should, in theory, be finished calibrated. [10]

Heidenhain makes their own *Adjusting and Testing Software* (ATS) that interfaces through their proprietary diagnostic tool called PWM 20. It connects through USB and includes a phase angle measuring unit and is where the datum shift is set. It has built-in tolerances that are deemed appropriate for the calibration of the encoder. The limits on the voltage amplitude are 0.20 Vpp and 1.30 Vpp, and the ATS will output errors if the measured voltage amplitude goes outside these limits. [10]

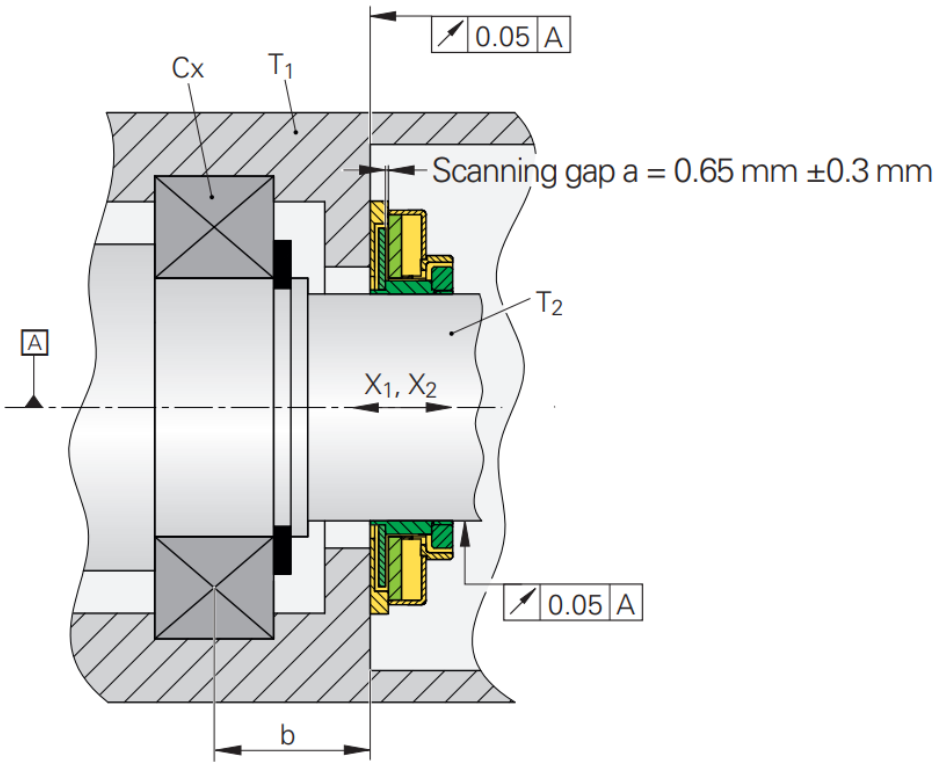


Figure 2.6: A schematic drawing of the ECI1118. The figure is from [10].

The alignment happens in the *scanning gap* of the encoder, as can be seen in Figure 2.6. This is the distance between the rotor and the stator, and can be measured indirectly with the ATS as a signal amplitude. This measurement gives out a peak-to-peak voltage amplitude in percentage, and the optimum scanning gap has a peak-to-peak voltage amplitude of 100%. The aforementioned voltage limits recommended by the software is equivalent to a percentage interval between 80-120 %. This recommendation equates to the shaft in the encoder and the scanning gap being allowed to have a mechanical range of motion of -0.33mm and 0.1mm during operation of the motors. This small range emphasizes the importance of a tight and accurate mounting of the encoder. [10]

The data transfer of this value from the encoder to the microcontroller is done through Heidenhains proprietary bidirectional interface for encoders, called EnDat, and it transmits the absolute position of the encoder. It facilitates with a direct reading of the actual physical position. As with other encoder interfaces like *Synchronous Serial Interface* (SSI), EnDat encoders transmit absolute position data on demand.

The encoder is connected to the FPGA on the Enclustra XU5-module on the control card. Position feedback is essential through the use of sensored motor control. The angle of the rotor shaft compared to the stator provided by the encoder with 18-bit res. is sent in as input into the FPGA, which then acts as an interface and transmits the data to the software in the Ultrascale+ card. Using the EnDat 2.2 protocol, a digital bidirectional interface for encoders, position values are transmitted in sync with the clock signal in the electronics it is connected to. EnDat uses specific optimization, which helps to reduce electromagnetic interference through the transmission of sensor data. [11]

The Endat 2.2 protocol is integrated into the FPGA design through an *Advanced eXtensible Interface* (AXI) to *Advanced Peripheral Bus* (APB) bridge, which makes it possible to interface with the Endat IP package in the FPGA from the software in the Ultrascale+. With the use of interrupts in the FPGA, the encoder sampled at the extremum points of the PWM carrier signal, see Figure 2.11. This interrupt notifies the processor core in the software running on the Ultrascale+ ARM cores, which runs an iteration of the motor control algorithm after receiving this data.

2.8 Software architecture

An overview of how the three cores interact with each other can be seen in Figure 2.7. The A53-core receives sensor data from the VCU card (Vehicle Control Unit), see Section 2.1, and sends the setpoints to the controllers in the two R5-cores based on this. The R5-cores send control signals to the VSI (Voltage Source Inverter), which then controls the motor. The motor sends feedback data back to the controller, which again sends back state feedback data to the A53-core and the VCU.

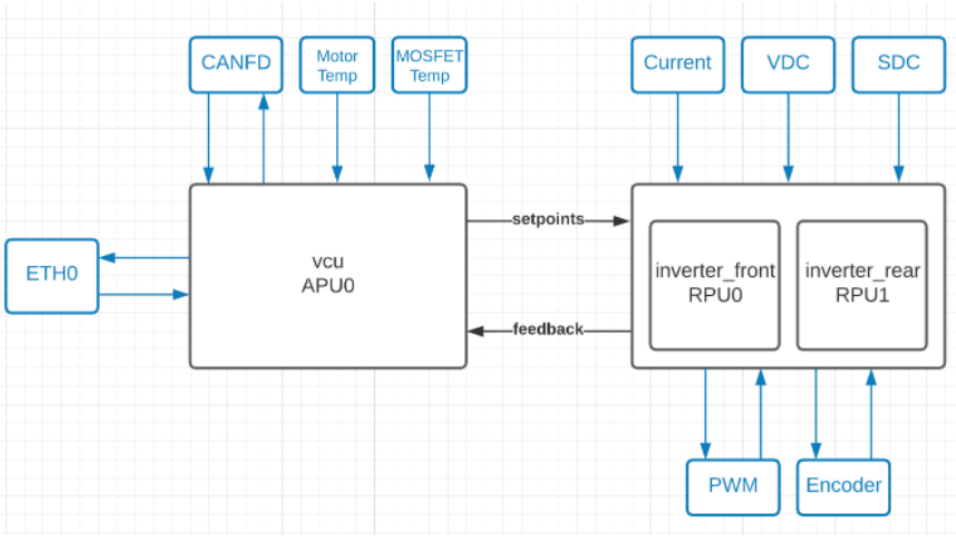


Figure 2.7: This diagram shows how the cores interact with each other and some of the rest of the system.

The one A53-core in use on the *Application Processor Unit* (APU) is referred to as the `vcu` in the software since it handles information received from the VCU. The two R5-cores in use in the *Real-time Processing Unit* (RPU), are called `inverter_front` and `inverter_rear` in the software since they run the motor control algorithm for two motors each in the front and in the rear.

The motor control algorithm running in each of the R5 cores is based on the carrier signal that runs on 20kHz. The encoder is sampled at every maximum and minimum peak of the carrier wave, and interrupts the Endat 2.2 protocol in the FPGA. The interrupt is connected from the FPGA to the A53-core, which triggers an iteration of the control loop in each of the R5-cores. The control loop runs the motor control algorithm every minima of the carrier signal, and right before it is triggered, sensor data is read from the ADC.

After the control loop is finished running, the main in the A53-core runs other less prioritized tasks not operated at switching frequency until the next minima, and an interrupt occurs one period later.

Thus, the control loop runtime is limited by the switching period of 1/(20kHz). If the runtime of the control loop is too close to the switching period, lower priority tasks can't be executed in time. This is called a timeout condition, where the timeout is the period between interrupts.

2.9 Motor control methods

As can be seen in Table 1, the motor has some restrictions in nominal speed when the torque reaches the output limit of 11.1 Nm. To exceed this value, there are some control methods with added solutions that can be implemented that can work against this restriction. In this thesis, *Direct Torque Control* (DTC), *Field Oriented Control* (FOC) and field weakening control will be presented, as well as some methods for modulating both the output and the input signals to these algorithms. [7]

2.9.1 Field Oriented Control (FOC)

The motor control happens with the use of Field Oriented Control (FOC). Essentially, the FOC algorithm consists of the following steps:

1. Measure the angular position of the rotor, in addition to the phase currents i_a, i_b and i_c.
2. Run the Clarke/Park-transform, see Section 2.2, on the measured currents, outputting i_d and i_q.
3. Compute the reference currents from the setpoints from the VCU and the measured angular motor position.
4. Run the PI-controllers for both i_d and i_q.
5. Run the inverse Clarke/Park-transform on the output from the PI-controllers, now outputting a three-phase signal.
6. Calculate the duty cycles going into the VSI from the three-phase signals using modulation. [7]

Measuring the position of the rotor is done with the use of a motor encoder, see Section 2.7. From the power stage cards, the measurements of the DC link voltage and the phase currents come as input to the control card and the FOC with the use of shunt resistors. Since the motor is in a wye configuration, only two phase currents need to be measured as the third one can be calculated from the two.

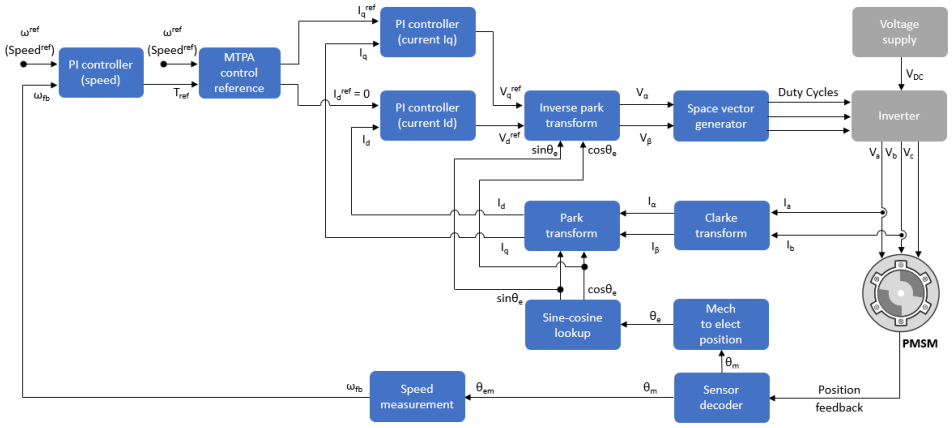


Figure 2.8: A block diagram of FOC, showing the feedback regulation of the speed through the inverter and the motors using the Clarke-Park transform and PI-controllers. Figure inspired from [12].

As described in Section 2.2, these currents can be turned two-dimensional, and sent through a PI-controller together with the reference currents calculated from the setpoints received from the VCU. Figure 2.8 shows a high-level block diagram of a FOC motor control algorithm.

2.9.2 Field Weakening Control

Field weakening control allows the motor to run at speeds higher than its nominal speed. This is done by reducing the *Back-EMF* (BEMF) of the motor. When the BEMF is reduced, the speed ω_r can be raised with no restrictions from opposing forces. The BEMF of the motor is defined as in $U_{EMF} = \psi\omega$, where ψ is the flux produced in the motors gap and ω is the motor speed in rpm.

Reducing the BEMF U_{EMF} can be done by reducing the magnetic flux of the stator ψ. The reduction happens along the d-axis by introducing a negative flux along this axis, which by Equation (7) can be made possible by introducing a negative i_d.

The goal of field weakening is to maximize torque. This can be done with adjusting the currents in relation to the maximum current. The currents and voltages are limited by their dq-parts, see Equation (9). Those equations can be illustrated as in Figure 2.9, with theoretical limits for each variable shown with circles.

The first trajectory **I** in the figure illustrates the trajectory to be taken to achieve the maximum stator voltage with nominal torque, **II**, and thus *Maximum Torque Per Ampere* (MTPA). In the second trajectory, the torque τ is decreaced to increase the speed ω.

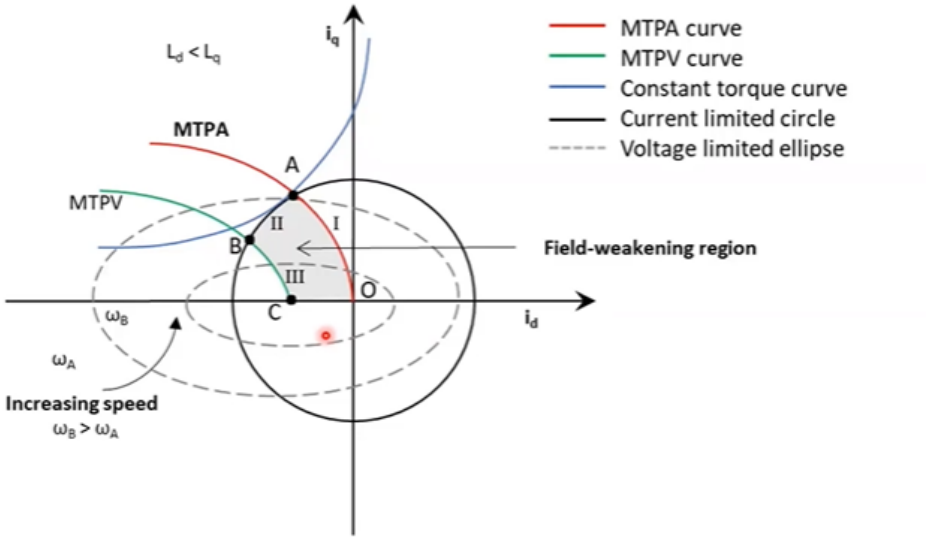


Figure 2.9: This figure shows the different trajectories I, II and III that the current can go along, with respect to the maximum current. The dashed ellipse represents the voltage limitations wrt. the speed. The gray area enclosed by the trajectories is called the field-weakening region. Figure inspired from [13].

After the second trajectory, MTPA can no longer be used to increase the torque of the motor, thus **III** illustrates the trajectory to be taken to achieve *Maximum Torque Per Voltage* (MTPV), or *Maximum Torque Per Flux* (MTPF). In each trajectory, the limits come from the bounds from the DC link voltage and the motor specification.

As derived and defined in chapter 7.4.1 *"Optimal ϕ_c control"* in [7], the currents needed to achieve MTPA and thereby represent the current trajectory I in Figure 2.9 are defined as

$$\begin{aligned} i_d^{\text{MTPA}} &= \frac{\frac{\psi_m}{3} - \sqrt[3]{(\frac{\psi_m}{3})^3 + \frac{(L_q - L_d)^2 \tau_e}{3\psi_m}}}{L_q - L_d} \\ i_q^{\text{MTPA}} &= \frac{\tau_e}{\psi_m + i_d(L_d - L_q)} \end{aligned} . \tag{10}$$

2.9.3 Direct Torque Control (DTC)

DTC is similar to FOC in which it runs a control algorithm for both torque and flux, however it does this without the delay of running a modulation, thus the algorithm runs much faster. In addition, it does not require the rotor position for speed control, thus relieving the need for a motor encoder in the system. [7] Figure 2.10 shows a high-level block diagram of an implementation of DTC.

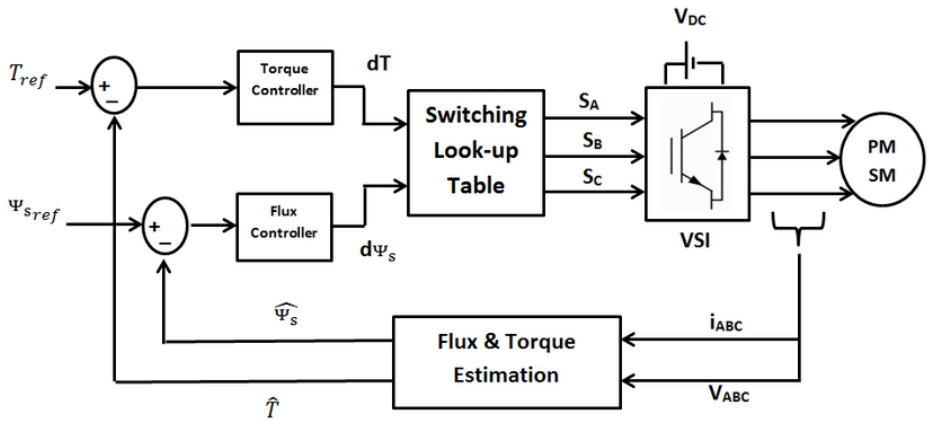


Figure 2.10: A high level block diagram of DTC, showing the torque and flux reference inputs and the torque and flux estimated from the current and voltage measurements, and the outputs from the the switching table going into the inverter. Figure inspired from [14].

Torque and flux are the inputs to this algorithm, and they are proportionate to the voltage levels and vectors created in the motor, as described in Section 2.9.4. The magnitude of flux can be increased or decreased in relation to which magnitudes the direct components of the voltage vectors have in the sector in question. The indirect components of the voltage vector can give an increase or decrease in torque.

Flux	**Torque**	**Sector**					
		S_1	S_2	S_3	S_4	S_5	S_6
$\psi_{\text{stator}} = 1$	$\tau_{\text{output}} = 1$	V_2	V_3	V_4	V_5	V_6	V_1
	$\tau_{\text{output}} = -1$	V_6	V_1	V_2	V_3	V_4	V_5
$\psi_{\text{stator}} = -1$	$\tau_{\text{output}} = 1$	V_3	V_4	V_5	V_6	V_1	V_2
	$\tau_{\text{output}} = -1$	V_5	V_6	V_1	V_2	V_3	V_4

Table 2: Switching table for choosing vectors in DTC that can increase or decrease the flux and torque.

A switching table, see Table 2, is used to show the impact that each voltage vector has on the torque and flux inside any given sector the vector is in. By using two hysteresis controllers, which are ON/OFF-controllers that filter high-varying input values so that the output varies more smoothly, the torque and flux can be increased or decreased according to this LUT. [7]

2.9.4 Modulation methods

There exists multiple *Pulse Width Modulation* (PWM) methods in the world of motor control, where the most widely used are described in this chapter. The modulation techniques are applied on the last step of the control algorithm, before the signals are sent out to the VSI. The speed of the motor is regulated with the use of pulse width modulation. [7]

Sinusoidal Pulse Width Modulation (SPWM)

The duty cycle is decided by the amount of time each of the transistors remain in an open position during a period. Figure 2.11 shows the duty cycle of a square wave signal in relation to the shape of its respective sinusoidal wave. A figure showing three phases with a carrier signal can be seen in Figure 2.12. The I21 inverter is a three-phase inverter, thus the three phase signals shown in the figure gets sent to each of the three half-bridges in the VSI, see Figure 2.2. [7]

Each of the four inverters in the car has its own carrier signal which is compared to the three phases, and the two R5-cores are responsible for updating the duty cycle in the software on each of the two inverters that they are in charge of, either in the front or in the rear.

Space Vector Modulation (SVM/SVPWM)

The *Space Vector Modulation* (SVM), or *Space Vector Pulse Width Modulation* (SVPWM), is another modulation technique that is the last step of the FOC algorithm. It generates the three phase voltages which are sent into the VSI. Voltages in the dq-space that have been through PI-controllers are sent through an inverse Park transform that converts them into the $\alpha\beta$-space, see Section 2.2. The resulting voltages are input to the space vector modulation, which outputs duty cycles. [17]

A space vector is made through binary inputs to the transistor in the three half bridges. The information says something about which transistor in each half bridge should let current through. It gives this information through binary numbers, and an example of an input of 100 is shown in Figure 2.13. The output is three phase signals. [19]

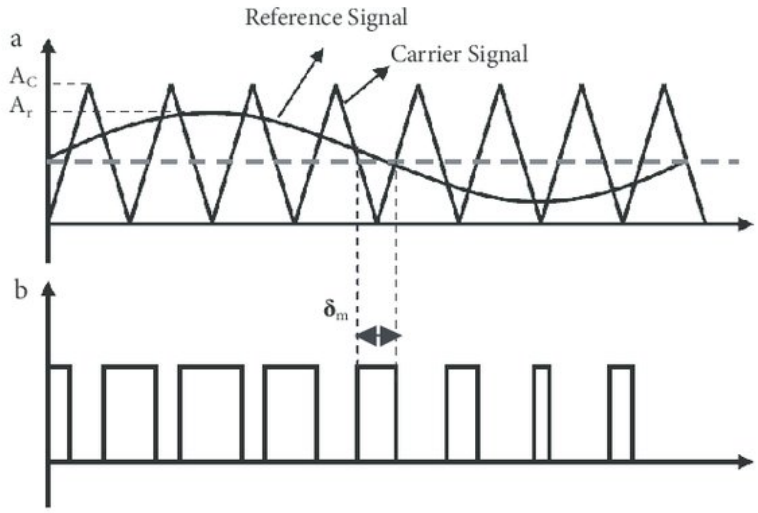


Figure 2.11: The upper graph shows a sinusoidal current (reference signal) and a high frequency clock signal (carrier signal). The lower graph shows the resulting square wave from a compare of the two aforementioned signals, and this output is sent to the transistors shown in Figure 2.2. The transistor that receives this signal will stay in "on"-mode and let current through when the duty signal is high, shown with δ_m. Figure inspired from [15].

Furthermore, $2^3 = 8$ configurations can be made of three half bridges. All of the eight configurations can be seen in Figure 9.7, where each configuration of transistors represents one space vector in the motor. Six *basic* vectors can be made, and two null vectors can also be made. [19]

To create more than six vectors around the sphere, and thus making the vectors take a continuous revolution, the null vectors are used. By switching between two basic vectors inside one of the six sections in the sphere, the angle of the space vector can be changed, and therefore an approximation of a space vector can be made all throughout the sphere. Also, by including null vectors in the switching sequences, the magnitude of the resulting space vector can be changed. [19]

Since SVPWM uses a weighted average of the three nearest vectors to generate the wanted voltage, so the resulting modulation waveform has double hump characteristics, as can be seen in Figure 2.15.

SVPWM can be achieved with a sector finding algorithm. The algorithm utilizes the relationship between the values of u_d and u_q in the stator and which sector the vector is in. A diagram showing the algorithm can be seen in Figure 9.8, where the sectors are defined in Figure 2.14.

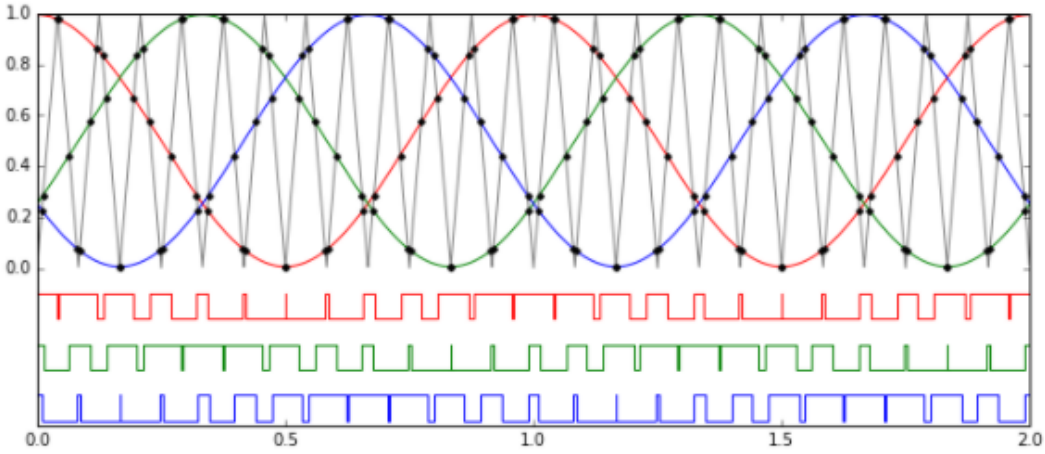


Figure 2.12: The three phase currents shown in sinusoidal and in their respective square wave form. The grey sawtooth signal with the same amplitude as the sinusoidal signals is a high frequency clock signal, and is called the carrier wave. Figure taken from [16].

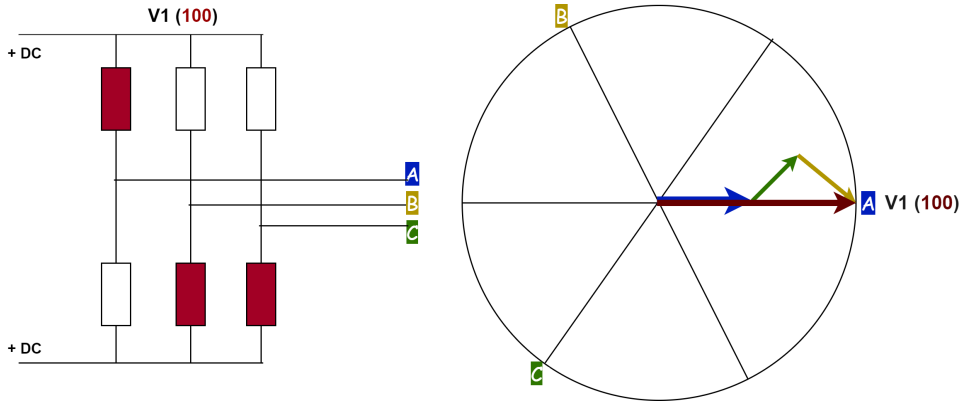


Figure 2.13: An example of one configuration of three half-bridges. In each of the half bridge one transistor can let current through. Here, one transistor connected to + DC and two transistors connected to -DC, shown in red, are on and letting current through. Each half bridge sends one current through, represented with A, B and C. The sum of the three vectors that each of the currents provides are summed up to one space vector, represented here by the red vector. Figure inspired from [18].

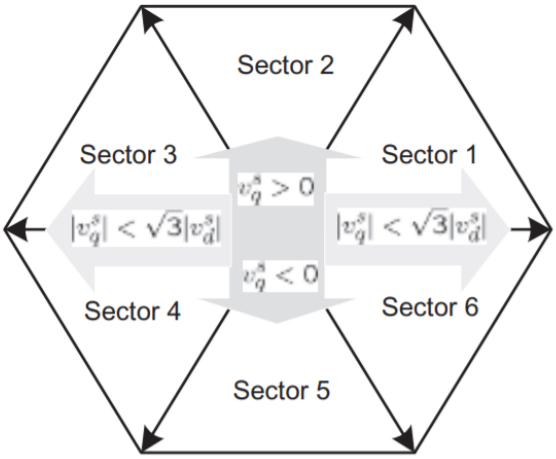


Figure 2.14: Image representing the sectors in the sixtant and which sector each condition points to. Image is inspired from chapter 10.2.4 in [20].

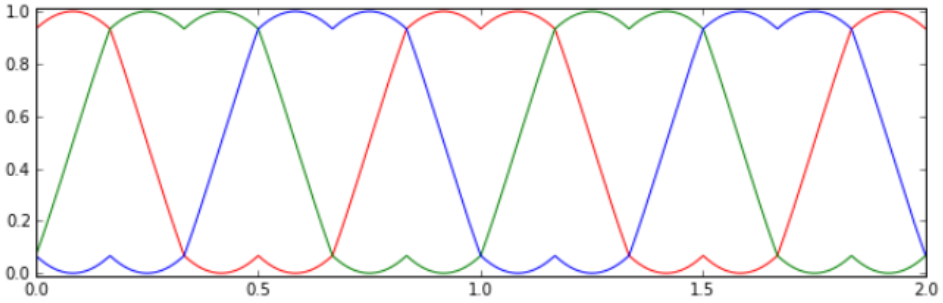


Figure 2.15: Figure shows the double hump characteristics of a space vector pulse width modulated signal. Figure inspired from [21]

Difference between SPWM and SVPWM

The difference between SPWM and SVPWM is that SVPWM's modulation waveforms have double hump charasteristics, see Figure 2.15. With this, it utilizes more of the source voltage than SPWM that just use sinusoidal modulation waveforms. It can be shown that the line-to-line voltages of the two methods can be approximated as

$$V_{\text{line-to-line}} = \begin{cases} \frac{V_{\text{DC}}}{\sqrt{2}} \approx 70.7\ \%\ V_{\text{DC}}, & \text{Space Vector PWM} \\ \frac{\sqrt{3}V_{\text{DC}}}{2\sqrt{2}} \approx 61.2\ \%\ V_{\text{DC}}, & \text{Sinusoidal PWM} \end{cases}, \tag{11}$$

derived in ([20], p. 361).

This can also be shown with the use of the vector configurations defined in Section 2.9.4, see Figure 2.16. The outer hexagon represented by the six-step commutation of the six basic vectors will have its peaks at $\frac{2V_{DC}}{3}$. SPWM will have its peak value at $\frac{V_{DC}}{2}$, while SVPWM will have its peak value at $\frac{V_{DC}}{\sqrt{3}}$.

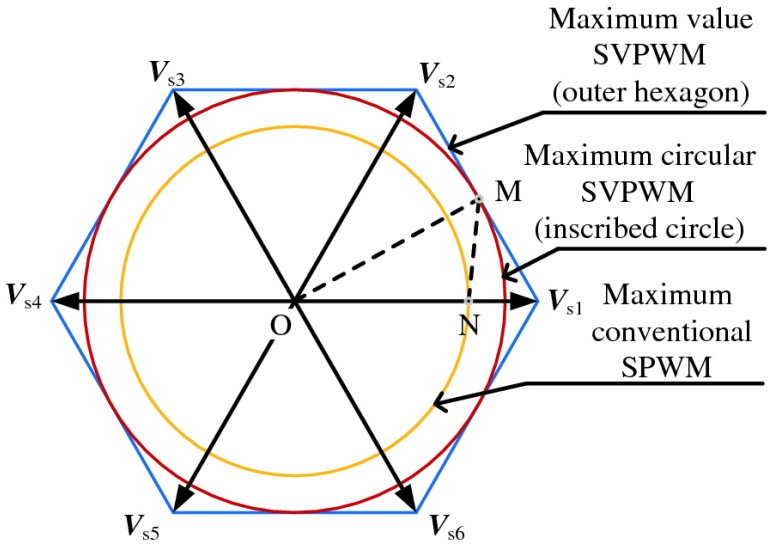


Figure 2.16: Image representing the peak voltages with radii that SVPWM and SPWM will stay inside of. The peak voltage achievable with SPWM is represented with a radius of $\vec{ON}$, while SVPWM will have its peak value with a radius of $\vec{OM}$ Figure inspired from fig. 1.1 in [22].

Injection of third order harmonics

To utilize more of the source voltage in SPWM, another wave can be injected to each SPWM phase voltage, which will increase the peak voltage without suffering added oscillation. This can be done with third harmonic waves, which are waves with three times the fundamental frequency of the original SPWM signal, see Figure 2.17. This method is called *Third Harmonic Sinusoidal Pulse Width Modulation* (THISPWM). [19]

The peak size of the modulated sum will be lower than the peak fundamental output voltage. So, the peak of each fundamental phase voltage will have a peak of U, whereas the sum of that phase voltage with a third harmonic wave will have a peak of $0.87\,U$. Thus by using third harmonic injection (THI), a higher peak can be reached while keeping the same line-to-line voltages. This can happen because the harmonics cancel themselves out from common-mode voltages between each of the phases, thus the receiving windings will only see a pure sinusoidal signal. [19]

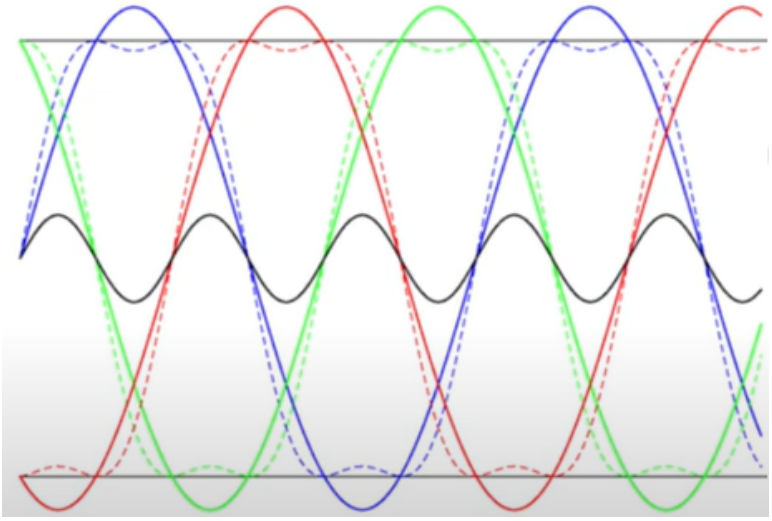


Figure 2.17: Graph showing the three phase voltages with the same fundamental frequencies and phase shifts, with a third harmonic wave corresponding to each of the three phases in black. The resulting modulated signal with the third harmonic injection is shown in dotted lines for all three phases. Image inspired from [19].

2.9.5 Feed forward control

From the voltage equations in Equation (6), one can observe that the voltage v_d is dependent on ψ_q, and v_q is dependent on ψ_d. Furthermore, from Equation (7), one can see that ψ_q is proportionate with i_q, and ψ_d is proportionate with i_d. This makes v_d is dependent on i_q, and v_q dependent on i_d. These terms are therefore called cross-coupling terms, and this dependency can be *decoupled* using feed forward control. [7]

3 Literature Review

Several technologies within motor control are interesting to look into. The first two chapters are a literature review on the previous master theses that have worked with the motor control of Revolve NTNU's current inverter. The third chapter presents theses that have been done on sensorless control of the motor, and then the fourth chapter presents work done on encoder calibration. Finally, the last chapter, Section 3.6, discusses the presented literature, and a choice is made on which design that will be worked on in this thesis.

3.1 Motor control in R19

In 2019, Revolve alumni Håkon Skeie wrote his master thesis on motor control of I19, the older generation inverter [23]. His design consisted of FOC with the use of Heidenhain's motor encoder, with added field weakening control for the reference torque, see Figure 3.1.

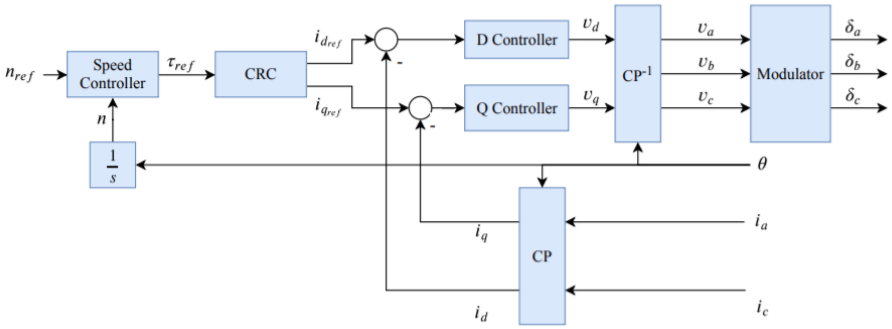


Figure 3.1: A simplified block diagram of the motor control that Skeie implemented, with reference inputs n_{ref} coming from the VCU, θ coming from the motor encoder and i_a and i_c is measured from the power stage (VSI) cards. [23]

This design is based on NTNU professor Roy Nilsen's creation of a motor control of a PMSM in the class TET4120 *Electric Drives* [7].

Skeie's solution implemented field weakening control on the torque to find the current i_d, with the use of both *Maximum Torque Per Ampere* (MTPA) and *Maximum Torque Per Flux* (MTPF). MTPA was run with the Equation (10) with the torque τ as input, and a MIN-block in Simulink was used to switch between this and MTPF, based on the resulting i_d reference current, see Figure 3.2.

To run the MTPF, a Newton-Raphson method was run to find an approximation of the stator current angle, which was used to find the optimal current i_d. This was done with the use of the equations

$$\begin{aligned} 0 &= f(\epsilon, \psi_s) - \tau_{eref} \\ x_{n+1} &= x_n + \frac{f(x_n)}{f'(x_n)} \end{aligned} \quad , \tag{12}$$

where the first equation defines the algorithm used to converge to get the stator angle ϵ, and the second equation is the definition of the Newtons method. The function $f(\epsilon, \psi_s)$ and its derived form $f'(\epsilon, \psi_s)$ are defined as

$$\begin{aligned} f(\epsilon, \psi_s) &= (\sin(\epsilon) - (1 - \frac{x_d}{x_q})\frac{\psi_s}{2\psi_m}\sin(2\epsilon))\frac{\psi_s \psi_m}{x_d} - \tau_e \\ f'(\epsilon, \psi_s) &= (\cos(\epsilon) - (1 - \frac{x_d}{x_q})\frac{\psi_s}{\psi_m}\cos(2\epsilon))\frac{\psi_s \psi_m}{x_d} \end{aligned} \quad . \tag{13}$$

Furthermore, in the "Modulator"-block, he made an SPWM that is run together with added third harmonic injection.

The work was made for a SAM E70 ARM Cortex M7 *Microcontroller Unit* (MCU). First, the work was conceptualized in Simulink, only to be used for simulation of the motor, and then after that it was implemented and written by hand in C code. The Simulink file was partly lost after his thesis was delivered, thus what is left is an earlier revision.

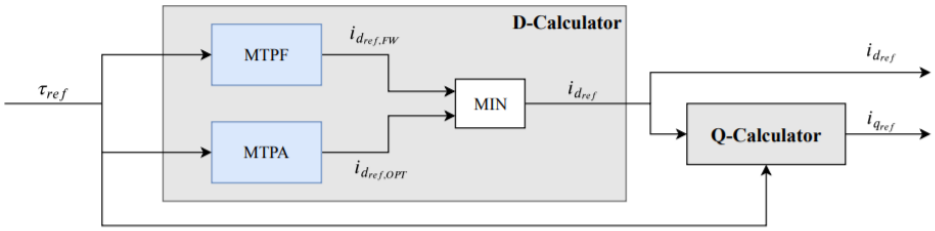


Figure 3.2: A simplified block diagram of the CRC, or Current Reference Calculator, where the field weakening control happens [23].

3.2 Motor control in R20

The year after Håkon Skeie did his master on motor control, Francesco Fanin did both his project- and thesis on the same control system [24], [25]. His project thesis both researched and implemented improvements of the modulation of the motor control, and his master thesis researched areas of improvement for the field weakening in the motor control.

3.2.1 Modulation improvements

Fanin's project thesis implemented *Generalized Discontinous Pulse Width Modulation* (GDPWM) on Skeie's motor control. GDPWM works by using different modulation techniques based on the modulation index. The modulation index M is defined as the ratio between the line-to-neutral voltage v_{lm} and the six-step voltage V_{DC},

$$M = \frac{V_{\text{lm}}}{\frac{2V_{\text{DC}}}{\pi}}. \tag{14}$$

The GDPWM Fanin implemented uses Space Vector PWM (SVPWM) in the low modulation range, where $M < 0.65$, and switches to *Discontinous Pulse Width Modulation* (DPWM) when $M \geq 0.65$. More specifically, when $0.65 \leq M \leq 0.81$ DPWM2 is used, and finally when $M > 0.81$ DPWM1 is used.

DPWM happens when a discontinuous zero-sequence wave is injected into the three-phase signals, as it is injected for third harmonics that was described in Section 2.9.4. For this, an inverter leg is clamped to either the positive or negative side of the DC bus for a maximum period of 120°. During the clamping, there can be no switching; thus this method decreases switching losses. DPWM1 happens when the modulated PWM signal with the highest magnitude defines the zero-sequence signal, and DPWM2 happens when all three modulated signals are phase-shifted with 30°, and the resulting highest modulated signal defines the zero-sequence signal.

GDPWM was shown to have good results for modulation, since SVPWM had great performance when modulation indices were low, and DPWM had great performance when modulation indices were high.

3.2.2 Field weakening improvements

Fanin wrote his master thesis on different algorithms for generating current references in the field weakening region [25].

Furthermore, Fanin modified the Newton-Raphson model that Skeie implemented in his thesis. He calculated the stator angles beforehand and put them in a table together with the input torque for a quick and fast response. This angle was then used in the equations in the Newton Raphson method in Equation (12). Fanin concluded in his thesis that this modified scheme was the optimal algorithm for the use in a racecar, since it both alleviated oscillations and any issues with approximations.

He also looked into making custom LUT for the current components i_d and i_q to very quickly retrieve information on them for any given torque τ, thus modifying the current reference calculator that Skeie made, see Figure 3.3. This method was deemed a feasible alternative to running numerous iterations of the Newton-Raphson method to approximate the d-current from the torque. However, because of Covid-19 breaking out in March of 2020, he could not implement and test this method.

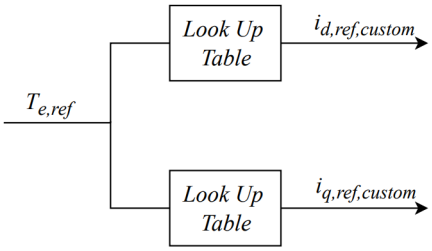


Figure 3.3: Fanin's modified current reference calculator that's based on Skeie's model, see Figure 3.2, but with an added LUT. [25]

3.3 Sensor-free control

There are multiple benefits with implementing a sensorless motor control on the car. As described in Section 2.9.3, the benefits include not needing a motor encoder connected to each of the four motors on the car, which can save a lot of space and weight. This chapter will look into an implementation of an improved DTC, defined in Section 2.9.3, and how it compares to FOC, defined in Section 2.9.1.

3.3.1 Implementation of Direct Torque Control (DTC)

A paper was done in the university of Tianjin University in China in 2017, where they implemented a direct torque control algorithm for a *Permanent Magnet Synchronous Motor* (PMSM). The work was done on a three-level inverter instead of a two-level one; however, the basic implementation remains similar. [26]

The implementation focused on reducing steady-state torque ripples and maintaining a smooth vector switching. This was done by creating a torque/flux LUT with multiple voltage vectors and thus a precise control. The LUT found which voltage vectors matched with the flux and torque and had corresponding duty cycles that can be outputted to the VSI. [26].

The implementation from the paper resulted in the instantaneous torque response that the standard DTC already has, in addition to some torque ripple reduction in steady states. This was done with a three-part algorithm, where only the first two are relevant to a two-level inverter:

1. Make a LUT that extends the standard switching table that's presented in Section 2.9.3. The table consists of torque and flux change rates caused by the voltage vectors. It can be extended with multiple voltage vectors and precise control levels. In the paper, the three-level inverter had 18 basic voltage vectors and $18(N_d - 1)$ number of virtual voltage vectors with fixed direction. N_d is the number of pieces that d_n is divided into, between 0 and 1. d_n is the duty cycle of the basic vector V_n. The LUT is universal for the DTC of different power levels. [26]

2. The vectors in the LUT are selected based on a cost function that is defined in the paper. The cost function calculates the effect that the voltage vectors have on the torque and flux, and the vectors that minimize this cost function are selected. [26]

Characteristics	**Field Oriented Control**	**Direct Torque Control**
Dynamic torque response	Slower	Faster
Static torque, current and flux response	Lower ripple and distortion	Higher ripple and distortion
Parameter sensitivity	Decoupling depends on stator inductances L_d^s and L_q^s, and the rotor flux ψ^r	Sensitive to stator resistance R^s
Controllers	PI-regulator for current control	Hysteresis controllers for torque and stator flux ψ^s
Switching frequency	Constant	Variable
Complexity of implementation	High complexity, need Clarke/ Park-transform	Less complexity than FOC, no need for any coordinate transformation

Table 3: Comparison between the control methods DTC and FOC in a PMSM, done in the thesis [27].

3.3.2 Comparison of Direct Torque Control (DTC) and Field Oriented Control (FOC)

In 2018, a survey published by the University of Munich summarized a comparison of DTC and FOC based on three other theses. [27] Even though there are many parameters to take into consideration when comparing the two methods, and a superior method can't be determined, a generalised comparison was made. Overall, the paper concluded that FOC has good steady-state characteristics, whereas DTC shows better dynamic performance. This gives a faster torque response that is more desirable for the running of the Revolve NTNU's racecar. More general comparisons are shown in Table 3. [27]

3.4 Effects of motor position sensor calibration

In a thesis done in 2019 at Chalmers University of Technology in Sweden, an evaluation was done on the effects of sensor measurement errors on a motor. [28] The calibration looks at the relationship between the angles between the stators' and rotors' magnetic fields. The work was done on a *Dedicated Hybrid Transmission* (DHT) system, which consisted of one *Internal Combustion Engine* (ICE) and two PMSMs. The position sensor used was a resolver, which is a bit different from an encoder. It consists of a rotor with a reference winding and a stator with two windings that are 90 degrees apart. The rotor in the resolver needs AC power to run, which again induces a voltage in the stator. The shaft position can be found from this induced voltage and the reference voltage.

However, the calibration errors are analogous to those occurring from using an encoder. Through simulations, a faulty calibration of the position sensor was shown to be related with torque ripples. More specifically, torque ripples in the range ± 5 Nm were observed with current offsets larger than 3.5 A. The report concluded that there were significant power losses with just one mechanical degree offset. This shows how vital the calibration of the motor position sensors is.

Table 4 shows the results from the simulation of the motor with imposed angle offset errors. Resistive power loss increases exponentially with the increase in offset error. This quickly leads to critical faults with the running of the motor. [28]

Offset error (mech. deg.)	**Increased resistive power loss (Watt)**	**Percentally increased power loss (%)**
0.25	2	0.5 %
0.5	6	1.5 %
1	22	5.5 %
2	101	25.5 %
3	309	78 %

Table 4: The increased resistive power loss in the machine due to angle measurement offset errors. [28]

3.5 Autocalibration of motor position

In a thesis [29] done on actuators for dynamic robots done by Benjamin G. Katz in 2016, a position sensor calibration procedure was made that aligned the zero-position of the sensor with the rotors d-axis of the rotor in the motors used in the project. In the thesis, the brushless electric motors ran electric actuators, and the position sensors used were hall-effect, absolute digital encoders. [29] The procedure to find the position sensor offset went as follows:

- A current is imposed so that the magnetic field from the stator is rotated with little velocity for a full rotation, both for a full clockwise and a counter-clockwise rotation.
- The rotor will move along with this stator voltage vector for a full rotation, and this position of the rotor is then registered.
- During each rotation, the rotor magnetic field along the d-axis registers the stator field along the α-axis. These are proportional axes, as can be seen in the Park transform Equation (3).

- After letting the motor cycle for multiple rotations, the offset was then found from the average difference inf angles between the Park transformed voltages; the rotor voltage vector along the d-axis, and the stator voltage vector along the α-axis.
- Since motors have some degree of friction and experience cogging, the position of the rotor was non-linear around the physical cogging points of the rotor, which is symmetric for each 360° cycle. By just recording the offset at any point during the rotation of the motor, this could include the variations from the constant tracking error. To remove this continuous non-linearity, the offset was recorded using a moving average FIR filter to compensate for the torque cogging. [29]

This procedure gave efficient results of the offset. This was shown by intentionally offsetting the position sensor error with 0.75mm off the d-axis. Without running this procedure and imposing an offset on the position sensor, this caused severe errors in positions that made the rotor turn in the opposite direction of where it was urged towards. After a successful calibration of the motor using this procedure, the torque ripples were not measurable in comparison with the torque from the cogging of the motor [29].

3.6 Summary

The implementation of a stable and accurate DTC will have many positive effects on the running of the motor. Like reduced cost and weight on the car, less complexity wrt. the encoder interface currently used on the inverter, and less need for maintenance and accurate calibration. However, a choice was made to continue with the work that Skeie and Fanin have done already concerning FOC, since there are still a lot of improvements to be done on this side that can provide great results on the motor. In addition, going from having a sensored to a sensor-free control in just one semester is a very ambitious goal, and would require large amounts of work and risk.

Because of the already successful improvements on the modulation of the motor controller done by Fanin, see Section 3.2.1, the author wanted to prioritize looking into improvements in other aspects of the motor control algorithm.

The nominal speed of the Fischer motor Revolve NTNU uses is 13250 rpm at 600V DC, see Section 2.4. At this stage, field weakening is introduced to reach peak torque at 20000 rpm to counteract the BEMF that keep the motor from reaching peak torque. The field weakening control improvements that Fanin researched on his master, mentioned in Section 3.2.2, is an interesting design to implement. A custom-made LUT with torque and current measurements will be a faster implementation than running through at maximum $N = 100$ iterations of the Newton Raphson method, as it is done in the current model.

A faster run-through of the motor controller results in possibilities for smaller switching periods of the control loop, or more time spent in the control loop doing sensor measurements or other procedures and checks. A design of a motor controller with a LUT will therefore be implemented in Simulink as part of the work for this thesis.

A choice was made to pivot towards making improvements on the calibration of the encoder. This is because calibration is a critical part of the running of the motor. As seen in Section 3.4, if the alignment between the rotor position and the encoder is inadequate, this could lead to drastically significant errors when driving the car, such as noise in the motor and power dissipation. A safe and reliable operation of the car is key, thus, having accurate encoder data is essential. In addition, having predictable and constant currents out to the motor is essential for motor control, and this will be achieved with high accuracy in the encoder alignment.

The improvements on the calibration will be based on the procedure used in Section 3.5. Even though this procedure was not done on an IPMSM, it will be comparable to this system since it uses absolute encoders and will be useful because of the solutions the procedure offers against torque cogging effects.

4 Design

The chosen design to work on is the already existing Field Oriented Control (FOC) made by Skeie, presented in Section 3.1, and the specifications of the design will be introduced in Section 4.1. The chosen improvements that were be done on this motor control will be presented in Section 4.2 and Section 4.4.

4.1 Specifications

For this master thesis, it is desired to make improvements to the code quality of the motor control algorithm that is run in the I21 today. This includes improvements on the code modularity and structure, as well as its runtime. As elaborated on in Section 1.1, the current code base with all of the mathematic formulas is not easy to read, as it is solely human-written code. A lot of time was spent on interpreting the code, and it is desired to instead have the complicated motor control algorithm in Simulink, which is made for mathematical applications. This will give a better overview of the model and an easier understanding of the algorithm. The runtime of the code can also be improved by code optimization objectives, which can be chosen in the configuration in Simulink.

In addition, the current way of running the current controller in the motor control algorithm is slow, because of the multiple iterations required to run the Newton-Raphson method that approximates the d-current. Thus, a design and an implementation of a motor controller with a current and torque LUT was seen to improve upon the time needed to run through the current controller.

Furthermore, the old way of calibrating the motor encoder is inefficient due to it being done manually each time it's done. This takes multiple hours to do, even for those experienced with the process. Furthermore, as described in Section 3.4, if the calibration isn't done correctly, this could require much more current to reach the desired rotational speed of the motor. Thus, a finely tuned code algorithm that could automatically calibrate the encoder and leave out the human aspect of the process would significantly improve the encoder calibration.

All of this would be improved by the following functional and technical specifications, followed by the concrete acceptance requirements that the system will need to validate to successfully uphold the specifications. The functional specifications include:

F1 Make improvements on the calibration of the motor encoder

- **F1.1** Reduce the time taken to calibrate the encoder.
- **F1.2** Reduce the possibility for human failure and human inaccuracy when calibrating the encoder by having an automated solution.
- **F1.3** Reduce the i_q-current necessary to achieve 20 krpm with a more accurate calibration through an autocalibrated system.

F2 Make improvements on the code base of the motor control algorithm.

- **F2.1** Switch out parts of the current structure of the code base with a modular solution.
- **F2.2** Add a code wrapper that interfaces between the old structure and the new modular structure.
- **F2.3** Reduce the runtime of the motor control algorithm.
- **F2.4** Design and implement a more efficient version of the motor control algorithm.

The technical specifications of the design that are built upon the functional specifications are as follows:

T1 Regarding **F1.1** and **F1.2**, the current solution of manually calibrating the motor encoder will be replaced with an auto calibrated solution.

T2 Regarding **F1.3**, by getting a more accurate calibration, less q-current is needed to correct the alignment of the α- and β-axes with the d- and q-axes.

T3 Regarding **F2.1**, and **F2.3** the current solution will be switched out with a modular code generated Simulink model of the controller, which will be sent through MATLAB's code generation. When doing this, the code will be optimized with respect to execution efficiency. This is because the Embedded Coder that Simulink uses to generate C code, has optimization strategies to increase the execution speed of different parts of the code, like shown in [30] and [31].

T4 Regarding **F2.4**, a motor controller model in Simulink will be made, where the current controller is switched out with one using a Look-Up Table (LUT).

The acceptence requirements that the design should uphold are as follows:

A1 After calibration, the torque response should stabilize between -2 Nm and 2 Nm when the motor is running at 20 krpm.

A2 After implementation of the code generated Simulink model of the motor controller, it should have an equal or faster runtime than what the current solution on the I21 code has now.

4.2 Design of the motor control model

In this chapter, the original design of the motor simulator will first be introduced. This simulator contains the motor controller that is used as inspiration for the motor controller made in this design. Afterwards, the new design of the motor control model based on the simulator will be introduced.

4.2.1 Motor simulator

The highest level of the Skeie's self-made Simulink model is as shown in Figure 4.1. This is the model used for simulations of the motor, including the mechanical system itself. The mechanical system was made based on measurements on how the motor acted in Revolve NTNU's previous seasons. This was recorded in the organizations very own self-made analyzation software called Revolve Analyze. Then, the motor load was approximated and modeled as damping from vehicle drag [23]

The actual motor control is in the subsystem block called "PM Motor Control for 3 phase machine", and its content is shown in Figure 9.1. This is the part that would be separated from the simulator, code generated, and merged with the I21 code. To do this, some changes would have to be made to the way it is built up, but first, an overlook of how it currently is in the simulator will be presented.

In the simulator, the motor control outputs the duty cycle *d1*, and inputs the following shown in Table 5. The motor control in Figure 9.1 contains five major subsystems, each of which works with different parts of the motor control algorithm. These subsystems are shown in Section 9.

- The **"Speed Controller"** subsystem takes the reference and measured rotational speed of the motor, n, and runs a PI-controller on the error between them, and finally outputs the torque from it.
- The **"CRC"**, or Current Reference Calculator, inputs the torque and calculates the first two different i_d-references, using MTPF and MTPA algorithms, as explained in Section 3.1. The smallest of the two is chosen and used for calculating the i_q current. This subsystem is shown in Figure 9.3.

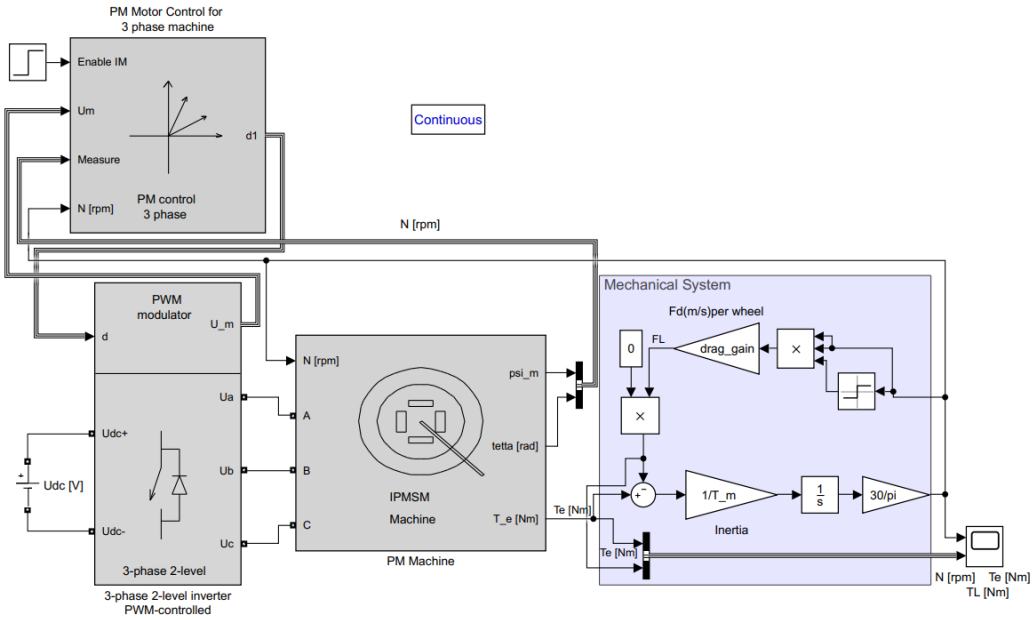


Figure 4.1: The model made for running a three phase motor, seen here from the top level. Figure taken from [23].

- The **"pwm modulator1"** is the modulator that runs the SPWM and outputs the final duty cycles that go out to the gate drivers. This subsystem uses an S-Function block, which is a code block that gives a definition as to what tasks to perform during different parts of the simulation. [32] This function creates a third harmonic pulse width modulation that takes dead times and other losses into account for a more realistic simulation.

- The **"Decoupling IM motor"**-subsystem is responsible for decoupling some of the multiplexed input vectors, and it then calculates the inputs for the above-mentioned control subsystems. This subsystem is shown in Figure 9.4. It contains a "Decoupling" subsystem shown in Figure 9.5, a "Current Measurement" subsystem, which is shown in Figure 9.6, and a "Voltage Measurement" which is almost identical to the "Current Measurement"-subsystem, for voltages instead of currents.

- The **"digital current-controller1"**-subsystem contains the PI-regulators that inputs the errors between both reference and measured currents i_d and i_q, and outputs the voltage u_d and u_q that is used to find the modulation index and the phase angle in the stator space that is needed in the **"pwm modulator1"**-subsystem block. This subsystem is shown in Figure 9.2.

Inputs	Description
Enable IM	The signal that runs the initial step-function of the model.
Um	A multiplexed vector which contains the U_{DC}, the three phase voltages U_a, U_b and U_c, and the three currents I_a, I_b and I_c.
N [rpm]	The speed in rpm, which in this model is simulated, but in the car it is fed back to the inverter from the Vehicle Control Unit (VCU). The VCU takes sensor inputs from the car and runs a torque vectoring algorithm that finds the optimal setpoints for the car, such as the rotational speed.
Measure	A multiplexed vector which contains the inductance ψ_m and the encoder angle offset

Table 5: An overview over the different inputs on the motor controller, shown in Figure 9.1.

4.2.2 Design of new motor controller

The design of the motor control model can be separated into three parts. First, the structural part of the design, which consists of the changes in the Simulink code base and block diagram. Further, the design of the way the motor control model is organized in repositories will be presented. Lastly, the design of how the new motor control model will be to be ported into the already existing code in the inverter will be presented.

Code base

The full original simulator made by Skeie, shown in Figure 4.1 will be left unchanged and kept 'as is for further use. The only change done here is the tuning of the PI-controllers in the current controller.

The simulator made by Skeie was made with an ATSAM MCU in mind, whereas the I21 has a Zynq Ultrascale+ MPSoC. Thus, some parts of the algorithm need to be ported to the new MPSoC. The speed and torque controllers are, however independent from specific features in the ATSAM.

The new design will, as mentioned, consist of only a motor controller, where a copy of Figure 9.1 will be used as a baseline for this. The changes that need to be made on Figure 9.1 include making it work for fixed time steps, as the inverter runs on a fixed 20 kHz. The simulator uses a variable time-step in some parts of the controller, including most of the "pwm modulator1"-subsystem block. It is therefore desired not to include this subsystem block in the new motor controller. This means that the output of the motor controller will be outputs from the "digital current-controller"-subsystem.

Organizing of source code

In addition to changes in the block diagram, organizational changes will be done to the project repositories. The desired structure is shown in Figure 4.2.

The repository where all the C code of the I21 inverter is, described in Section 2.8, is called `MPSoC_Inverter`.

It is not desired to have all of the .slx- and .mat-files, including the temporary project files generated by MATLAB, together in the same repository as `MPSoC_Inverter`, which only consists of C-code. Doing this will be unorganized, and it will be hard to keep track of the different systems and models. Therefore, a new repository was made called `Motor_Control_Simulink`. This is the repository where the Simulink- and MATLAB files of the simulator and the motor controller should be. The repository contains all the .mat- and .slx-files needed to run both the models. This is also where the code generation is run and where the .c-files are made from the Simulink project.

The .c-files that are code generated from `Motor_Control_Simulink` will be used and run on the inverter through the `MPSoC_Inverter` repository. One could just copy these files from one repository to the other. However, when the motor controller model in Simulink is updated in the future, version control will be difficult to uphold.

Therefore, to maintain structural integrity, a third repository will be made called `Inverter_Motor_Controller`, which will inherit only the code generated .c-files from `Motor_Control_Simulink`. This repository will be used as a submodule in the host repository `MPSoC_Inverter`, which will make it simple and user-friendly to always have the latest version of the motor controller in the main inverter repository.

Code generation from Simulink to running code

To run the Simulink model on the Zynq Ultrascale+ MPSoC, some code interfaces between the Simulink motor controller model and the microprocessor have to be made. First of all, the Simulink model needs to be code generated.

By first running the model through MATLAB's Embedded Coder [30], the resulting function will be inserted in the code base, and appropriate measures have to be taken to merge this into the already existing main-file that runs the motor control that is flashed onto the inverter. This includes making a code wrapper.

Code generation from Simulink is compiled with respect to execution efficiency and will not be high in readability. This wrapper should therefore be an interface between the user and the code generated .c-files.

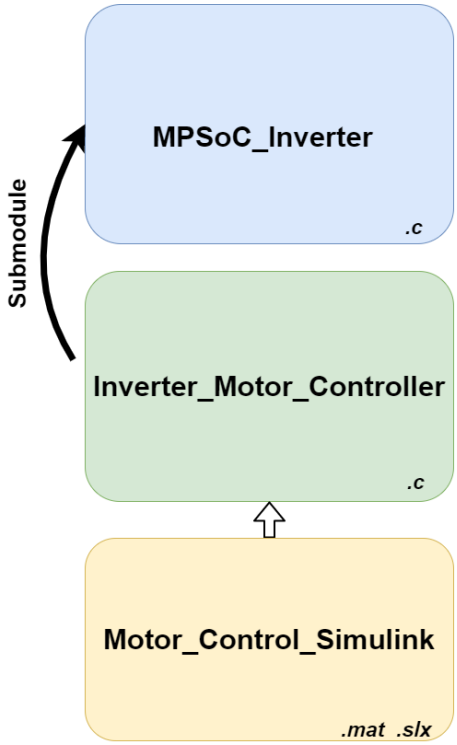


Figure 4.2: Overview of the repository structure and inheritance of the motor control for the inverter software.

The wrapper will contain two structs, called `motor_controller_input_t` and `motor_controller_output_t`. The former would contain all input variables that correlates to those from the code generated .c-files, and the latter would contain all correlating output variables.

In addition, the wrapper will contain set- and get-functions which will set the inputs and get the outputs of the code generated .c-files. These functions will use pass by reference, which means that they would take in pointers as input and assign the value directly to the variable. An alternative to this would be to use pass by value, which would temporarily store a copy of the variable every time the function is called. An additional read- and write operation would be required to set or get a variable, which costs overhead.

There would only be one set function that will set all the inputs of the model, and in addition, only one get function. This would lead to less overhead in comparison to having multiple set- and get-functions for each of the individual inputs and outputs for the model since the processor then has to jump to a specific point in the memory each time the function is called.

4.3 Design of a motor controller using a Look-Up Table (LUT)

By switching out the Newton-Raphson method to calculate the currents i_d and i_q with a LUT, a faster run-through of the motor control will be achieved. This will allow for an increased switching speed and for reduced torque rippling.

A LUT can be made by using a sensor that measures torque for each setpoint of i_d and i_q. The more values in the LUT, the more accurate the torque response will be. However, then it will take more computational space. The LUT can be used to find the optimal current reference in relation to the torque input, as it is done in MTPA, and to find the optimal i_q current reference for the field weakening.

The improvements could be made as such; the voltage threshold for entering the field weakening region could be found using the relationship:

$$V_{\text{DC}} = \sqrt{|\ V_d\ |^2 + |\ V_q\ |^2} \tag{15}$$

The input V_{DC} is calculated from the reference v_d and v_q as in Equation (15), from the current controllers. This reference then goes into an error with the measured 600 VDC from the battery package. This error is sent through an A/V PI controller that inputs the voltage and outputs the i_d current. This reference current can be fed into a 3D LUT along with the reference torque τ_{ref}, which approximates the input values and finds the i_q-current that matches up with the approximations. This LUT is found from measurements on the motor and is generated using a script to find the optimal torque reference values. The two currents i_d and i_q can then go into the same current controllers which already exist in the model; see the "D Controller" and "Q controller" in Figure 3.1.

4.4 Design of an improved calibration procedure

To calibrate the encoder, the stator offset angle is needed. This has normally been found manually in the past in Revolve NTNU. This design is described in Section 4.4.1. A design that will improve upon this procedure is a more automatic way of finding the offset. This is described in Section 4.4.2.

4.4.1 Design of a manual calibration procedure

There are two ways to calibrate the encoder manually. One way is to apply a DC current from one of the phase currents to the other two. This current imposes a stator field on the rotor in the direction of the a-axis, which then aligns the d- and a-axes. See Section 2.2 for a definition and relationship of the (a, b, c)- and the (d, q)-system.

Another method is to take an initial guess at the stator offset angle and apply an i_d-current until the current no longer has an effect on the torque or acceleration of the motor. This method is a bit more tedious to implement and requires feedback about the torque through a torque sensor that was not acquirable by the author during the writing of this thesis.

4.4.2 Design of an automatic calibration procedure

The automatic design of the calibration procedure of the encoder is based on Benjamin G. Katz algorithm from his master thesis; see his algorithm described in more detail in Section 3.5.

A script will be written in the code base of the MPSoC that will spin the motor 360 degrees slowly back and forth to find the average stator offset angle. This will be added to the datum shift of the encoder, and then the motor shall be spun iteratively with a run-in procedure up to 20 krpm to validate the calibration.

5 Implementation

The implementation of the designs from Section 4 are described in this chapter. They have been done with the limitations mentioned from Section 1.2 in mind, and on the basis on work from previous years, like Håkon Skeie's motor simulator made in Simulink described in Section 3.1, and Francesco Fanin's research on field weakening improvement described in Section 3.2.

5.1 Implementation of the motor control model

The implementation of the motor control model was three fold. First, it consisted of the work done on the Simulink code base, then on the organizational structure of the repository of the code base, and finally on the total merge of the code base with the already existing C code that is running on the MPSoC.

5.1.1 Implementation of code base

First, the model was tuned to get the proper torque-, speed and current responses using trial and error, and previous knowledge on methods like the Ziegler-Nichols method. When the model was finished tuned, a MATLAB project was made that contained only the now finely tuned motor control subsystem, see Figure 9.1. This was called `motor_controller`, and its upper most level was structured like in Figure 5.1.

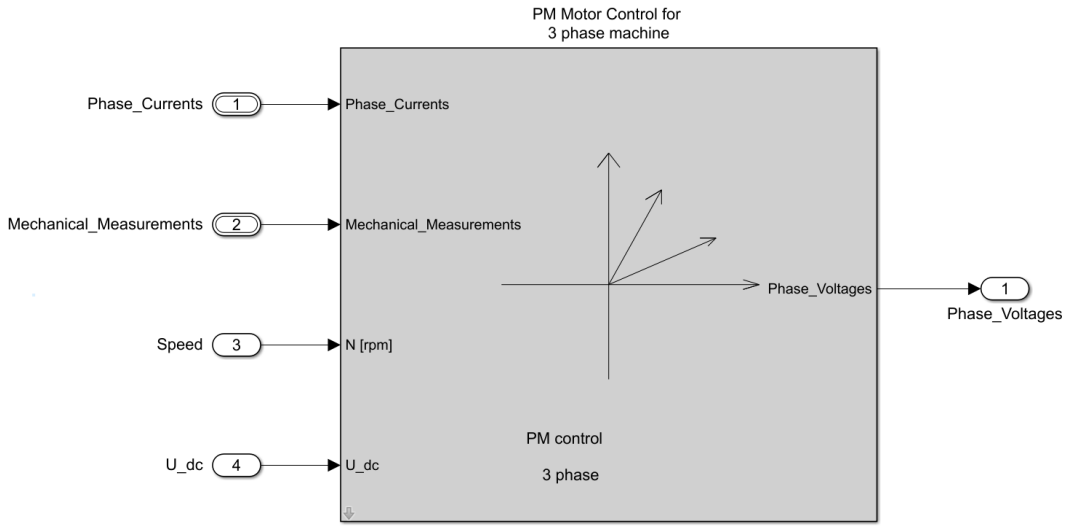


Figure 5.1: Simulink block diagram showing the upper most level of the motor controller model.

Inside the upper most subsystem shown in Figure 5.1, the motor controller was divided into four subsystems, shown in Figure 9.9. This is based on the motor controller from Figure 9.1, with the modulator removed. The speed controller just contains a PI-regulator that outputs the reference torque.

The "Decoupling IM Motor"-subsystem calculates the Clarke/Park-transformed current and voltages, which are necessary inputs to the other control blocks.

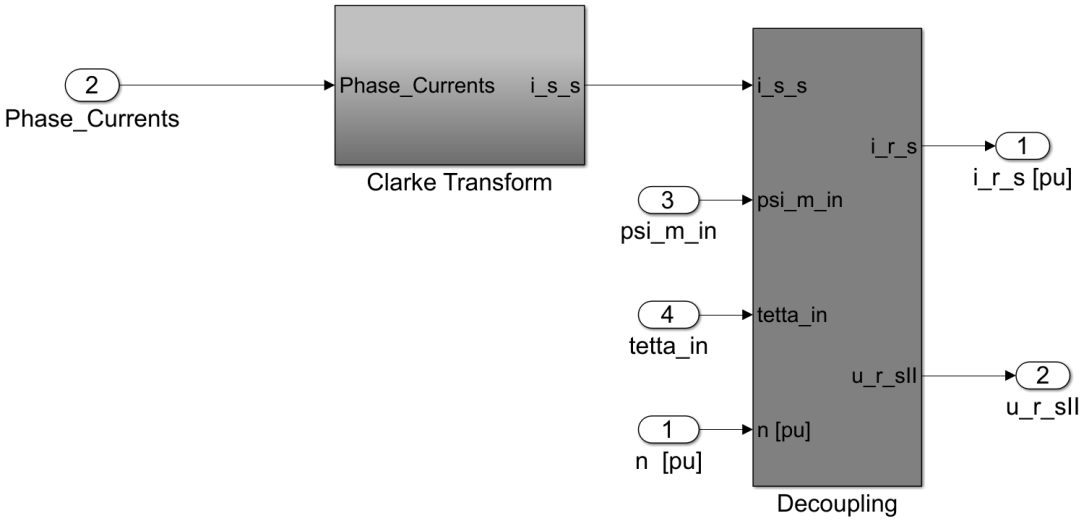


Figure 5.2: The contents of the "Decoupling IM motor"-subsystem from the motor controller shown in Figure 9.9.

"Decoupling IM motor" contains two more subsystems; the leftmost calculates the Clarke-transform of the currents. The rightmost, "Decoupling" shown in Figure 9.10, calculates the Park-transform of the currents, which is again used to find the voltages u_d and u_q.

The "Current controller"-subsystem was made into a subsystem reference, which means that it is also in use by the motor simulator. Both the motor controller and the simulator use this subsystem identically. The contents of this subsystem can be seen in Figure 9.11. The changes in this subsystem included running the phase voltages through an inverse Park transform to collect the u_alpha and u_beta before being outputted. Furthermore, the delay to theta was changed to being two switching periods, or $2 * f\_sw$ samples.

The "Is_calculator"-subsystem, seen in Figure 9.12, is where the field weakening control happened, and where the d- and q-current references from the torque requests were calculated. The currents were calculated as in Equation (10), where they were also saturated based on the systems limits. The d-current was calculated in two ways, by MTPV and MTPA, see Section 2.9.2 for a more detailed description of the two methods. The minimum of the two resulting d-currents was used further on as reference towards the "current controller" subsystem. The algorithm that runs the Newton-Raphson method is inside the subsystem called "Field-weakening EPSILON".

Changes that were done on the model included the change to fixed time step on the period `T_sw` based on the inverters frequency of `f_sw=20000`. Furthermore, all calculations to pu (per-unit) was done in the Simulink model.

Additonally, all scopes and unused signals going out to terminators was removed for the ease of structural overview. Instead of having scopes in the block diagram, the signals that needed to be logged could be viewed in Simulink's own Data Inspector.

More changes included changing the period `Tsamp` used in the calculations of the PI-controllers from `Tsamp=T_sw/2` to only `Tsamp=T_sw`, since the motor controller isn't run on both carrier high and carrier low, but only synchronously on carrier low on the MPSoC. The simulator runs it asynchronous on every extrema.

The model also didn't need to manage interrupts only on particular blocks the same way the simulator had, as the whole of the motor controller should run every 20.000 Hz. Interrupts in the MPSoC are generated from outside sources based on sensors on the car, which is taken care of by ISR's in the already existing C code on the chip. Thus, the interrupt inputs coming from the `Enable_IM` in the motor simulator, as shown in the input to the "PM Motor Control for 3 phase machine" in Figure 4.1, were not included in the motor controller.

5.1.2 Implementation of code organization

To interface between the repositry structures visualized in Figure 4.2, GitHub Actions was used. A figure showing the workflow of this is shown in Figure 5.3. All the actions done in relation to the `Motor_Control_Simulink` repository are shown inside of the yellow block, and the actions done in relation to the the `Inverter_Motor_Controller` repository are shown inside the green block. All actions that have to be done manually are red, and all actions that happen automatically from the Git workflow are blue.

This worked by making a .yml-file that dispatched a trigger every time a release was made in `Motor_Control_Simulink`, or whenever a commit was pushed to the branch `generate-code`; thus whenever the motor control model was updated. This .yml-file was made as shown in Listing 5.1.

The release had to contain a .zip-file called `generated_code.zip` containing all of the code generated .c and .h-files.

```yaml
name: Publish to Inverter_Motor_Controller

on:
  release:
    types: [published]
  push:
    branches: [generate-code]

jobs:
  publish:
    runs-on: ubuntu-latest

    steps:
      - name: Trigger Inverter_Motor_Controller
        run: |
          curl \
          -X POST \
          -u "User:${{secrets.USER_PAT}}" \
          -H "Accept: application/vnd.github.v3+json" \
          -H "Content-Type: application/json" \
          https://api.github.com/repos/RevolveNTNU/
    Inverter_Motor_Controller/dispatches \
          --data '{"event_type": "new_release"}'
```

Listing 5.1: Code from on_release.yml, which contains a Git workflow action in the repository `Motor_Control_Simulink`. The username has been switched out for this thesis.

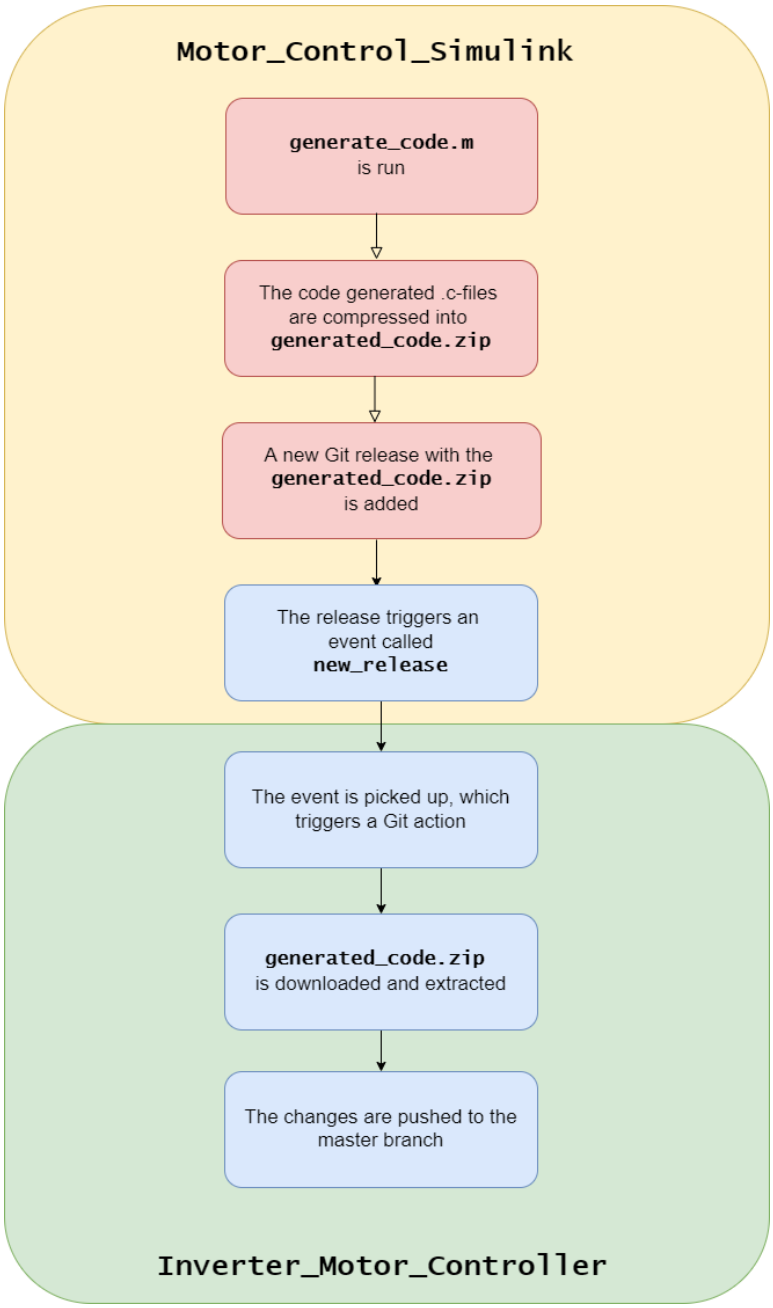


Figure 5.3: Figure showing the Git workflow between the repositories `Motor_Control_Simulink` and `Inverter_Motor_Controller`.

```yaml
name: Update to new release

on:
  repository_dispatch:
    types: [new_release]

jobs:
  update:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Install jq
      run: |
        sudo apt update
        sudo apt install jq
    - name: Find release
      run: |
        curl \
        -u User:${{secrets.USER_PAT}} \
        -H "Accept: application/vnd.github.v3+json" \
        https://api.github.com/repos/RevolveNTNU/Motor-Control-Simulink/releases/latest \
        >> asset_response.json
    - name: Download
      run: |
        wget \
        -q \
        --auth-no-challenge \
        --header='Accept:application/octet-stream' \
        https://${{secrets.USER_PAT}}:@api.github.com/repos/RevolveNTNU/Motor-Control-Simulink/releases/assets/$(jq '.assets[0].id' -r asset_response.json) \
        -O generated_code.zip
    - name: Remove old files
      run: |
        rm -r include/*
        rm -r src/*
```

Listing 5.2: Code from on_release.yml, part 1 of 2, which contains a Git workflow action in the repository `Inverter_Motor_Controller`. The username has been switched out for this thesis.

```
37      - name: Unzip
38        run: |
39          unzip -o generated_code.zip
40
41      - name: Get version
42        id: version
43        run: |
44          echo "::set-output name=version::$(jq '.tag_name' -r asset_response
    .json)"
45
46      - name: Remove temporary files
47        run: |
48          rm generated_code.zip
49          rm asset_response.json
50
51      - name: Commit files
52        run: |
53          git config --local user.email "action@github.com"
54          git config --local user.name "GitHub Action"
55          git add -A
56          git commit -m ${{steps.version.outputs.version}} -a
57
58      - name: Push changes
59        uses: ad-m/github-push-action@v0.6.0
60        with:
61          github_token: ${{ secrets.GITHUB_TOKEN }}
62          branch: master
```

Listing 5.3: Code from on_release.yml, part 2 of 2, which contains a Git workflow action in the repository `Inverter_Motor_Controller`. The username has been switched out for this thesis.

Another GitHub Action was made in the repository `Inverter_Motor_Controller`, that listened to trigger events of the type `"new_release"`, that `Motor_Control_Simulink` released. This GitHub Action was defined as in Listing 5.2 and Listing 5.3.

From Listing 5.2, when a trigger of the type `"new_release"` is found after listening for it, the .zip-file called `generated_code.zip` containing all of the code generated .c-files is downloaded to the `Inverter_Motor_Controller` repository. Then, from Listing 5.3, the old code generated .c-files in the repository are removed, and the new zipped file is unzipped and replaces the old code generation files. The zipped file itself is then removed, and then the changes with the newest version of the repository is pushed to Git.

The `Inverter_Motor_Controller` repository is therefore kept up to date by GitHub Actions. This repository is a submodule in the `MPSoC_Inverter` repository, which contains the code that is run on the MPSoC. Whenever the submodule has a new commit on its repository, the user only has to update its submodules on `MPSoC_Inverter`, which takes one line of code in a shell:

```
$ git submodule update
```

or a few clicks using a Git GUI like Gitkraken. The `MPSoC_Inverter` already contains multiple submodules which needs to be kept up to date regardless, so adding another submodule will not add complexity to the system.

5.1.3 Implementation of the merging of the design with the existing code base

The code generation was done using Simulink's Embedded Coder. Since the inverter's Zynq Ultrascale+ MPSoC has both ARM Cortex A53 and R5 cores, both "Embedded Coder Support Package for ARM Cortex-A Processors" and "Embedded Coder Support Package for ARM Cortex-R Processors" was used for the process. The settings for which cores the code generation would build to, amongst other settings like what toolchain is used and which control compiler optimizations used, was configured in the MATLAB environment. Chosen compiler optimization strategies was improved execution speed and memory usage [31]. This was saved in a .m-file which was loaded and run at the project startup.

Several files had to be made to configure the code generation. These included files like generate_code.m. This code loaded the Simulink model to the workspace and generated the C code using the follwing MATLAB function:

```
slbuild(load_system(fullfile('..','motor_controller.slx')))
```

Further, the code generated .c- and .h-files were extracted and saved in `src` and `include` folders respectively. These folders were saved in a higher level folder called `generated_code`, which was manually zipped into `generated_code.zip`, which was used for the Git workflow mentioned in Section 5.1.2.

The wrapper consisted of a source and a header file, called `simulink_interface.c` and `simulink_interface.h`.

The source file looked as in Listing 5.5.

The structs `motor_controller_input_t` and `motor_controller_output_t` were defined as follows in the `simulink_interface.h`:

```c
typedef struct
{
    /* Currents */
    float current_u;  // [A]
    float current_v;  // [A]
    float current_w;  // [A]

    /* Flux and angle */
    float psi;        // [Wb]
    float theta;      // [deg]

    /* Speed */
    float speed_rpm;  // [rpm]

    /* Voltage */
    float voltage_dc; // [V]

} motor_controller_input_t;


/*
 * Motor Controller output:
 */

typedef struct
{
    /* Voltages */
    float phase_voltage_u;  // [V]
    float phase_voltage_v;  // [V]
    float phase_voltage_w;  // [V]

} motor_controller_output_t;
```

Listing 5.4: Structs defined in the wrapper `simulink_interface.h`.

```
1  void simulink_controller_set_inputs(motor_controller_input_t *
       motor_controller_input_p){
2
3      /* Currents */
4      motor_controller_U.Phase_Currents_g.current_u =
       motor_controller_input_p->current_u;
5      motor_controller_U.Phase_Currents_g.current_v =
       motor_controller_input_p->current_v;
6      motor_controller_U.Phase_Currents_g.current_w =
       motor_controller_input_p->current_w;
7
8      /* Flux and angle */
9      motor_controller_U.Mechanical_Measurements_h.psi_m_in =
       motor_controller_input_p->psi;
10     motor_controller_U.Mechanical_Measurements_h.tetta_in =
       motor_controller_input_p->theta;
11
12     /* Speed */
13     motor_controller_U.Speed = motor_controller_input_p->speed_rpm;
14
15     /* Voltage */
16     motor_controller_U.U_dc = motor_controller_input_p->voltage_dc;
17 }
18
19
20 void simulink_controller_step(motor_controller_output_t *
       motor_controller_output_p){
21
22     /* Iterate through the motor controller from input to output */
23     motor_controller_step2();
24
25     /* Get outputs */
26     motor_controller_output_p->phase_voltage_u = motor_controller_Y.
       Phase_Voltages[0];
27     motor_controller_output_p->phase_voltage_v = motor_controller_Y.
       Phase_Voltages[1];
28     motor_controller_output_p->phase_voltage_w = motor_controller_Y.
       Phase_Voltages[2];
29 }
```

Listing 5.5: Two of the functions from the wrapper `simulink_interface.c`, where the setter-function `simulink_controller_set_inputs()` updated the `motor_controller_input_t` struct, and `simulink_controller_step()` iterated once through the motor controller model, and then updated the `motor_controller_output_t` struct.

5.2 Implementation of a system using a Look-Up Table (LUT)

Figure 9.13 shows how the field weakening control can be implemented with the use of a custom made LUT.

Figure 9.14 is made as described in the design chapter Section 4.3.

The "2-D T(u)"-block illustrates the LUT. This system requires a modification of the original Simulink model, since the i_d and i_q no longer is calculated from the torque τ using Equation (10). Rather, the current i_d is found using V_{DC} as described in the Section 4.3, and then i_q is found using a LUT based on the values of i_d and the torque τ. This meant a total change in the "Is calculator"-subsystem, which originally looked like Figure 9.12.

Here, the mathematical block "hypot" is used to calculate the square root of sum of squares of the voltages.

The Fw_delay gain is a number between 0 and 1 that delays the entering of the field weakening region, and must be tuned when a LUT is acquired. This is necessary because it is desired to control when the field weakening region is entered. It is desired to enter the field weakening late, since i_d decreases during this region, and then the torque τ at any given i_q would decrease. Thus, torque per ampere decreases when field weakening is entered. This is a sacrifice that has to be made to reach higher speeds than the nominal speed of the motor.

5.3 Implementation of an improved calibration procedure

The implementation of the encoder calibration was first done manually, and then the design of a more automatic calibration technique was implemented.

5.3.1 Setup of test rig

The testbench of the motor is pictured as in Figure 5.4.

The motor is pictured on the left with the encoder coupled on the housing. The power is supplied through the orange cable from the VSI, which are the PCB's on the inverter that takes PWM signals as input and outputs the three phase currents (more on the VSI in chapter Section 2.1). In the photo they are shown underneath a cover, since they have 600 VDC running through them when the motor runs at peak speed with 20 krpm. This can kill instantly if touched, thus the protective cover is used as a precaution.

Because of the high amounts of current and voltage running through this card at peak power, they are situated on top of water cooling blocks. The water container is underneath the bench in a red plastic box, and the water is transported through the system with the help of an electric water pump.

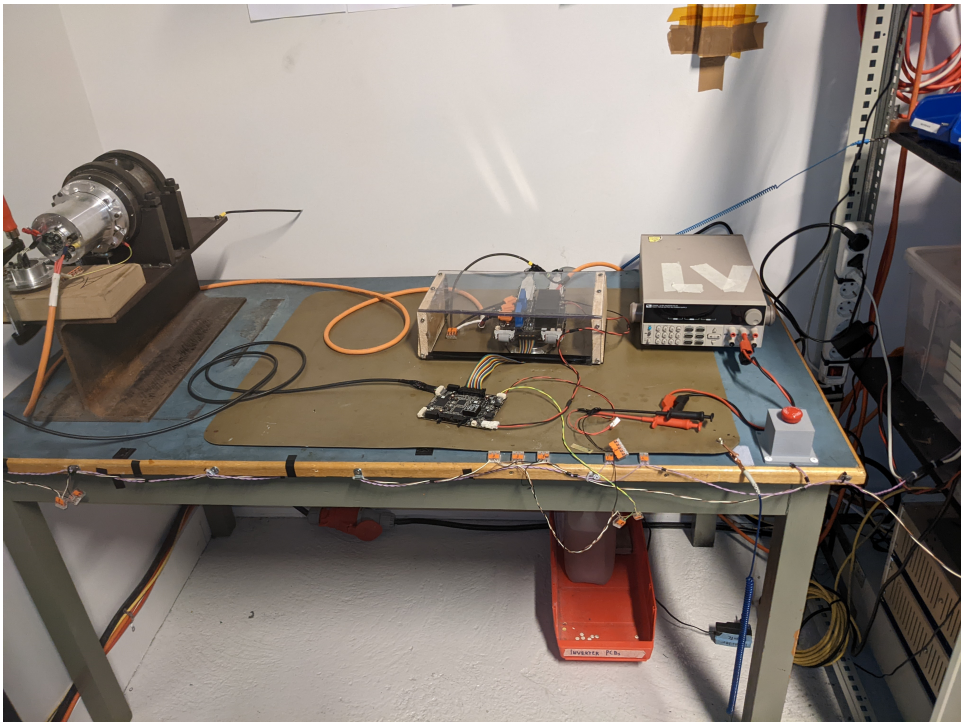


Figure 5.4: Setup of calibration at the motor testbench, with no power connected.

The PCB that is not underneath the cover is the control card, which is also supplied with 600 VDC from a voltage supply not pictured on Figure 5.4, and with 24 VDC through the power supply labeled "LV". It communicates with the VSI through ribbon cables. The control card is directly connected to the encoder through the black cable coupled on the PCB's with DuraClik connectors. It also has a CAN-bus that is used to retrieve data from the system.

Barely pictured on the right is a box filled with the PCB's that are normally in the cars accumulators' front compartment. These include the *Accumulator Management System* (AMS), that keeps track of the state of the accumulator; in this case the huge voltage supply that disburses 600 VDC through the system.

On the edge of a table is an emergency stop button, which is connected to the system through a *Shutdown Circuit* (SDC). There are two of these buttons near the bench, one is not pictured and is placed near the test rig by the computer that analyzes the output data from the control card. For safety measures, there are always minimum two people working on the motor together during high voltage testing.

5.3.2 Mounting

The size of the scanning gap between the rotor and the stator is dictated by the mounting situation. Later adjustment is possible only through the insertion of shim rings.

Figure 5.5: Four motors finished mounted and calibrated.

5.3.3 Implementation of a manual calibration procedure

First, 20A was provided from a power supply into the U-phase and out of the V- and W-phases of the VSI, thus from phase U to V and W. This should in theory align the rotor field (d-axis) with the U-phase (α-axis), as is desired from the Park-transform, see Figure 2.3. The applied 20A should then impose a stator field on the rotor in the direction of the U-phase (α-axis), and thus align the d- and α-axis.

From the assumption that the axes were aligned, the mechanical offset in the encoder datum shift was then manually sat using Heidenhain's firmware called the PWM20-tool, c.f. Section 2.7.

5.3.4 Implementation of an automatic calibration procedure

The semi-automatic calibration of the encoder was based on Benjamin G. Katz master thesis, see his algorithm described in more detail in Section 3.5. A similar algorithm procedure was done in this implementation:

- Firstly, the datum shift was set to 0 in the ATS via Heidenhain's PWM 20 tool. This was to set the encoder zero equilibrium.
- Then, a reference angle was set in the software of the control card, written in the code as in Listing 5.6. This was meant to spin the motor clockwise for one 360° cycle, and then counter-clockwise for another 360° cycle.
- The code was added to an already existing state machine, so the state `CALIBRATE_ENCODER` was created and checked for as in line 1 in Listing 5.6. Further, a variable `delta_angle` was created in line 6 to step through the cycle 0.1 rotation a second, for the frequency of `PWM_FREQUENCY_HZ = 20000` that the inverter runs on.
- After the `reference_angle` was updated, the `angle_offset` was set and cycled once positively for 360° if it was negative.

The encoder angle was measured and plotted together with the reference angle and the angle offset. Only a screenshot was taken from the first run with the offset angle, see Figure 5.6, and as can be seen from the plot, the offset was sinuous and when measured at any one particular timestamp, had a sizeable deviation of around ±17°. This was because the encoder angle was sinuous when measured, which again is a consequence of the cogging of the motor.

To counteract this, a higher d-current reference was used. Before, the q- and d-current references used were

$$\begin{aligned} i_{\mathrm{q\_ref}} &= 0 \\ i_{\mathrm{d\_ref}} &= \frac{3}{I_{\mathrm{Nominal}}} = \frac{3}{22.6}\mathrm{A} \approx 0.133\mathrm{A}. \end{aligned} \tag{16}$$

where I_{Nominal} is the motors nominal current, see Table 1.

```
if(status_check(CALIBRATE_ENCODER)) {

    // Up down 0-360 deg
    static bool counting_up = true;

    // 0.1 rotations per second
    float delta_angle = 0.1 * 360.0f / ((float)PWM_FREQUENCY_HZ);

    if(counting_up) {
        state.reference_angle += delta_angle;
        if(state.reference_angle >= 360.0f) {
            counting_up = false;
        }
    } else {
        state.reference_angle -= delta_angle;
        if(state.reference_angle <= 0.0f) {
            counting_up = true;
        }
    }

    //Set the encoder offset
    state.angle_offset = state.reference_angle - state.encoder_angle;

    if(state.angle_offset < 0) {
      state.angle_offset += 360.0f;
  }
}
```

Listing 5.6: Code snippet from the inverters control loop, where a reference angle is run for alternating 360° cycles.

The mitigation was done by amplifying the self-tuned constant in the denominator from 3 to 20. Thus, this increased the d-current reference to

$$i_{\mathrm{d\_ref}} = \frac{20}{22.6}\mathrm{A} \approx 0.885\mathrm{A} \tag{17}$$

This improved the deviation of the offset, see Figure 5.7. It reduced to a mere $\pm \approx 3°$.

Since the rotor is symmetrical, the motor cogging was identical for every cycle. Thus, an offset of the error between the reference angle and the measured angle was first measured in a period of $T = 100$s in Figure 5.6, and then taken the average of. This was increased to $T = 200$s for the second run, in addition to the over 6 time increase in d-current reference, see results in Figure 5.7. For the second run, cooling blocks were added to the inverter as to keep it from overheating when running with a higher current for so long. The 200 second run was long enough for the average of the offset to only differ in the third decimal, which was deemed as accurate enough after the previous errors were resolved.

After running for $T = 200$s, the cogging effects were virtually removed, and the offset of the two was added to the datum shift in the encoder soft-

ware. The encoder has 18-bit resolution, so the datum shift, Δ, is updated as

$$\Delta = 2^{18} * \frac{E_{\text{offset}}}{360^{\circ}} + E_{\text{abs}}, \tag{18}$$

where E_{offset} is the encoder offset in degrees, and E_{abs} is the exisiting absolute encoder position. This datum shift was manually added to the encoder through Heidenhains ATS and PWM 20 tool.

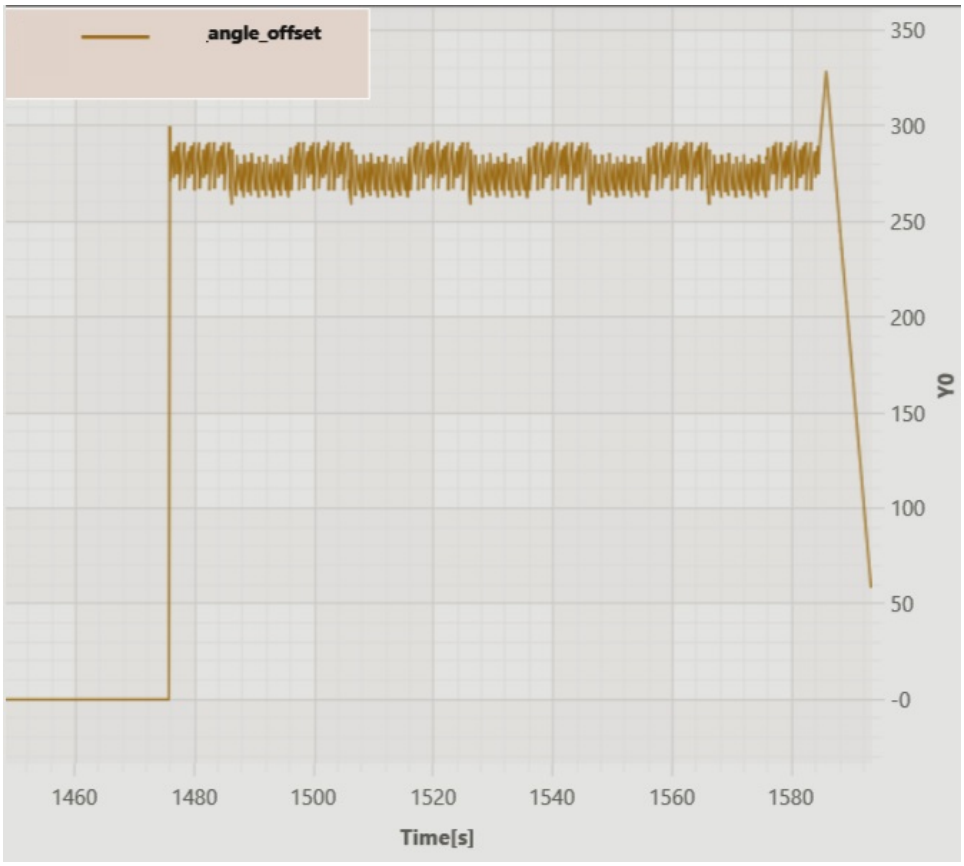


Figure 5.6: A screenshot taken of the measurement of the angular offset of the first run with the semi-automatic calibration method. The .csv-file of the measurement was not saved this day, and only the offset was recorded. The unit of the y-axis is degrees.

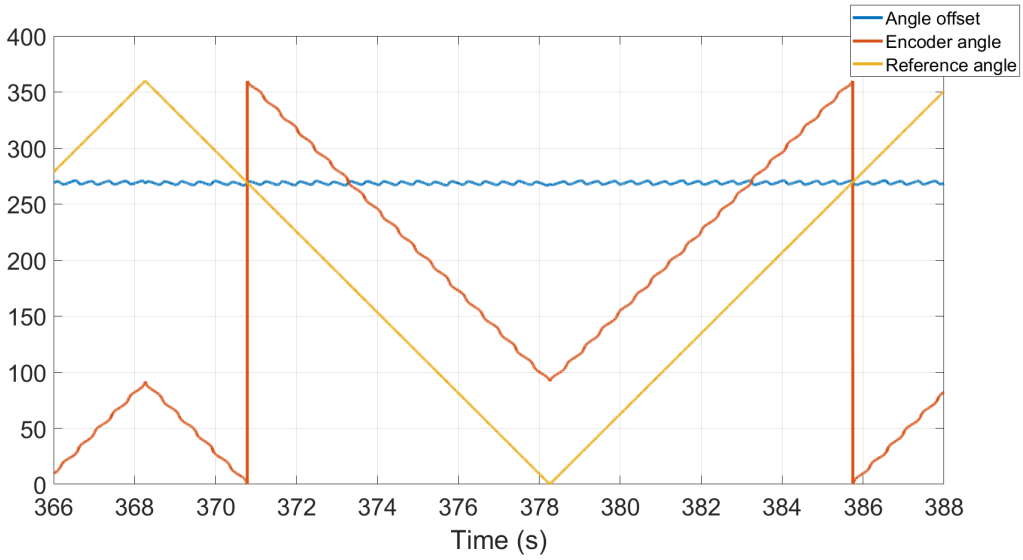


Figure 5.7: A plot of the measured encoder angle, the applied reference angle and the offset between the two.

6 Testing and Results

This chapter contains the testing procedures done on the designs implemented in Section 5. If tests weren't completed, it is described in each chapter. The results from each test on each implementation are presented, and then they are discussed in Section 7.

6.1 Testing and results of the motor control model

The motor control model was implemented as described in Section 5.1. The model was merged with the existing code base, and the resulting control loop was made and was ready to be run on the MPSoC and used to spin motors. The resulting current- and torque responses would validate the design of the motor control model. However, as described in Section 1.2, because of unsolved repeated errors in the firmware Xilinx Software Development Kit used to run the code on the MPSoC, the code could not be run nor validated. These build errors were attempted to be debugged without success; this is described in more detail in Section 1.2.

6.2 Testing and results of the motor control model with a Look-Up Table (LUT)

The motor control model was implemented as described in Section 5.2. The actual LUT was not made because a torque sensor was not acquired.

As the model had the same inputs and outputs as the motor controller implemented in Section 5.1, the testing of the model would have been done the same way as in Section 6.1. The code-generated motor control with a LUT could be merged with the existing code base and run in the same way on the MPSoC as in Section 5.1. The resulting current- and torque responses would validate the design of the motor control model. Again, because of unsolved repeated errors in the firmware Xilinx Software Development Kit used to run the code on the MPSoC, the code could not be run nor validated. These build errors were attempted to be debugged without success; this is described in more detail in Section 1.2.

6.3 Testing and results of an improved calibration procedure for the encoder

The testing of the calibration of the encoder was done first on the manual implementation, and then on the automatic implementation. The testing procedure was the same for both of the implementations.

6.3.1 Testing and results of manual calibration

After the datum shift was set with the manual calibration routine, the testing was performed by spinning the motor up to 2 krpm with field weakening control applied, and then validating the results.

This manual calibration routine took several hours spanning multiple days and gave only one of two results depending on how the datum shift in the encoder was misaligned.

Either, the results were as shown in Figure 6.1, which was a result of the encoder having a negatively misaligned datum shift. When the encoder had a misaligned datum shift in the other direction, results were as shown in Figure 6.2.

This shows the q-current growing large in either the positive or negative direction when entering the field weakening region, which again gave an unstable torque response.

Up to 10 hours were used on attempting to calibrate one encoder manually. Because of time limitations, a choice was made to stop trying to calibrate the encoders this way, as it took too much time. This resulted in a non-successful manual calibration.

6.3.2 Testing and results of automatic calibration

After the datum shift was manually added to the encoder through Heidenhains ATS and PWM 20 tool, tests were done to validate the precision of the calibration.

Tests were done by running the motor incrementally up to 20 krpm and then analyzing the resulting torque and current responses. This test consisted of a run-in procedure and was done as follows, shown in Table 6. The run-in procedure was done incrementally from low speeds to peak speed to spot errors with the calibration early on and to prohibit any significant electrical and mechanical damages.

The test was done without load on the motor, so it spun freely, and the power supply had a current limitation of 42 A_{rms}. Using this run-in procedure, some plots were saved. The resulting run at 2 krpm with field weakening control were as shown in Figure 9.15.

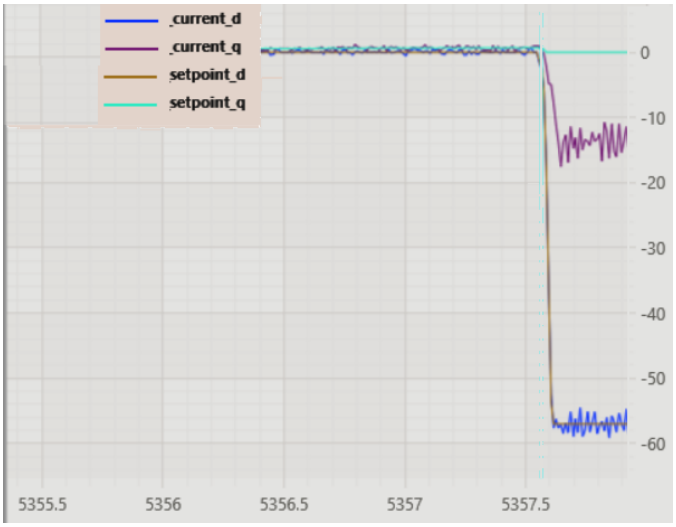
Speed reference (rpm)	Field weakening control applied?	Voltage applied to the transistors from the power supply (V)
1000	No	60
2000	No	60
2000	Yes	60
2000	No	600
5000	No	600
7000	No	600
10000	No	600
14000	No	600
14000	Yes	600
17000	Yes	600
20000	Yes	600

Table 6: Run-in procedure of the testing done to validate the calibration of the motor encoder.

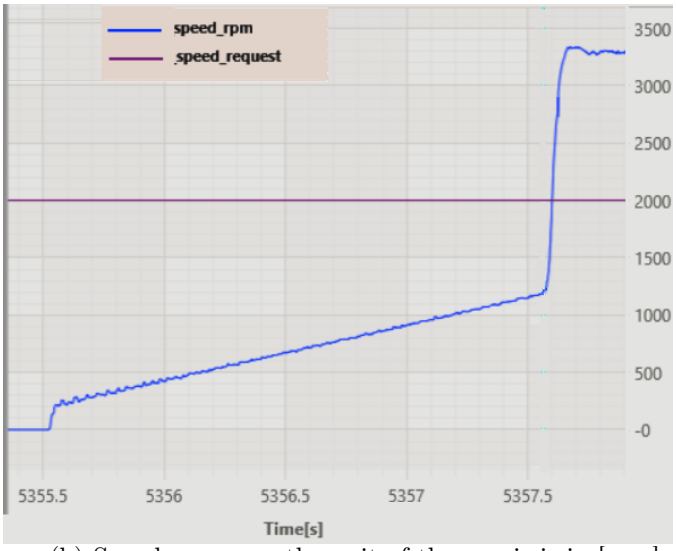
After a successful run-in procedure was performed, where the torque and current responses were shown to be satisfactory, the resulting run at 20 krpm with field weakening control were as shown in Figure 9.16.

As can be seen in Figure 9.16, after approximately 7.3 s, the speed of the motor hit the nominal speed of $n_{\text{Nominal}} = 13250$ rpm. This is where field weakening control was applied, and negative d-current was added to the system. At this stage, the reference q-current sank to ≈ -0.2 A, resulting in a proportionate decrease in torque. The torque sank to ≈ -0.2 Nm, where it stabilized with a variation of $\approx \pm 0.25$ Nm. It got a slightly higher variation than before the field weakening control was applied.

The calibration routine took 200 seconds, and the run-in procedure took about 500 seconds to complete. There were some breaks here and there to set the datum shift, analyze data between each run, and do safety checks that assured that the equipment was functioning as it should be (i.e., that the motor was not overheating). In total, the motor was successfully calibrated and validated in approximately 15 minutes.

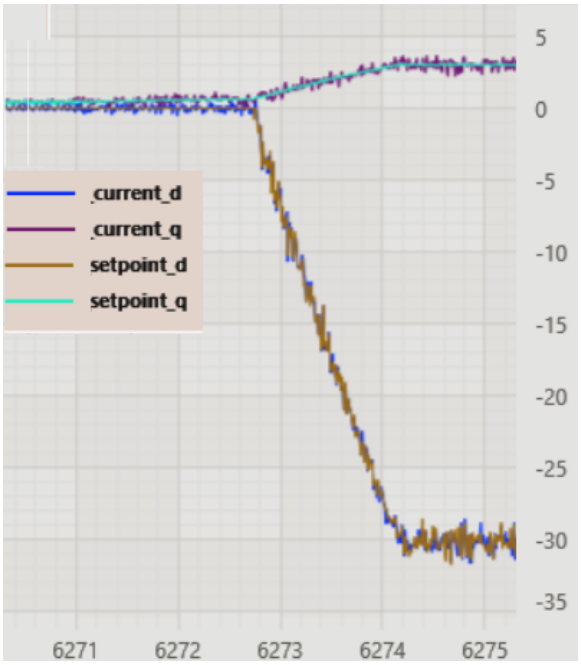


(a) Current response, the unit of the y-axis is in [A].

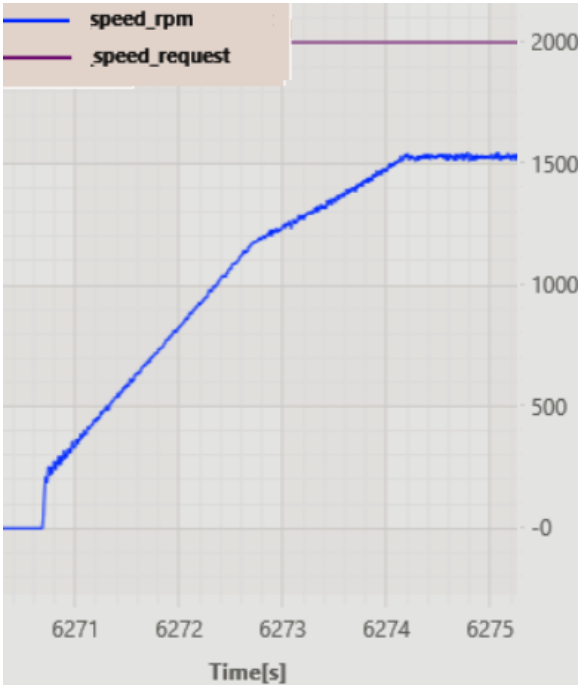


(b) Speed response, the unit of the y-axis is in [rpm].

Figure 6.1: Screenshots taken of the measurement of the speed and current responses after running the motor from 0 rpm to 2 krpm with the encoder having a negatively misaligned datum shift. The .csv-file of the measurement was not saved this day.



(a) Current response, the unit of the y-axis is in [A].



(b) Speed response, the unit of the y-axis is in [rpm].

Figure 6.2: Screenshots taken of the measurement of the speed and current responses after running the motor from 0 rpm to 2 krpm with the encoder having a positively misaligned datum shift. After the results from Figure 6.1, a limitation on the q-current was sat at $i_q = \pm$ 3A, where it saturated. This was done to prevent large torque responses and, therefore, mechanical damages. The .csv-file of the measurements was not saved this day.

7 Discussion

Firstly, the results of the implementation of the motor controller will be discussed. Following this, the results of the implementation of the motor controller using a Look-Up Table (LUT) will be discussed. Finally, the results from the improved calibration procedure will be discussed.

7.1 Discussion of motor control model

The motor control model was finished implemented, but because of time limitations on debugging existing build errors described in Section 1.2, the improvement of the runtime of the control model was not verified. However, because the motor controller already in use in the inverter is based on the motor controller from the simulator, it is expected to have similar torque and current responses as it already does. In addition, it is expected that the runtime of the motor controller to be shorter because of the code optimization done with respect to efficiency in execution speed and memory usage.

The implementation of the code structure was, however, a success. The resulting motor controller model was simplified from the motor controller in Simulink. One of the reasons for this was that all scopes and unused signals going out to terminators were removed for the ease of structural overview. Instead of having scopes in the block diagram, the signals that needed to be logged could be viewed in Simulink's own Data Inspector. This resulted in multiple subsystems being significantly smaller and less complex. The "Decoupling IM Motor" for instance, shown originally in Figure 9.4, ended up only being as shown in Figure 5.2. The "Current Measurement", shown in Figure 9.6 and "Voltage Measurement" subsystems mostly only outputted to terminators, which resulted in both of them being compressed to only a Clarke-transform and the "u_dc [pu]"-output being taken out to the motor controller overview, shown in Figure 9.9.

Additionally, due to the model being successfully implemented in Simulink, the model had more structural overview than the motor controller made in the C code.

This upholds the functional requirements **F2.1** and **F2.2**, and the technical requirement **T3** made in Section 4.1. It did not uphold the functional requirement **F2.3**, nor the acceptance requirement **A2**, as the implementation was not tested or validated.

7.2 Discussion of a motor controller using a Look-Up Table (LUT)

The model for the motor control with a look-up table was made in Simulink and implemented as in Section 5.2. It was made with the motor controller Section 5.1, and was made with the already existing model.

As the motor controller with a LUT has the same inputs and outputs as the motor controller made in Section 5.1, the same wrappers made in Section 5.1.3 and code organizational structure made in Section 5.1.2 could be used for the motor controller with a LUT as well.

However, because of limitations on resources of a proper motor test bench to make the measurements required in a LUT, the LUT itself was not made, and therefore the model could not be tested nor verified. To make the LUT, either a torque sensor would be required, or an already finished LUT calibrated explicitly according to the exact motor Revolve NTNU uses is required. The former is expensive and difficult to acquire; multiple known companies in the local area were contacted to try to borrow this without success; the latter could be gotten from Fischer, who makes and sponsors the motor Revolve NTNU uses. Fischer Electronics did not answer inquiries regarding this. Although the implementation of the LUT would give improvements on the torque ripples in the field weakening part of the motor control, it would not be realistic without any of those two aforementioned options.

This upholds the functional requirements **F2.4**, and the technical requirement **T4** made in Section 4.1.

7.3 Discussion of an improved calibration procedure

The tests of the calibration of both manual and automatic calibration of the encoder were done without load on the motor, so it spun freely. This was done to see the worst-case scenario on how the currents affected the torque response. This is because the motor torque is dependent on both the friction and any extra load on the motor. If the load is high, the acceleration of the torque will be low, and vice versa. Thus, the sensitivity of the torque response will be lower and more stable with more load added to the motor. Therefore, the test done in this thesis is the worst-case scenario in terms of the stability of the torque response.

7.3.1 Discussion of a manual calibration procedure

When trying to reach peak speed through field weakening control, manual calibration required significantly more q-current to maintain the same speeds as the auto-calibration did. This can be seen firstly in Figure 6.1, whereby only running the motor towards 2000rpm, and where the encoder had a negatively misaligned datum shift. Here, the q-current was varying at around

-14 A, which is significantly lower compared to when the encoder is more accurately calibrated, and varies around -0.1 A when run towards 2000 rpm, as can be seen in Figure 9.15. As a reaction to the sudden high absolute value of the q-current, a lot more torque was generated than what was desired, which happened because the torque and the q-current are proportionate. This resulted in a much higher speed response than desired and is why the speed response hit 3300rpm in Figure 6.1, where the reference speed was actually 2000rpm.

The d-current also dropped to a lower current, stabilizing at around -58A, compared again to the more stable run in Figure 9.15, where it stabilized at around -49A. This was likely because the motor controller tried to compensate for the decrease in q-current.

In Figure 6.2, the encoder had a positively misaligned datum shift. Here, the current responses were slightly more delayed than when the encoder was negatively misaligned. This could be because the misalignment was less for the particular run than when it was negatively misaligned - thus, the encoder offset was smaller. This is likely because the rotor was in a mechanical cog closer to the encoder zero reference in the positive direction than when it was in the mechanical cog in the negative direction compared to the encoder zero reference.

The q-current had a positive increase which saturated at 3A, which, as mentioned in Section 6.3.1 was done to prevent significant torque responses similar to what happened when running the motor with a negatively misaligned datum shift in the encoder. Because of the saturation in q-current, the torque saturated as well, which made the speed saturate at 1550 rpm. Without the saturation in place, the q-current would have likely increased more, resulting in an increase in torque, which would have made for a more unstable system. The d-current stabilized at around -30 A, which is more positive than when the encoder was negatively misaligned. This was likely because the motor controller tried to compensate for the increase in q-current.

Manual calibration was not completed because it took too much time. It was deemed a limitation on both the master thesis and the deadlines that the electrical systems on the car, including the motors and encoders, had to reach. Up to 10 hours were used on attempting to manually calibrate the encoder, which was a lot more than the 15 minutes needed to calibrate the encoders automatically.

’

7.3.2 Discussion of an automatic calibration procedure

The torque stabilized at ≈ -0.2 Nm, with a variation of $\approx \pm 0.2$ Nm. This was well within the min and max limits of the torque response. Since this run was without load, the variation of the torque response will only lessen running it with a load connected to the gearbox and upright on the car. This means that the torque response will be even more stable on the car.

During the manual calibration of the encoder, results were generated from when the encoder was not accurately calibrated. Under these circumstances, the current and torque needed to compensate for the misalignment in the datum shift were high and kept growing more unstable depending on the misalignment and the size of the offset.

When the encoder was more accurately calibrated, less current was needed to compensate for the misalignment, and fewer torque ripples were produced. This resulted in less power loss, which correlates to the results from Section 3.4.

This upholds the functional, technical, and the acceptance requirements made in Section 4.1.

This upholds the functional requirements **F2.1** and **F2.2**, and the technical requirement **T3** made in Section 4.1. It did not uphold the functional requirement **F2.3**, nor the acceptance requirement **A2**, as the implementation was not tested or validated.

8 Conclusion

This thesis was supposed to cover improvements on the motor control algorithm of the inverter of a Formula Student racecar. Implementation of the improvements was made; this included using a code-generated motor control model made in Simulink and merging this model with existing C code that runs on the Microprocessor System on Chip (MPSoC) of the inverter in the car. However, validation of this model was not performed because of time limitations, more closely described in Section 1.2.

Furthermore, this thesis did successfully implement a design based on the improvement of the motor control model that switched out the previous approximations of i_d calculated with numerous iterations of the Newton-Raphson method from a reference torque τ_{ref}, with a LUT. This LUT was intended to contain pre-measured values of i_d and τ, that could be run through faster than the NR-method with approximations. However, this implementation was not tested as well due to a lack of torque sensor equipment and time limitations.

In addition, improvements were designed, implemented, and validated on encoder calibration. A procedure was implemented that automatically found the stator offset angle. This was validated to have a significant improvement compared to the previous method of manually finding the offset. The improvement includes the reduced time from up to 10 hours, to only 15 minutes needed to finish calibrating one encoder for one motor. This improved procedure will benefit the encoder calibration process for the following years in Revolve NTNU.

9 Further work

Suggested further work on the implementation done in this thesis is listed below.

- Test and validate the implemented motor controller from Section 5.1.
- Make a LUT for i_q and τ_{ref} using measurements done with a torque sensor on the Fischer motors
- Test and validate the implemented motor controller from Section 5.2, with the aforementioned measured LUT.
- Save the corrected datum shift found in the autocalibration procedure described in Section 5.3.4, in the encoders non-volatile memory by using the EnDat 2.2 interface. This will render the PWM20-tool superfluous, and will make the calibration procedure even faster.
- Run the calibration routine as a function from the user interface as a button in the dashboard, using a CAN-message from the dashboard PCB to the I21 MPSoC. This can be run while the car is on stand. There already exists CAN-messages from the dashboard to the I21 that can spin the motors with specific speed setpoints, with and without field weakening. Thus, the run-in procedure used to validate the accuracy of the calibration presented in Table 6 can be run on stand as well.
  - The dashboard can run safety checks after the button is pressed, like checking if the damper position is negative that ensures that the car is on stand and not on ground. Another possible stand check would be a current check on how much inertia is needed to spin the motor. To do this, a threshold can be found on how much inertia is needed to spin the motor when the wheels have no friction and the car is on stand versus how much it needs when the car is on the ground. If the safety checks passes, the dashboard could send a CAN-message to the inverter, urging it to run the encoder calibration script.

References

[1] "About us" page at Revolve NTNU's official webpages. Revolve NTNU. Dec. 2021. URL: https://www.revolve.no/about-us (visited on 12/06/2021).

[2] C. M. G. Hartviksen. "Boot sequence for processors used in control of electric motors in a Formula Student racecar." In: Dec. 2021.

[3] Do-Hyun Jang and Duck-Yong Yoon. "Space-vector PWM technique for two-phase inverter-fed two-phase induction motors." In: 39 (Apr. 2003), pp. 542 –549. DOI: 10.1109/TIA.2003.809448.

[4] et al. O'Rourke C. J. "A Geometric Interpretation of Reference Frames and Transformations: dq0, Clarke, and Park." In: 34 (Dec. 2019), pp. 2070–2083.

[5] "Clarke Transform." In: Matlab. Jan. 2022. URL: https://se.mathworks.com/help/physmod/sps/ref/clarketransform.html?s_tid=doc_ta (visited on 01/14/2021).

[6] "Park Transform." In: Matlab. Jan. 2022. URL: https://se.mathworks.com/help/mcb/ref/parktransform.html (visited on 01/14/2021).

[7] Roy Nilsen. TET4120 Electric drives. Compendium. 2018.

[8] I.D. Mayergoyz and W. Lawson. "Chapter 1 - Basic Circuit Variables and Elements." In: Basic Electric Circuit Theory. Ed. by I.D. Mayergoyz and W. Lawson. San Diego: Academic Press, 1997, pp. 1–32. ISBN: 978-0-08-057228-4. DOI: https://doi.org/10.1016/B978-0-08-057228-4.50005-7. URL: https://www.sciencedirect.com/science/article/pii/B9780080572284500057.

[9] Mercury XU5. Product Overview. Enclustra, 2022. URL: https://www.enclustra.com/en/products/system-on-chip-modules/mercury-xu5/ (visited on 01/17/2022).

[10] Encoders for Servo Drives. Heidenhain, Nov. 2021. URL: https://www.heidenhain.com/fileadmin/pdf/en/01_Products/Prospekte/PR_Encoders_for_Servo_Drives_ID208922_en.pdf (visited on 05/19/2022).

[11] EnDat 2.2 – Bidirectional Interface for Position Encoders. Technical Information. Heidenhain, Sept. 2017. URL: https://www.endat.de/fileadmin/pdb/media/img/383942-28_EnDat_2-2_en.pdf (visited on 01/17/2022).

[12] Matlab. Field-Oriented Control (FOC). 2021. URL: https://se.mathworks.com/help/mcb/gs/implement-motor-speed-control-by-using-field-oriented-control-foc.html (visited on 01/24/2022).

[13] Matlab. Field-Weakening Control of IPMSMs. Apr. 2021. URL: https://se.mathworks.com/videos/field-weakening-control-of-interior-permanent-magnet-synchronous-motors-ipmsm--1617815717811.html (visited on 01/22/2022).

[14] T.R. Khramshin, G.P. Kornilov, and R.R. Khramshin. "Three-Level Inverter-Fed Direct Torque Control of the Synchronous Motor." In: Procedia Engineering 206 (Dec. 2017). DOI: 10.1016/j.proeng.2017.10.714.

[15] Seyed Hosseini, Seyed Sadeghzadeh, and Yousef BEROMI. "A new method for active power factor correction using a dual-purpose inverter in a yback converter." In: (Jan. 2016), p. 4741.

[16] Three phase currents and their respective PWM signals. Microchip. Nov. 2021. URL: https://microchipdeveloper.com/local--files/mct5001:pwm/sinetri3.png (visited on 12/06/2021).

[17] H.W. van der Broeck, H.-C. Skudelny, and G.V. Stanke. "Analysis and realization of a pulsewidth modulator based on voltage space vectors." In: IEEE Transactions on Industry Applications 24.1 (1988), pp. 142–150. DOI: 10.1109/28.87265.

[18] Matlab. Motor Control, Part 5: Space Vector Modulation. Jan. 2021. URL: https://www.youtube.com/watch?v=Gj7qAlsq_m8&list=PLn8PRpmsu08qL-EG3DRMtRyokpXQJyhp7&index=5 (visited on 01/18/2022).

[19] Yngve Solbakken. Space vector PWM intro. 2017. URL: https://www.switchcraft.org/learning/2017/3/15/space-vector-pwm-intro (visited on 01/24/2022).

[20] K. H. Nam. "AC Motor Control and Electrical Vehicle Applications, Second Edition." In: 2019. ISBN: 978-1-138-71249-2.

[21] Microchip Developer Help. Space Vector Modulation. 2022. URL: https://skills.microchip.com/zero-sequence-modulation-for-three-phase-motors/690239 (visited on 05/29/2022).

[22] B. Zhang and D. Qiu. m-Mode SVPWM Technique for Power Converters. 2019.

[23] H. K. Skeie. "Automotive Drive for a Formula Student Racecar." In: 2019.

[24] F. Fanin. "General Discontinuous Pulse Width Modulationalgorithm applied in an Electric Racing Car." In: 2019.

[25] F. Fanin. "Optimization algorithms for the currents in the Field Weakening region of an Interior Permanent Magnet Synchronous Motor of an Electric Racing Car." In: 2020.

[26] W. Chen, Y. Zhao, and Z. Zhou. "Torque Ripple Reduction in Three-Level Inverter-Fed Permanent Magnet Synchronous Motor Drives by Duty-Cycle Direct Torque Control Using an Evaluation Table." In: 2017.

[27] M. Begh and H. Herzog. "Comparison of Field Oriented Control and Direct Torque Control." In: Mar. 2018.

[28] N. Höglund and D. Poposki. "Evaluation of position and current sensor technologies for a PMSM used in automotive applications." In: 2019.

[29] Benjamin G. Katz. "A low cost modular actuator for dynamic robots." In: 2018.

[30] "MATLAB Coder Optimizations in Generated Code." In: Matlab. 2022. URL: https://se.mathworks.com/help/coder/ug/matlab-coder-optimizations-in-generated-cc-code.html (visited on 02/14/2022).

[31] "Performance - Reduce memory usage and improve execution speed of generated code." In: Matlab. 2022. URL: https://se.mathworks.com/help/rtw/performance.html (visited on 02/14/2022).

[32] Matlab. What is an S-Function? May 2022. URL: https://se.mathworks.com/help/simulink/sfg/what-is-an-s-function.html (visited on 01/14/2021).

Appendix

9.1 Fischer data sheet

FISCHER Elektromotoren

Motordatenblatt [berechnete Daten]

TI085-052-070-04B7S-07S04BE2

mit Feldschwächung

High-Speed & Power-Systems

Projektnummer: 2018-079-2

	Zeichen	Einheit	Wert	
Nenndaten Wasserkühlung (φ = 0°)				
Nennmoment	M_{NennWk}	Nm	11,1	
Nennstrom	I_{NennWk}	A_{eff}	22,6	
Nenndrehzahl	n_{NennWk}	U/min	13250	
abgegebene Wellenleistung	P_{NennWk}	W	15404	
Wicklungsverluste[1] / *Gesamtverluste* [1,2]	$P_{VNennWk}$	W	254	*617*
Stillstands-/ Haltemoment	M_{HaltWk}	Nm	7,9	
Stillstands-/ Haltestrom	I_{HaltWk}	A_{eff}	16	
Daten bei S6 Betrieb (φ = -10°)				
Drehmoment	M_{S6}	Nm	24,6	
Strom	I_{S6}	A_{eff}	51	
Drehzahl bei Drehmoment	n_{S6}	U/min	12100	
abgegebene Wellenleistung	P_{S6}	W	31199	
Wicklungsverluste[1] / *Gesamtverluste* [1,2]	P_{VS6}	W	1282	*1619*
Stillstands-/ Haltemoment	M_{HaltS6}	Nm	17,4	
Stillstands-/ Haltestrom	I_{HaltS6}	A_{eff}	35,9	
Daten bei Spitzenlast (φ = -10°)				
Spitzenmoment	M_{Peak}	Nm	29,1	
Spitzenstrom	I_{Peak}	A_{eff}	61	
Drehzahl bei Spitzenmoment	n_{Peak}	U/min	11600	
abgegebene Wellenleistung	P_{Peak}	W	35366	
Wicklungsverluste[1] / *Gesamtverluste* [1,2]	P_{VPeak}	W	1843	*2167*
Daten				
Drehmomentkonstante	k_t	Nm/A_{eff}	0,492	
Spannungskonstante (Phase - Phase)	k_e	$V_{eff}/(rad/s)$	0,296	
		$V_{eff}/(U/min)$	0,031	
Motorkonstante	k_m	Nm/√W	0,447	
Leerlaufdrehzahl	n_{Leer}	U/min	13650	
max. zul. Drehzahl (Feldschwächung)	n_{max}	U/min	20000	
max. Frequenz (Leerlauf/*Feldschw.*)	f_{max}	Hz	910	*1333*
Zwischenkreisspannung	U_{Zk}	V_{DC}	600	
Ø Widerstand pro Phase (nur Wicklung)	R_{Ph20}	Ω	0,126	
Ø Induktivität pro Phase (nur Wicklung)	L_{Ph}	mH	0,393	
elektr. Zeitkonstante τ=L/R	τ_{el}	ms	3,11	
Polpaarzahl	n		4	
Schaltung			Stern	

Motordatenblatt [berechnete Daten]

TI085-052-070-04B7S-07S04BE2

mit Feldschwächung

High-Speed & Power-Systems

	Zeichen	Einheit	Wert
Daten Wasserkühlung			
Eintrittstemperatur Kühlmittel	T_{ein}	°C	10 ... 40
Max. zul. Kühlmitteltemperaturerhöhung	T_{max}	K	5
Min. erforderlicher Kühlmitteldurchfluss	Q_{min}	l/min	---
Volumen Kühlkanal	$V_{kühl}$	l	---
thermische Zeitkonstante	τ_{th}	min	---
Daten Mechanik			
Drehmasse Rotor (Einbausatz)	J	kgm²	0,33*10^-3
Motorgewicht ohne Gehäuse	m	kg	2,8
Statoraußendurchmesser ohne Gehäuse	d_A	mm	85
Statorinnendurchmesser	d_{LS}	mm	51,6
Eisenlänge	l	mm	70

Anmerkungen - Verluste

[1] Wicklungsverluste sind bezogen auf eine Spulentemperatur von 100°C.

[2] Die Gesamtverluste setzen sich zusammen aus: Wicklungsverluste; Statoreisenverluste; Rotorverluste;

Berechnung der Gesamtverluste:

Wicklungsverluste + Statoreisenverluste (bei Drehzahl X) + Rotorverluste (bei Drehzahl X)

Anmerkungen - allgemein

Achten Sie darauf, dass Ihr Regler den Motornenn- und Spitzenstrom bereitstellen kann.
Eine Anpassung der Drehzahl und Zwischenkreisspannung kann nach Rücksprache erfolgen.
Die im Datenblatt angegebenen Nenndaten gelten für eine Umgebungs-/Kühlmitteltemperatur von 20°C.
Die Drehmomente sind angegeben ohne Berücksichtigung der Reibverluste durch Lagerung oder Dichtungen.

Anmerkungen - Temperaturüberwachungssystem

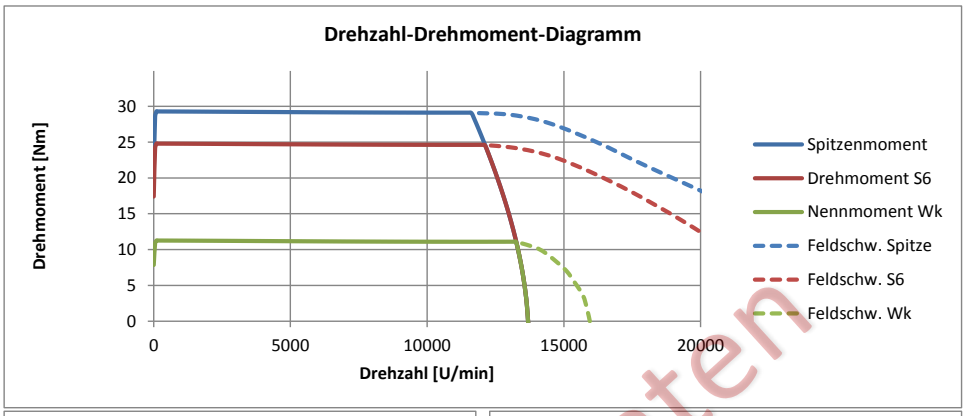
Da die genaue Betriebsart auch von der thermischen Anbindung des Motors abhängt, muss das eingebaute Temperaturüberwachungssystem ausgewertet und berücksichtigt werden. Dennoch gilt zu beachten, dass die Thermosensoren nicht die exakte Wicklungstemperatur anzeigen und diese durch thermische Kapazitäten um bis zu 20 K höher sein kann. Trotz einer elektrischen Isolation der Sensoren gegenüber der Wicklung dürfen die Sensoren nur über eine zusätzliche galvanische Trennung an den Regler/die Steuerung angeschlossen werden.

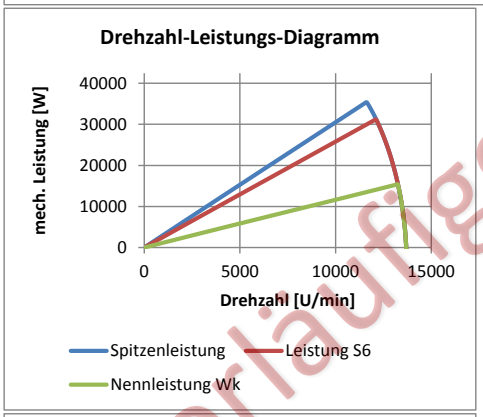
Motordatenblatt [berechnete Daten]

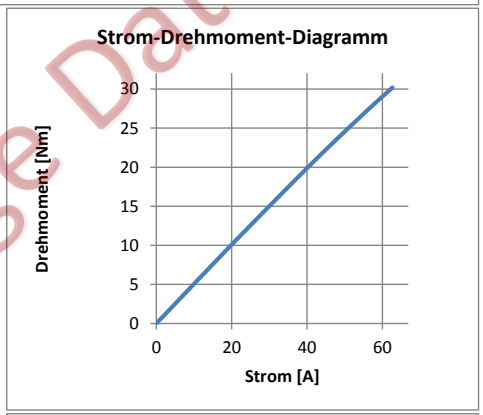
TI085-052-070-04B7S-07S04BE2

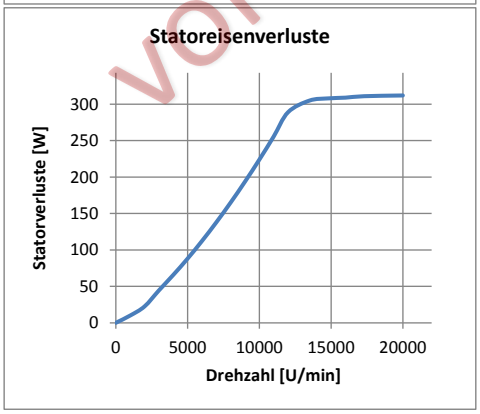
mit Feldschwächung

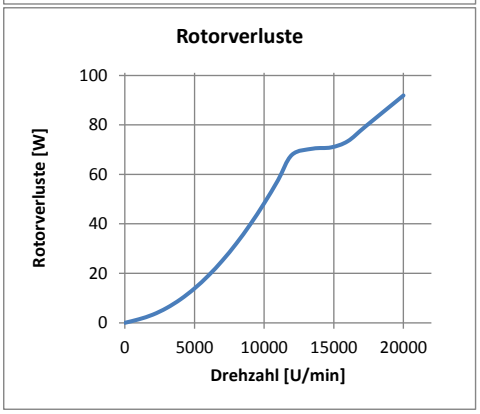
High-Speed & Power-Systems











Fischer Elektromotoren GmbH
Schützenstraße 19
D-74842 Billigheim-Allfeld
Geschäftsführer: Peter Fischer
Tel.: 06265/9222-0
Fax: 06265/9222-22
info@fischer-elektromotoren.de
www.fischer-elektromotoren.de

FISCHER Elektromotoren

Motor Datasheet [calculated Values]

TI085-052-070-04B7S-07S04BE2

with Fieldweaking

High-Speed & Power-Systems

Project-No.: 2018-079-2

	Symbol	Unit	Value	
Rated Data Water cooled (φ = 0°)				
Nominal Torque	T_{NomWC}	Nm	11,1	
Nominal Current	I_{NomWC}	A_{rms}	22,6	
Nominal Speed	n_{NomWC}	rpm	13250	
Nominal Power	P_{NomWC}	W	15404	
Winding Losses[1] / *Total Losses* [1,2]	P_{DWC}	W	254	*617*
Holding Torque	T_{HWC}	Nm	7,9	
Holding Current	I_{HWC}	A_{rms}	16	
Rated Data S6 duty (φ = -10°)				
Torque	T_{S6}	Nm	24,6	
Current	I_{S6}	A_{rms}	51	
Speed	n_{S6}	rpm	12100	
Power	P_{S6}	W	31199	
Winding Losses[1] / *Total Losses* [1,2]	P_{DS6}	W	1282	*1619*
Holding Torque	T_{HS6}	Nm	17,4	
Holding Current	I_{HS6}	A_{rms}	35,9	
Peak Data (φ = -10°)				
Peak Torque	T_{Peak}	Nm	29,1	
Peak Current	I_{Peak}	A_{rms}	61	
Speed at Peak Torque	n_{Peak}	rpm	11600	
Peak Power	P_{Peak}	W	35366	
Winding Losses[1] / *Total Losses* [1,2]	P_{DPeak}	W	1843	*2167*
Data				
Torque Constant	k_t	Nm/A_{rms}	0,492	
BEMF Constant (Phase - Phase)	k_e	V_{rms}/(rad/s)	0,296	
		V_{rms}/rpm	0,031	
Motor Constant	k_m	Nm/√W	0,447	
Idle Speed	n_{idle}	rpm	13650	
max. Speed (Fieldweaking)	n_{max}	rpm	20000	
max. Frequency (Idle/*Fieldweaking*)	f_{max}	Hz	910	*1333*
DC Bus Voltage	U_{DC}	V_{DC}	600	
Ø Resistance per Phase (Winding only)	R_{Ph20}	Ω	0,126	
Ø Inductance per Phase (Winding only)	L_{Ph}	mH	0,393	
electr. Time Constant τ=L/R	τ_{el}	ms	3,11	
Number of Polepairs	n		4	
Winding Connection			Star	

Motor Datasheet [calculated Values]

TI085-052-070-04B7S-07S04BE2

with Fieldweaking

High-Speed & Power-Systems

	Symbol	Unit	Value
Data Watercooling			
Inlet Temperature of Coolant	T_{in}	°C	10 ... 40
Max. Temperature rise of Coolant	T_{max}	K	5
Min. required Coolant flow	Q_{min}	l/min	---
Volume of cooling channel	V_{cool}	l	---
thermal Time Constant	τ_{th}	min	---

Data Mechanics			
Rotor Inertia (assembly set)	J	kgm²	0,33*10^-3
Weight of Motor w/o Housing	m	kg	2,8
Outer Stator Diameter w/o Housing	dA	mm	85
Inner Stator Diameter	dAg	mm	51,6
Length of Stator	l	mm	70

Annotations - Losses

[1] Winding Losses are referred to a Coil Temperature of 100°C.
[2] The total Losses are made up of: Winding Losses; Stator Iron Losses; Rotor Losses;
Calculation of total Losses:
Winding Losses + Stator Iron Losses (at speed X) + Rotor Losses (at speed X)

Annotations - general

Ensure that your servo drive can handle the Nominal- and Peakcurrent of the Motor.
An adjustment of the Speed and DC Bus Voltage can be done after consultation.
The nominal data in this datasheet are based on an ambient/coolant temperature of 20°C
The stated nominal Torques are without consideration of friction losses through Bearings or Sealings.

Annotations - thermal monitoring system

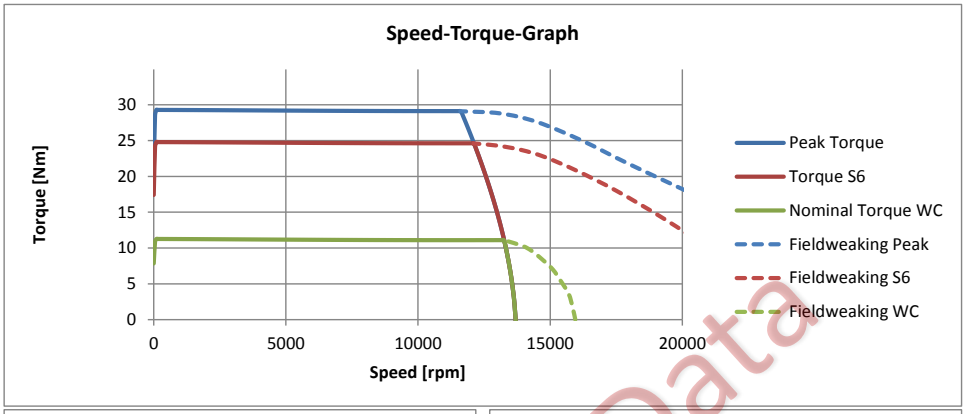
Because the exact duty type depends also on the thermal connection of the motor, the embedded thermal monitoring system has to be analysed and attented. However, attention has to be payed that the temperature sensors do not show the exact temperature of the winding and this could be up to 20 K higher due to thermal capacities. Despite an electrical insulation towards the winding, you are only allowed to connect the sensors to your controller by using a galvanic separation in between.

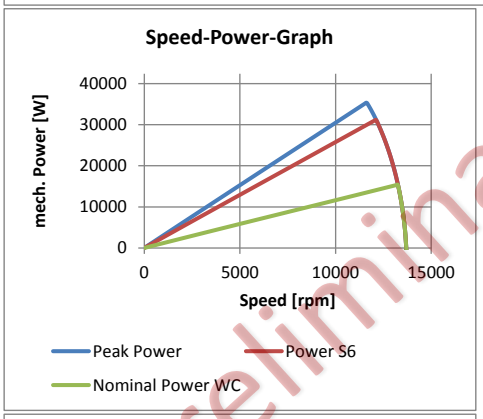
Motor Datasheet [calculated Values]

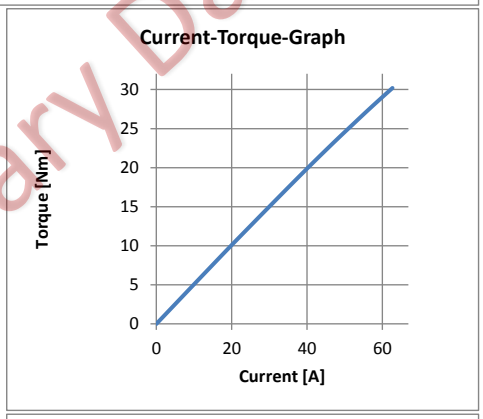
TI085-052-070-04B7S-07S04BE2

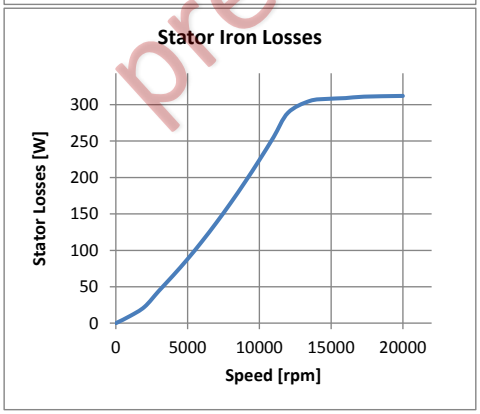
with Fieldweaking

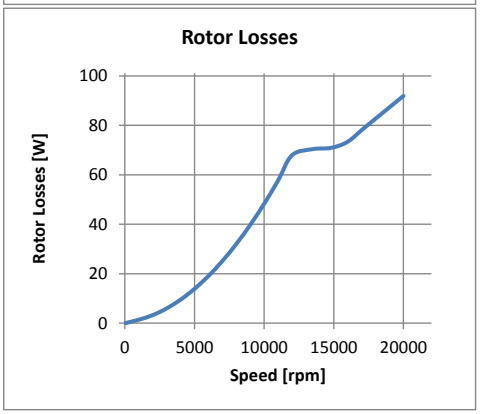
High-Speed & Power-Systems











Fischer Elektromotoren GmbH
Schützenstraße 19
D-74842 Billigheim-Allfeld
Geschäftsführer: Peter Fischer
Tel. (0049) 6265/9222-0
Fax: (0049) 6265/9222-22
info@fischer-elektromotoren.de
www.fischer-elektromotoren.de

9.2 Original motor simulator

The following models are taken from the simulator made by Håkon Skeie in 2019 during his master thesis, based on the work by NTNU professor Roy Nilsen's creation of a motor control of a PMSM in the class TET4120 *Electric Drives*. [7]

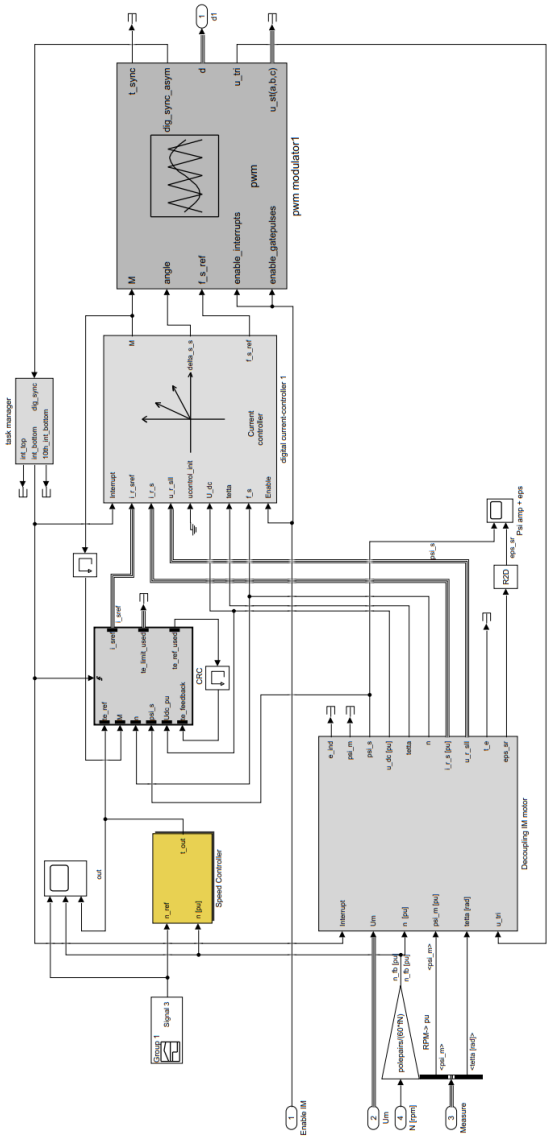


Figure 9.1: The motor controller, inside the "PM Motor Control for 3 phase machine" subsystem in Figure 4.1.

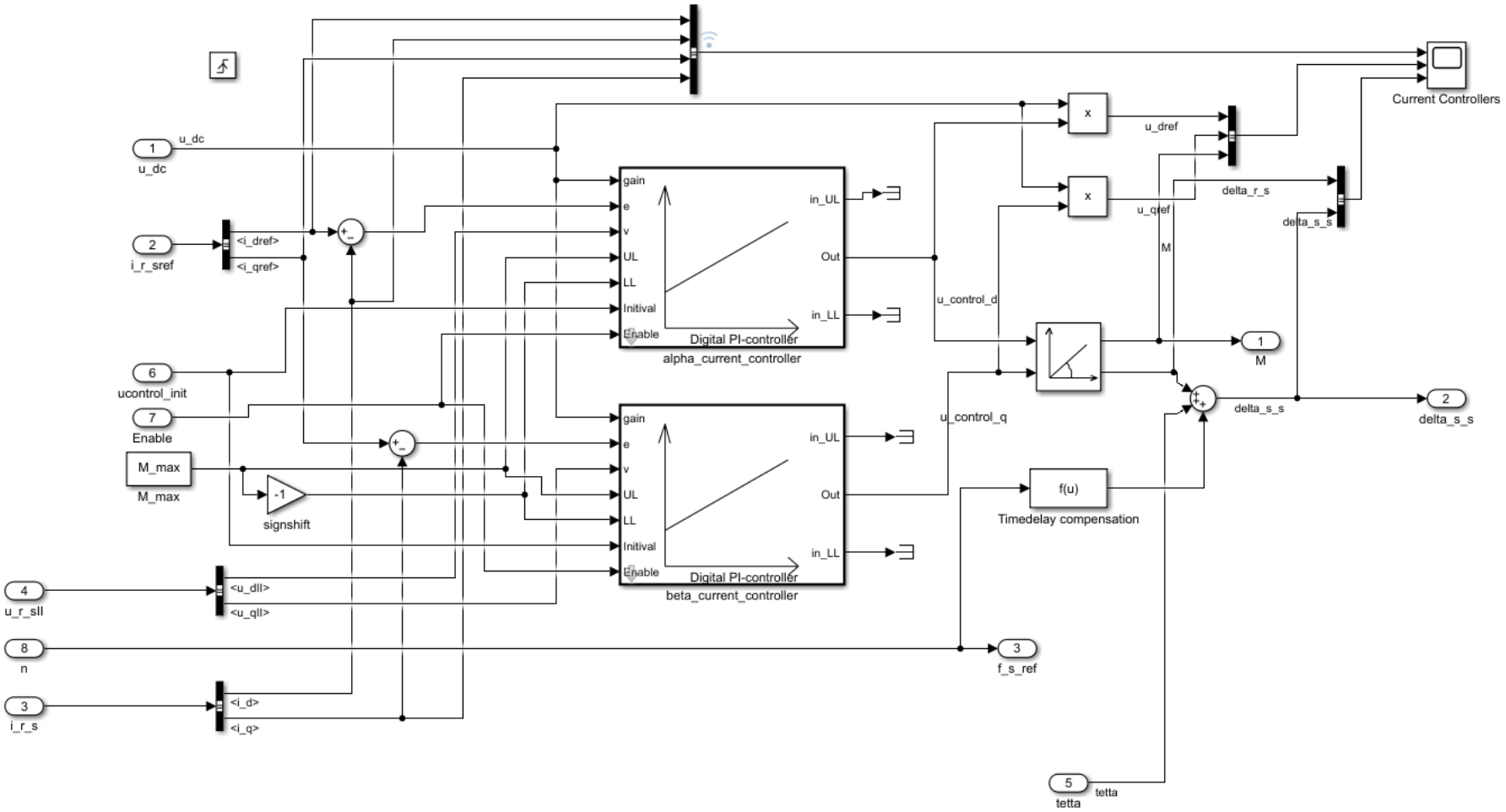


Figure 9.2: The contents of the "digital current-controller 1" subsystem in Figure 9.1.

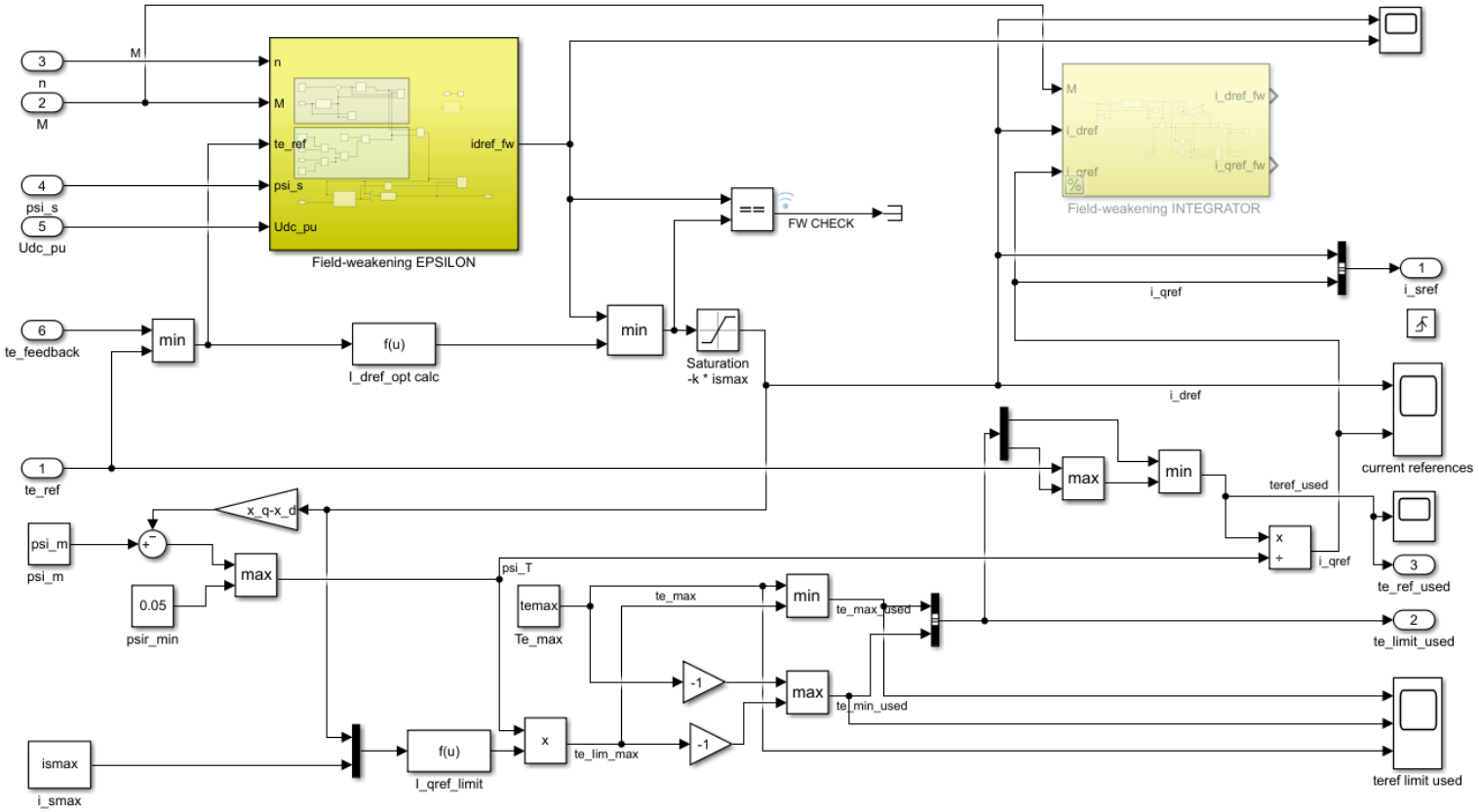


Figure 9.3: The contents of the "CRC" subsystem in Figure 9.1.

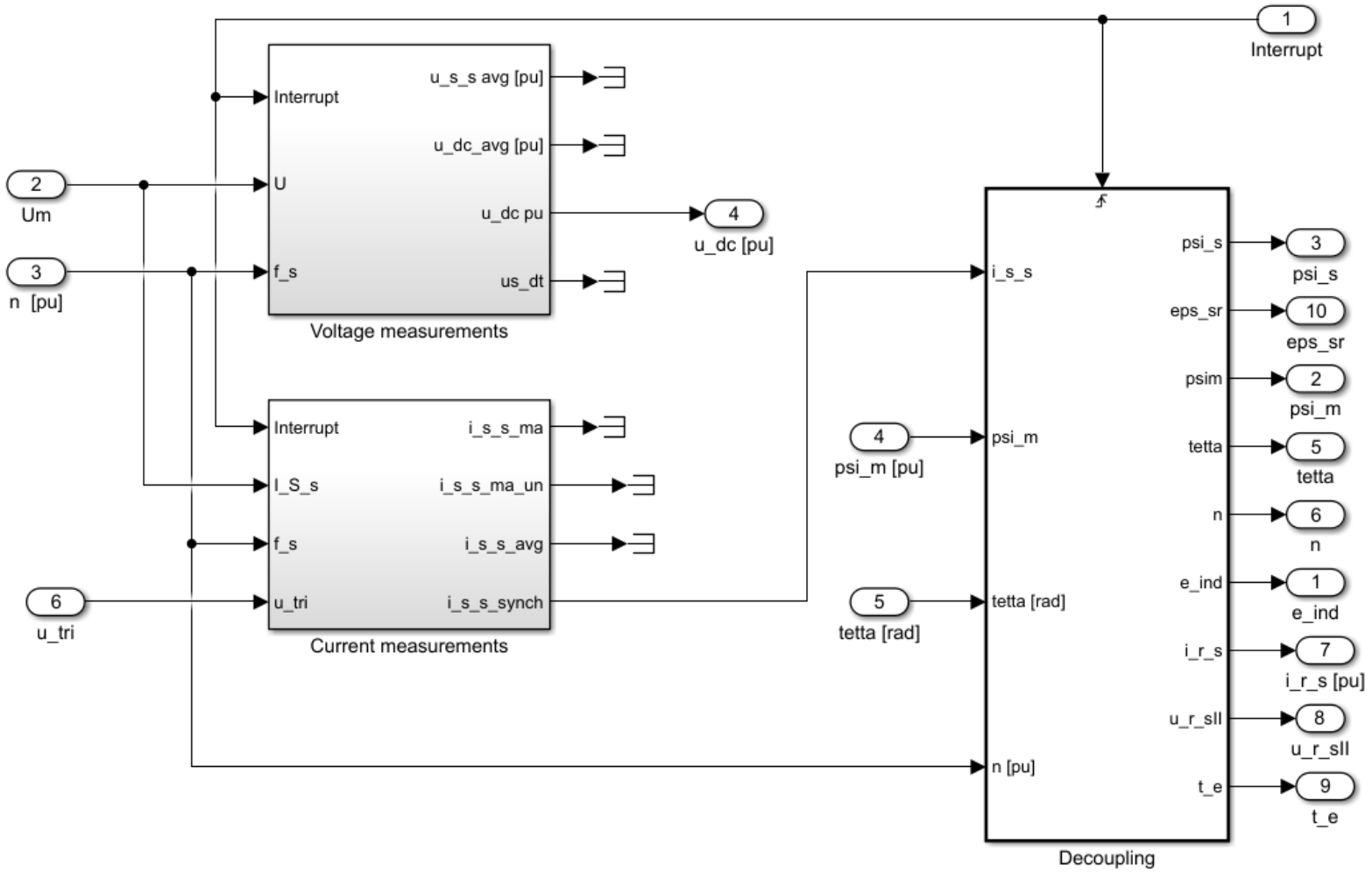


Figure 9.4: The contents of the "Decoupling IM" subsystem in Figure 9.1.

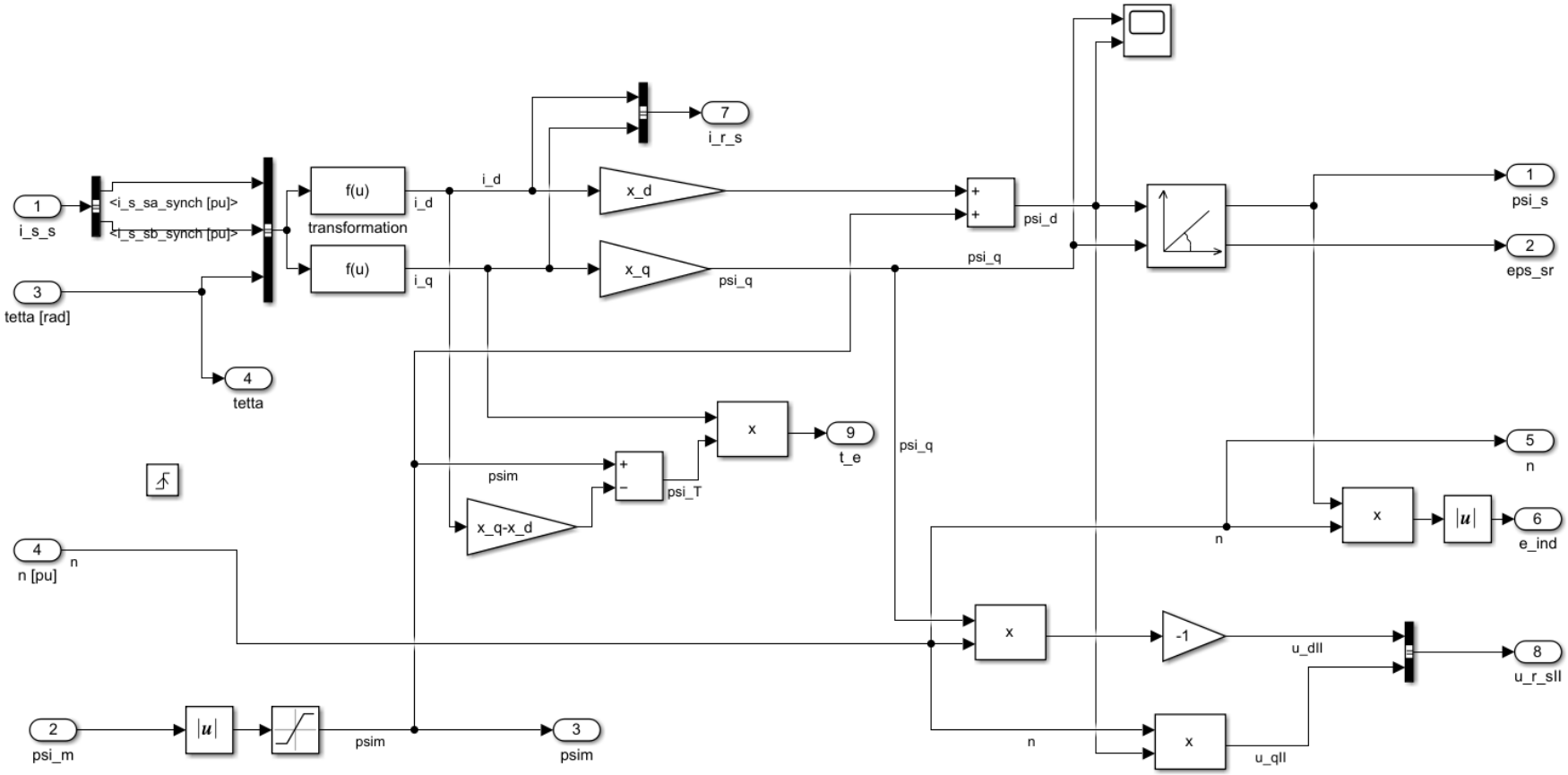


Figure 9.5: The contents of the "Decoupling" subsystem in Figure 9.4.

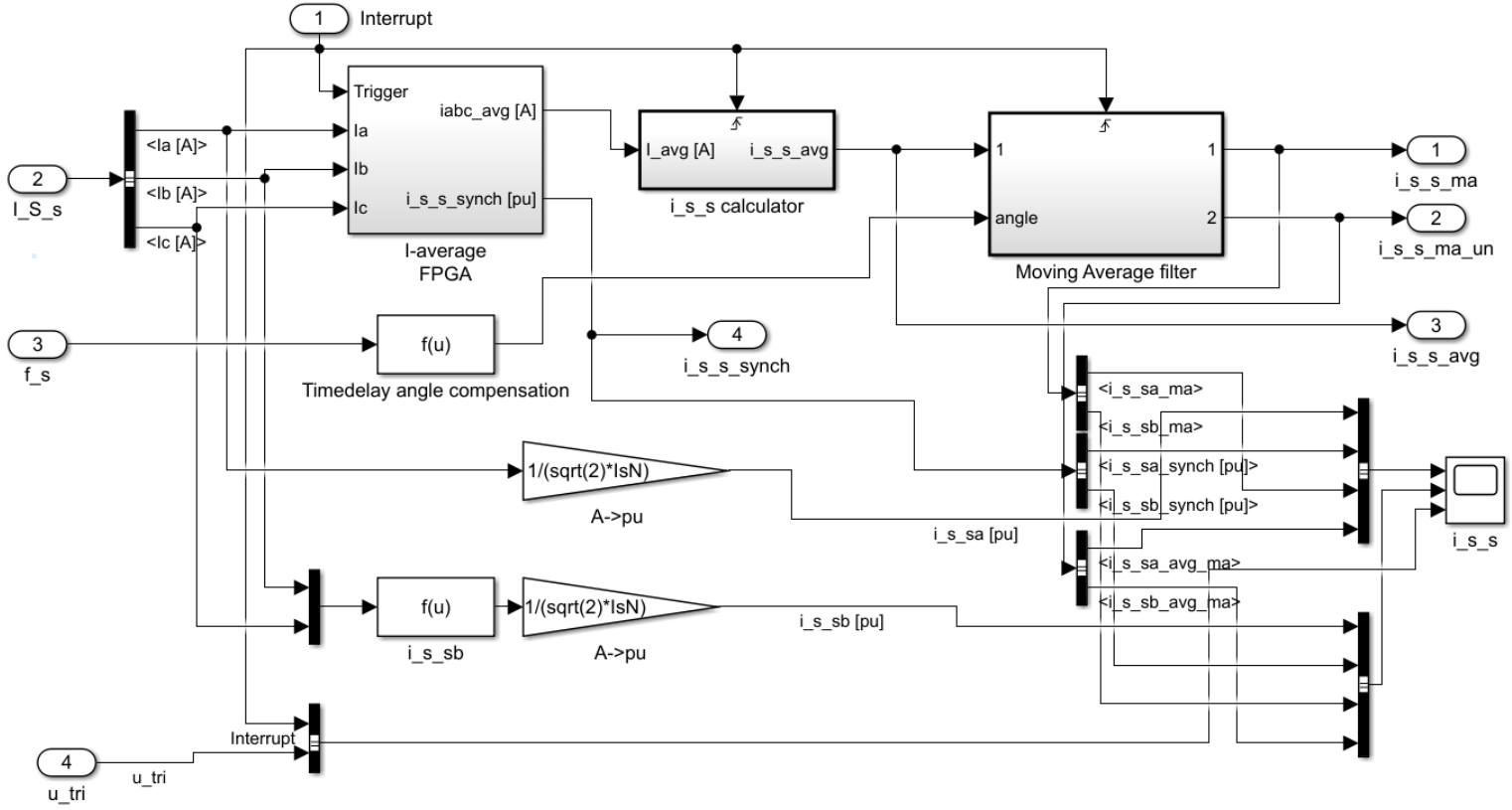


Figure 9.6: The contents of the "Current measurements" subsystem in Figure 9.4.

9.3 Scaled-up figures from Section 2

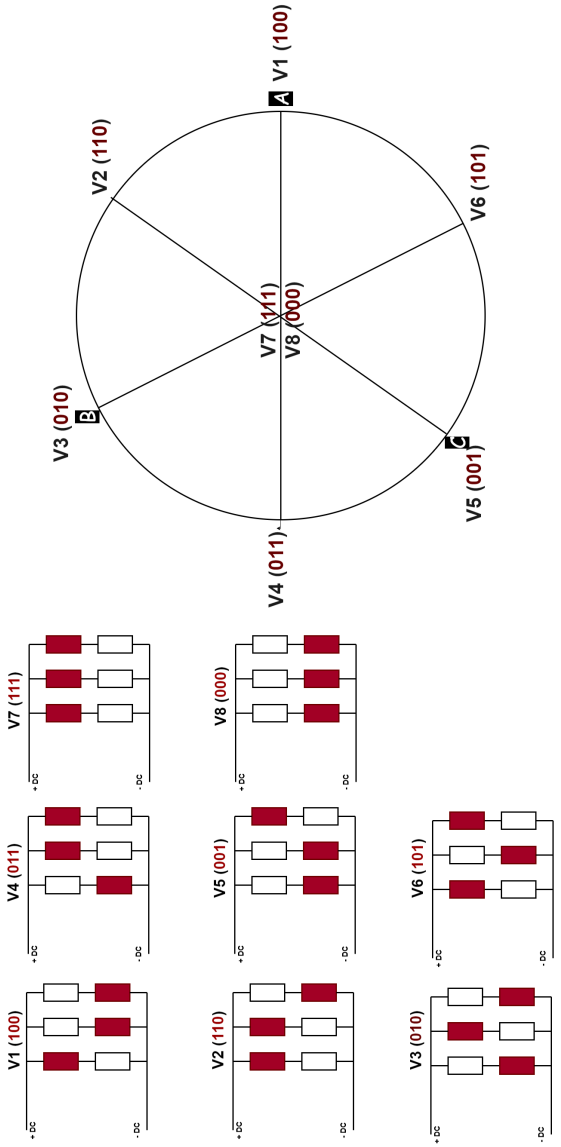


Figure 9.7: All eight configurations of the transistors are shown on the left, correspond to one space vector in the motor. V1-V6 creates six basic vectors that go along the six lines shown in the sphere. V7 and V8 create null vectors, shown in the origin. Figure inspired from [18]

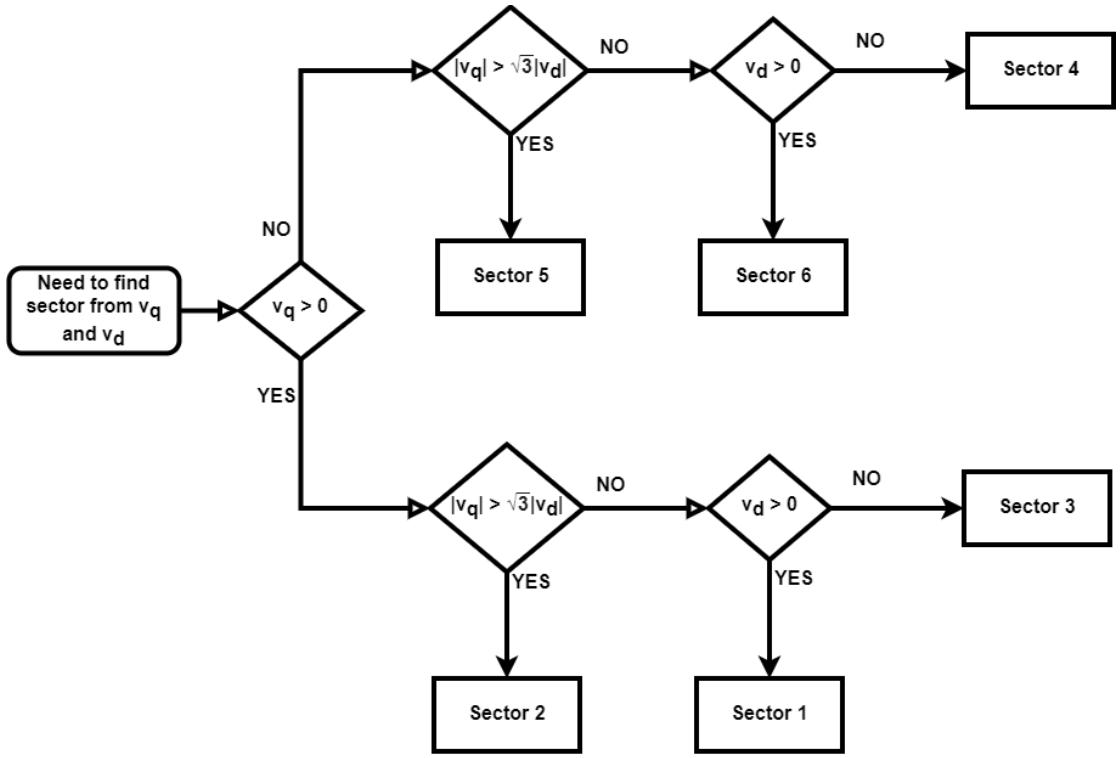


Figure 9.8: Image representing flowchart of the sector finding algorithm from the dq-voltages from the stator. Figure inspired from chapter 10.2.4 in [20].

9.4 Improved motor simulator

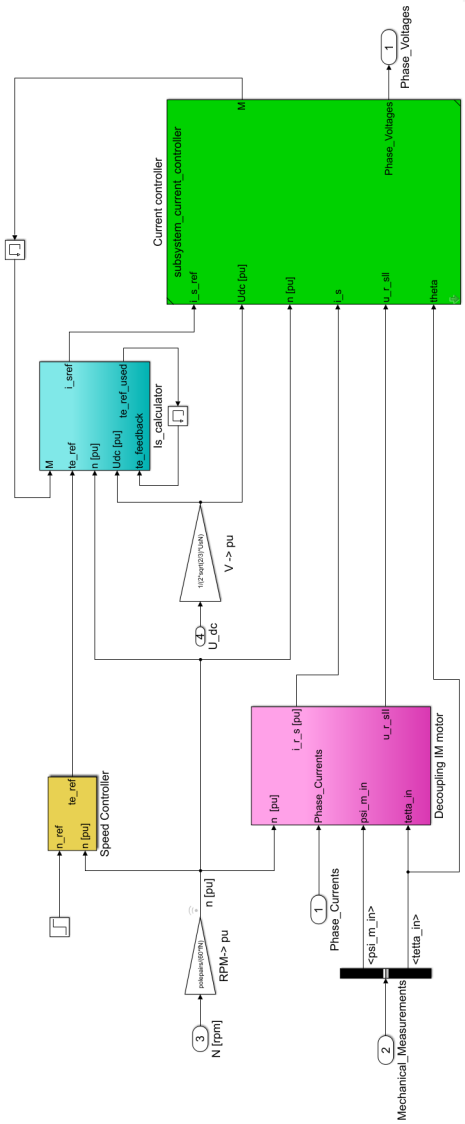


Figure 9.9: Overview over the contents of the motor controller.

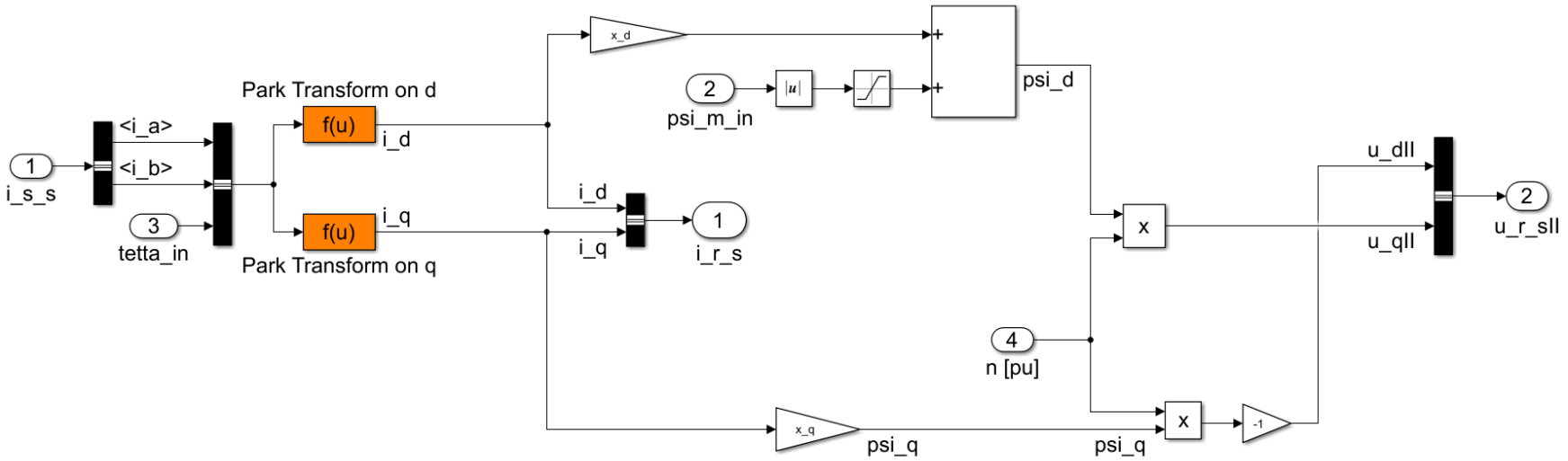


Figure 9.10: The contents of the "Decoupling"-subsystem from the "Decoupling IM Motor" shown in Figure 5.2.

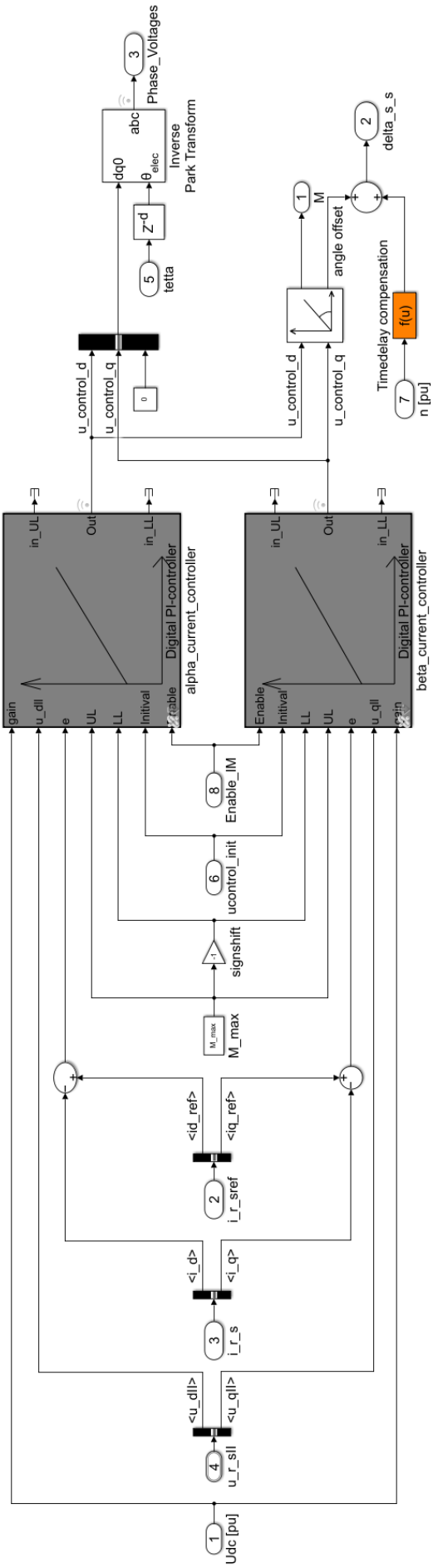


Figure 9.11: The contents of the "Current controller"-subsystem from the motor controller shown in Figure 9.9.

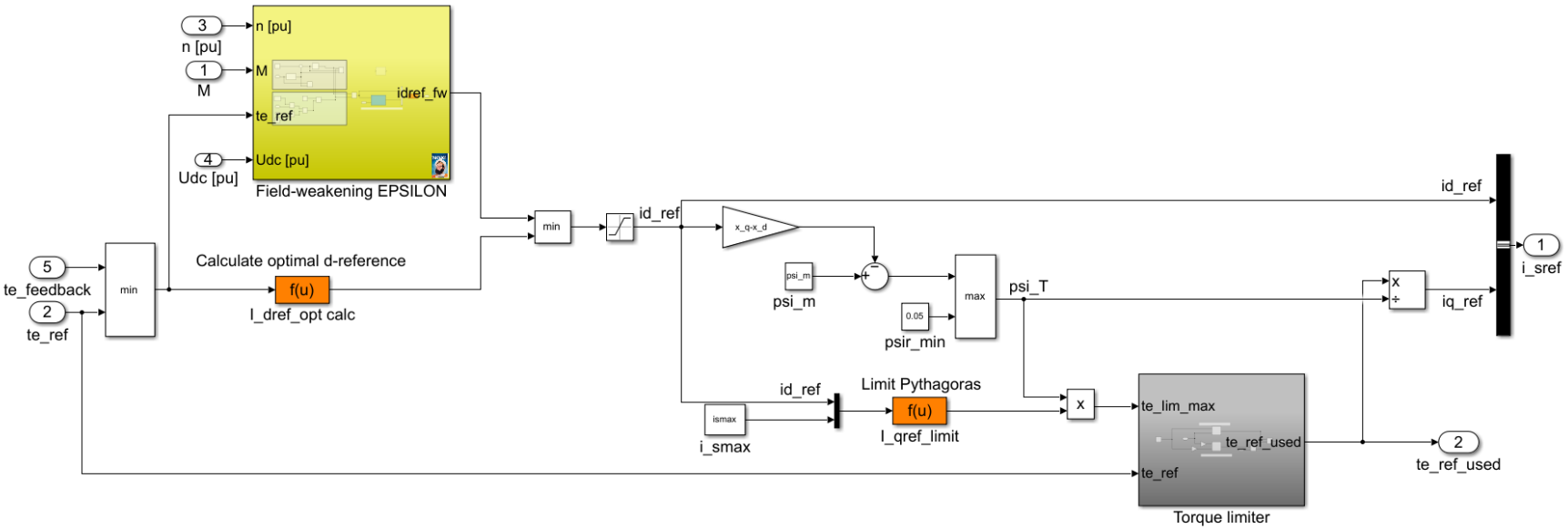


Figure 9.12: The contents of the "Is_calculator"-subsystem from the motor controller shown in Figure 9.9.

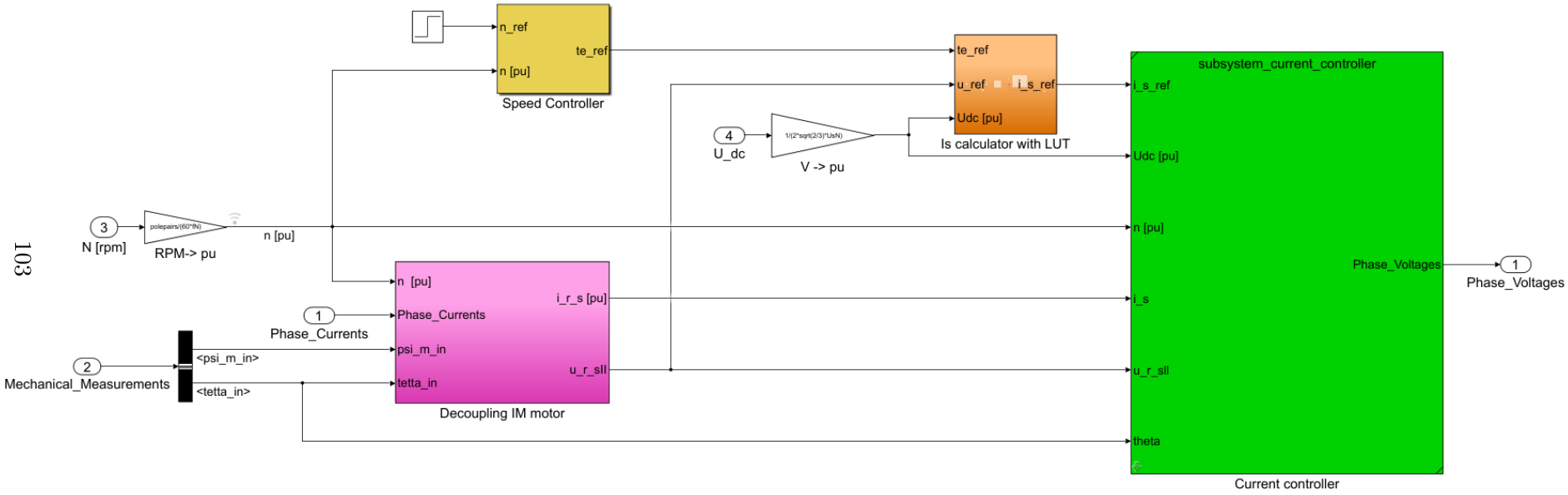


Figure 9.13: Simulink block diagram showing the upper most level of the motor controller model with an I_s- calculator based on a LUT.

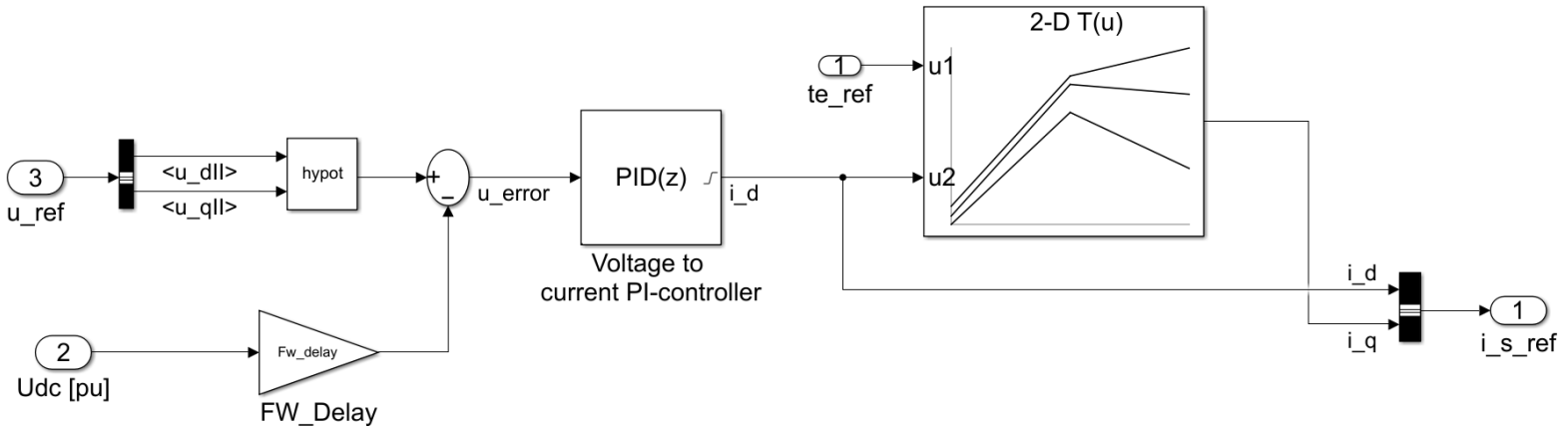


Figure 9.14: The contents of the "Is_calculator with LUT"-subsystem from the motor controller shown in Figure 9.13.

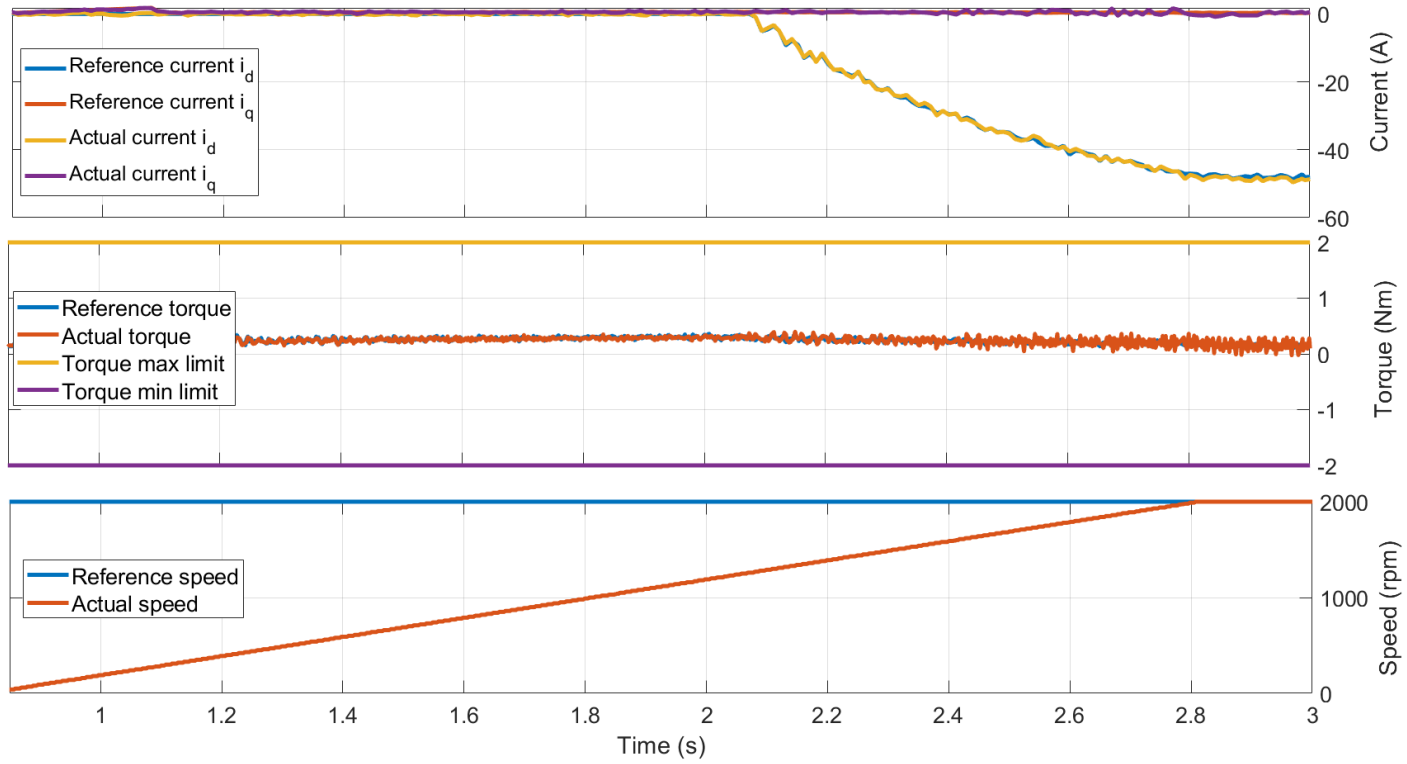


Figure 9.15: Current, torque and speed responses from running the motor from 0 rpm to 2 krpm.

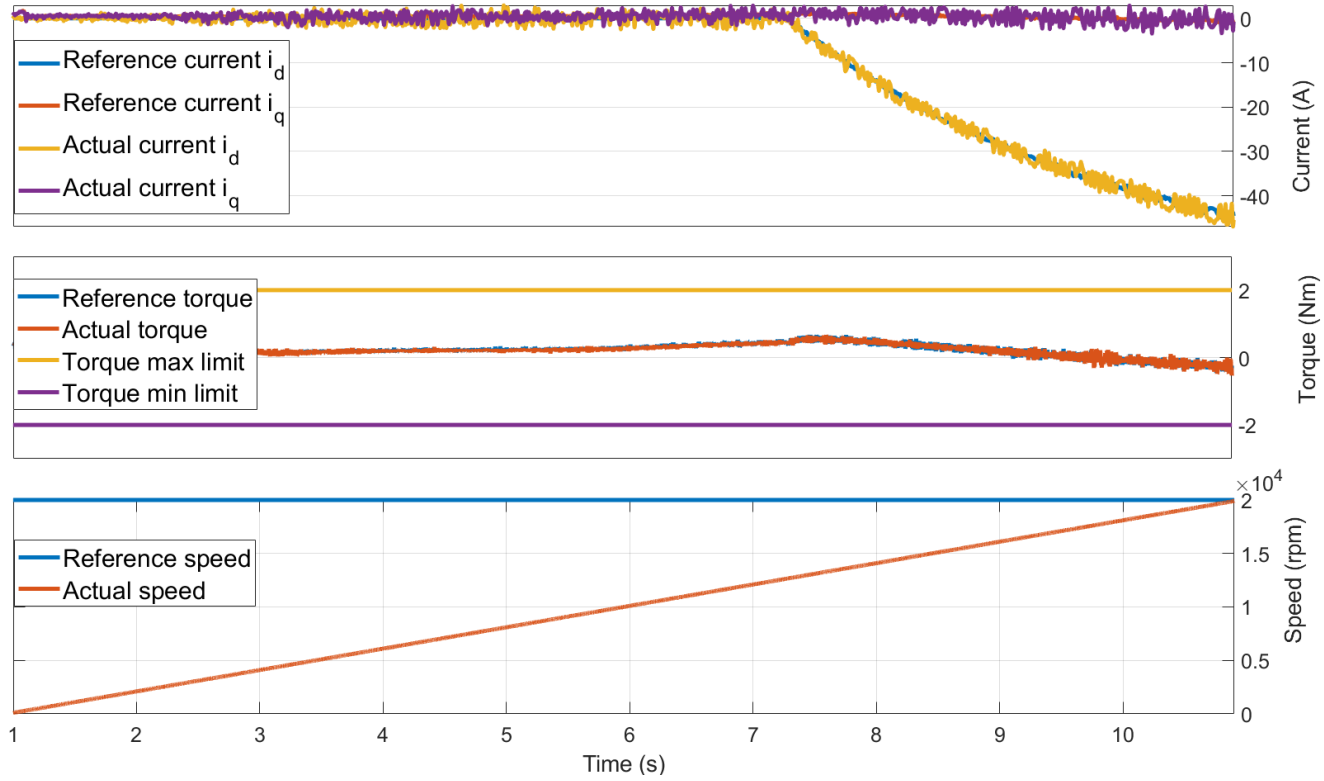


Figure 9.16: Current, torque and speed responses from running the motor from 0rpm to 20krpm.

Camilla Marie Greve Hartviksen

Improvements on the motor control of a Formula

NTNU
Norwegian University of
Science and Technology

NTNU
revolve