Drive System for an Electric Go-Kart

Estefanía Ruiz, Aitor Teran, Nicolai Fransen, Mihai Rusu,
Faheem Ahmad, Nicolás Murguizur Bustos

Energy Technology, PED2-841, 2019-05

Master's Project

Copyright © Aalborg University 2019

This report has been written using LaTeX. During the project Code Composer Studio has been used for code development and MATLAB and LTspice have been used for simulations, while a Piccolo controller from Texas Instruments has been used for the implementation of the control system. The creation of flow charts have been done using www.draw.io. Hardware schematic and PCB layout have been created using Altium Designer.

Department of Energy Technology
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Drive System for an Electric Go-Kart

Theme:
Dynamics in Electrical Energy Engineering

Project Period:
Spring Semester 2019

Project Group:
PED2-841

Participant(s):
Estefanía Ruiz
Aitor Teran
Nicolai Fransen
Mihai Rusu
Nicolás Murguizur Bustos
Faheem Ahmad

Supervisor(s):
Lajos Török
Erik Schaltz

Copies: 1

Page Numbers: 83

Date of Completion:
May 9, 2019

Abstract:

Insert abstract here...

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Contents

Preface **ix**

1 Introduction **1**

2 Problem analysis **3**
2.1 Problem statement . . . 3
2.2 Objectives . . . 3
2.3 System requirements . . . 3

3 Modeling and Design of Electric Power Train **5**
3.1 Induction machine . . . 5
3.1.1 Model of the induction machine . . . 5
3.1.2 MATLAB based validation of machine parameters . . . 6
3.2 Voltage source inverter . . . 9
3.2.1 Parallel operation of MOSFET . . . 9
3.2.2 Test results of inverter . . . 10
3.2.3 Test with signal generator . . . 12
3.2.4 Test with DSP . . . 12
3.3 Battery pack . . . 12
3.3.1 Obtaining parameters . . . 12
3.3.2 Model of the batteries . . . 14
3.3.3 Test results . . . 15
3.3.4 Conclusions . . . 19
3.4 Interface board . . . 20
3.4.1 PCB design . . . 20
3.4.2 PCB building . . . 20

4 Control of the induction machine **21**
4.1 Modulation techniques . . . 21
4.1.1 Sinusoidal PWM . . . 21
4.1.2 Third harmonic injection PWM . . . 22

4.1.3 Space vector PWM . . . 23
4.2 Selection of modulation technique . . . 25
4.3 Implementation of SVPWM . . . 25
4.3.1 Simulation results . . . 31
4.4 Field oriented control . . . 34
4.4.1 Indirect field oriented control . . . 35
4.4.2 Principle of rotor flux oriented control . . . 35
4.5 Implementation of indirect rotor flux oriented FOC . . . 40
4.5.1 Design of PI controllers in continuous domain . . . 40
4.5.2 Discretization of the controller . . . 46
4.6 Verification of controller . . . 46
4.6.1 Simulink model . . . 46
4.6.2 Simulation results . . . 46
4.7 Regenerative braking implementation . . . 46
4.7.1 Simulation results . . . 46
4.8 Regenerative Braking . . . 46
4.8.1 Types of regenerative braking . . . 46
4.9 Test results of the complete system in test bench . . . 47

5 Firmware design, implementation and testing 49
5.1 High level design . . . 49
5.1.1 Timing considerations . . . 52
5.2 System block . . . 53
5.2.1 System manager . . . 54
5.2.2 Scheduler . . . 54
5.2.3 High priority tasks . . . 56
5.3 User interface . . . 57
5.3.1 Graphical User Interface . . . 58
5.3.2 Analog acquisition manager . . . 59
5.3.3 Reference handler . . . 62
5.4 Motor control . . . 63
5.4.1 Position estimator . . . 64
5.4.2 Closed-loop control manager . . . 65
5.5 Safety . . . 66
5.5.1 Error manager . . . 66
5.5.2 Watchdog timer . . . 67
5.6 Validation . . . 67
5.6.1 Task Timing . . . 67
5.6.2 Debugging framework . . . 68

6 Discussion 73
6.1 Integration test in Go-kart platform . . . 73
6.2 Project management . . . 73
6.3 Problems and limitations . . . 73
6.3.1 Driver damage . . . 73
6.4 Future work . . . 73

7 Conclusion **75**
7.0.1 Key learning from this project 75

Bibliography **77**

A Space Vector Modulation **79**

B Template **81**
B.1 How Does Sections, Subsections, and Subsections Look? 82
B.1.1 This is a Subsection 82

Preface

This document describes the report of the group project "Drive System for an Electric Go-Kart" It has been developed from the 10th of February to the 31st of may of 2019, at Aalborg University, Institute of Energy, by the group PED-841. This report discusses the procedure and results of the design and implementation of the control of an inverter to drive an induction machine for electrical mobility. The literature references are shown in square brackets, with a number referring to a specific document which can be found in the bibliography. If the reference is after the dot, it means that it refers to the whole previous paragraph. Pictures and tables will be denoted in the X,Y format, with X representing the chapter and Y the figure or table number. The process and development has been based on the Problem Based Learning (PBL) method.

Aalborg University, May 9, 2019

Estefanía Ruiz
eruiza18@student.aau.dk

Aitor Teran
ateran18@student.aau.dk

Nicolai Fransen
nfrans18@student.aau.dk

Mihai Rusu
mrusu18@student.aau.dk

Faheem Ahmad
fahmad18@student.aau.dk

Nicolás Murguizur Bustos
nmurgu18@student.aau.dk

Nomenclature

Abbreviations:

ADC	Analog to Digital Converter
DSP	Digital Signal Processor
IM	Induction Machine
EOC	End of Conversion
FOC	Field Oriented Control
PWM	Pulse-Width Modulation
SPWM	Sinusoidal Pulse-Width Modulation
SVPWM	Space Vector Pulse-Width Modulation
THD	Total Harmonic Distortion
THIPWM	Third Harmonic Injection Pulse-Width Modulation
VSI	Voltage Source Inverter
SWC	Software Component
TCB	Task Control Block
UML	Unified modeling language
GUI	Graphical User Interface
FSM	Finite State Machine
CAN	Controller Area Network
UART	Universal Asynchronous Receiver Transmitter

Symbols:

f	Fundamental frequency
f_s	Switching frequency
k	Sector number
s	Slip
T_e	Electromechanical torque
T_{load}	Load torque
T_s	Switching period
V_{DC}	DC link voltage
$\vec{V}_k$	Active space vector
$\vec{V}_r$	Reference vector
ω	Angular velocity

1 Introduction

2 Problem analysis

2.1 Problem statement

2.2 Objectives

2.3 System requirements

3 Modeling and Design of Electric Power Train

Include all relevant parameters in each subsection

Add intro. NM

3.1 Induction machine

Induction machine nominal parameters		
Power	P_n	5.3 [kW]
Peak power (for max. 60 min)	P_{max}	7.3 [kW]
Phase voltage	V_n	13.85 [V]
Phase current	I_n	189 [A]
Power factor	PF	0.76
Frequency	f	58 [Hz]
Mechanical shaft speed	ω_n	1685 [rpm]
Electromagnetic torque	T_n	30.04 [Nm]
Induction machine constant parameters		
Magnetizing inductance	L_m	0.38 [mH]
Stator leakage inductance	L_{ls}	31.16 [μH]
Rotor leakage inductance	L_{lr}	31.16 [μH]
Stator resistance	R_s	2.5 [mΩ]
Rotor resistance	R_r	2.69 [mΩ]
Rotor moment of inertia	J	0.0151 [Nm^2]
Pole pairs	p	2

Table 3.1: Induction machine's parameters.

3.1.1 Model of the induction machine

Reference frame transformation is one of the most powerful technique invented in mathematics. It is used extensively to simplify 3 phase systems to simpler orthogonal 2 phase systems. Analysis of induction machine in 2 phase system requires

to write the stator and rotor voltage equations in $\alpha\beta$ reference frame as shown below in equation 3.1-3.4. Equation 3.1 represents stator voltage on α axis which is dependent on stator resistance (R_s), stator α axis current ($i_{\alpha s}$), number of pole pair (p) and stator α axis flux linkage ($\lambda_{\alpha s}$).

Similarly further equation variables can be identified, where subscript α or β represents the axis and subscript s or r represents stator or rotor. With ω_θ being the electrical rotating speed while ω_r is the shaft angular speed. Equation 3.5 represents the electromagnetic torque generated by the motor.

Motor equations

$$u_{\alpha s} = R_s i_{\alpha s} + p\lambda_{\alpha s} \tag{3.1}$$

$$u_{\beta s} = R_s i_{\beta s} + p\lambda_{\beta s} + \omega_\theta \lambda_{\alpha s} \tag{3.2}$$

$$u_{\alpha r} = R_r i_{\alpha r} + p\lambda_{\alpha r} - (\omega_\theta - \omega_r)\lambda_{\beta r} \tag{3.3}$$

$$u_{\beta r} = R_r i_{\beta r} + p\lambda_{\beta r} + (\omega_\theta - \omega_r)\lambda_{\alpha r} \tag{3.4}$$

Torque equation

$$\tau = \frac{3}{2} p L_m (i_{\beta s} \cdot i_{\alpha r} - i_{\alpha s} \cdot i_{\beta r}) \tag{3.5}$$

3.1.2 MATLAB based validation of machine parameters

Simulink model

Simulink model will be included as appendix, leaving here only equations and results.

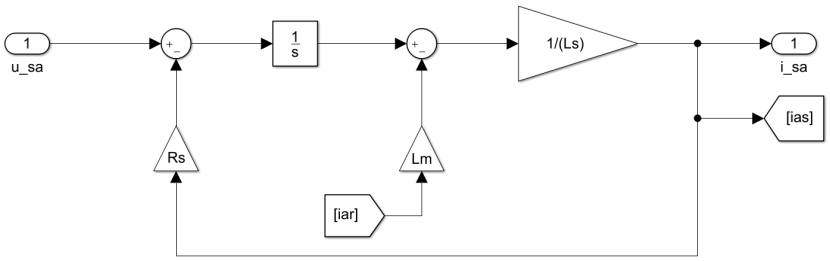


Figure 3.1: Simulink implementation of α stator voltage equation

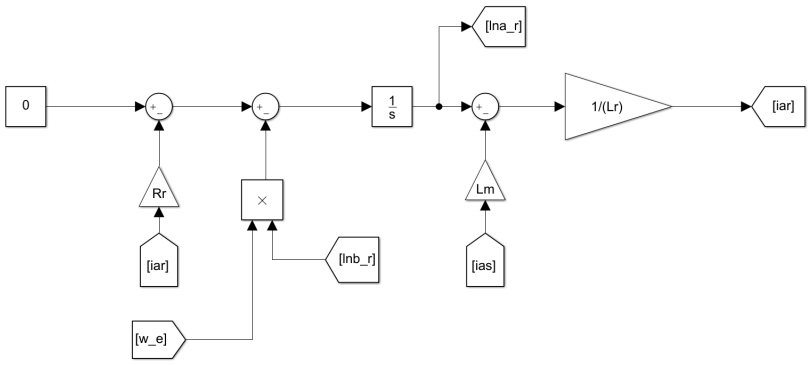


Figure 3.2: Simulink implementation of α - rotor voltage equation

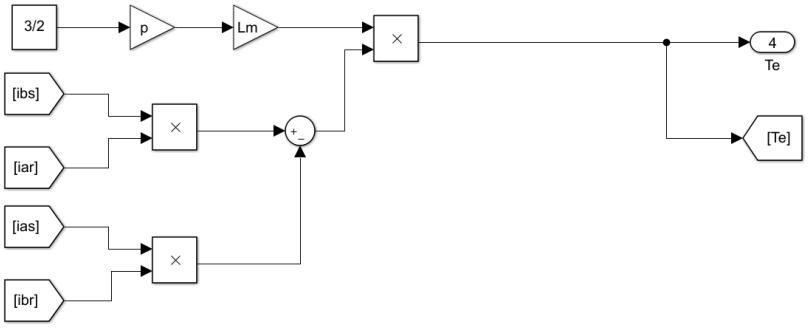


Figure 3.3: Simulink implementation of $\alpha\beta$ torque equation

Simulation results

No-load simulation

Nominal conditions simulation

Power factor validation

Using the Induction machine stator and rotor winding parameters and running under rated stator phase voltage yields a power factor of 0.75. In figure 3.4 below the phase A voltage and current is used to showcase the PF calculation.

We will be adding some calculations of PF on the image below

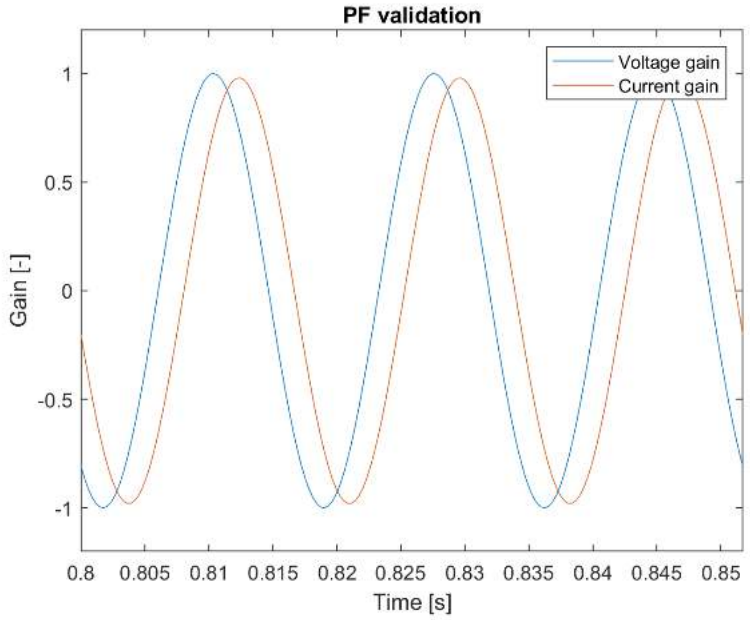


Figure 3.4: Simulation of machine power factor

Slip calculation

$$s = \frac{n_s - n_n}{n_s} \cdot 100 = \frac{1740 - 1681}{1740} \cdot 100 = 3.16\% \tag{3.6}$$

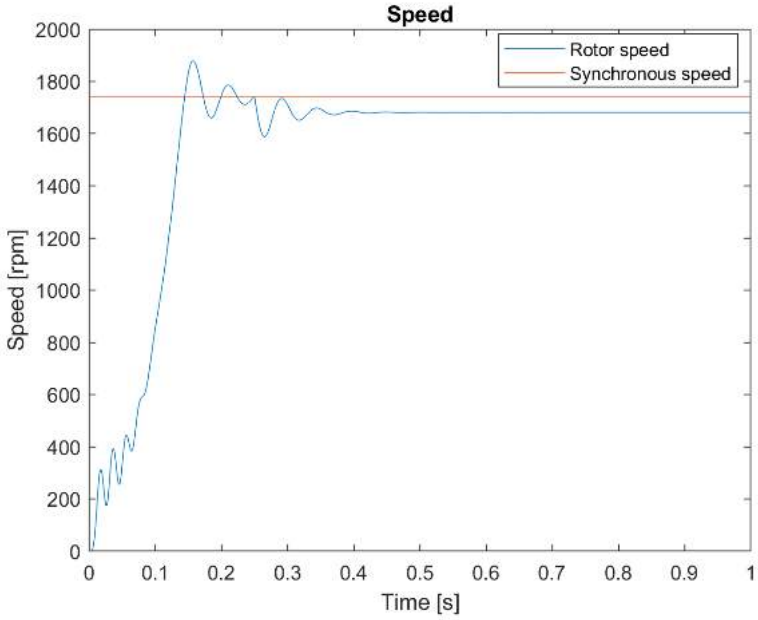


Figure 3.5: Simulation of machine slip

3.2 Voltage source inverter

The 3 phase (3ϕ) full-bridge voltage source inverter (VSI) is one of the most critical component of our system. The inverter is responsible for taking the command instruction from the logic processing unit and drive the motor accordingly. This 3ϕ inverter allows the full power to be delivered to motor during motoring operation and also provides the path for kinetic energy to rush back to the battery system during regenerative braking of Go-Cart.

Given the notorious behavior of semiconductor devices and their proneness to failure [1], [2] in a power electronics system, intensive care is needed to be taken while designing this inverter. Luckily for us, this heavy lifting have been taken care of by the esteemed group of year 2018 [3]. Who went to painstaking length in design process to ensure the normal functioning of inverter under rated as well as instantaneous peak operating conditions and provided us the opportunity to focus primarily on the control algorithm and implementation on DSP.

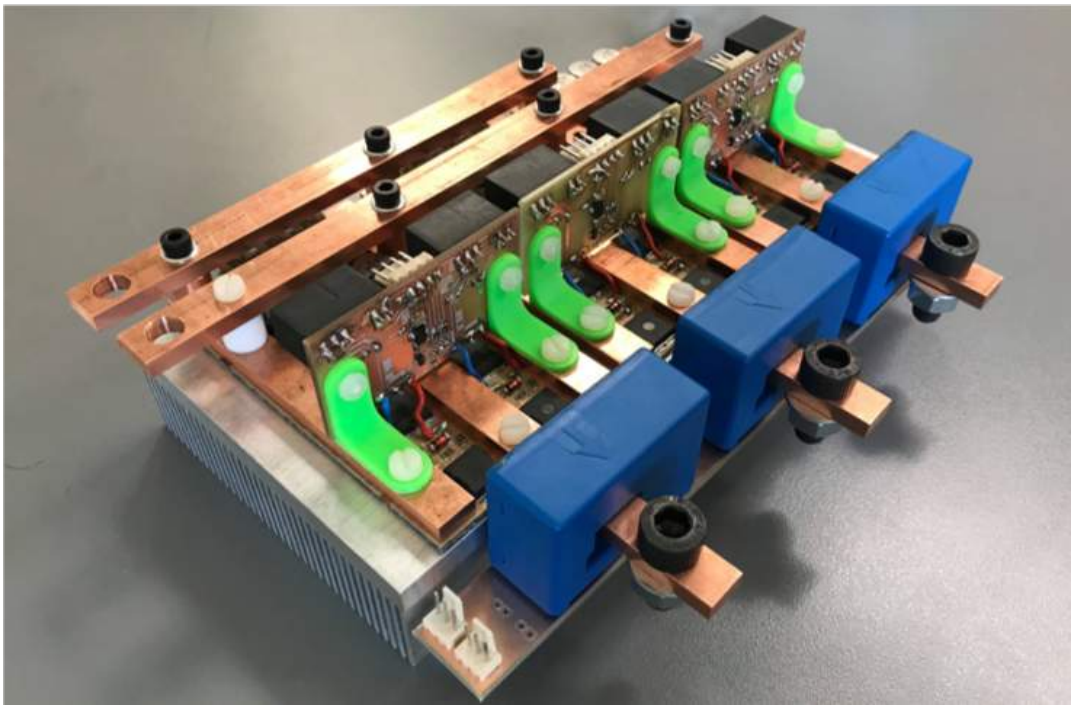


Figure 3.6: 3ϕ Inverter developed by 2018 group

3.2.1 Parallel operation of MOSFET

Based on motor electrical characteristics mentioned in section 3.1, each leg of the inverter needs to flow 263A$_{rms}$ current. This would result in 371A of peak current value. Low voltage MOSFET (<100V) with drain current capacity of above 300A is difficult to find. Hence the decision was made to parallel 2 MOSFET and have them share the load. This design decision lead to 12 MOSFETs for the 3ϕ full-bridge with each legs containing 4 MOSFETs. 2 MOSFETs in parallel for high side and low side switch respectively (figure 3.7).

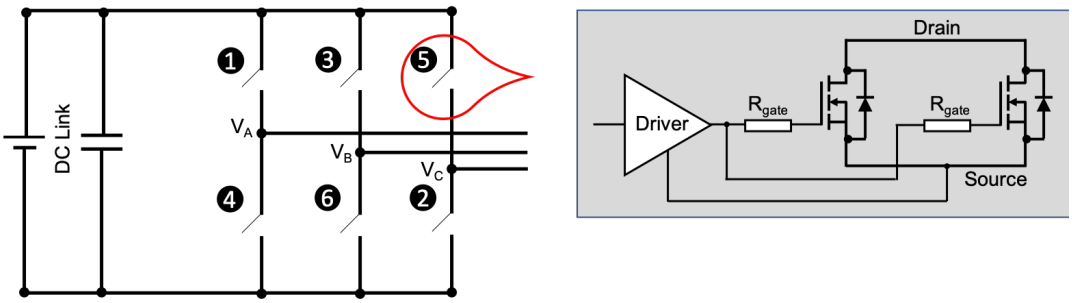
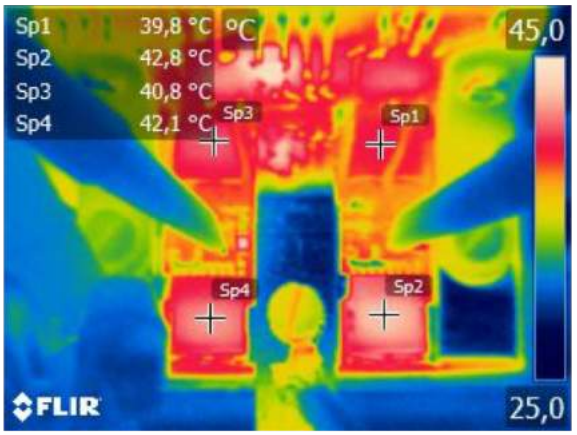


Figure 3.7: Parallel MOSFET design for load sharing

3.2.2 Test results of inverter

Even though the inverter was designed for peak operating conditions. Due to time constraints and availability of test equipment the team from 2018 only managed to test one leg of the inverter at max. 90A. At rated DC link voltage of 36V and switching frequency 20kHz each MOSFET is expected to generate 4.632W (using datasheet values) of loss which needs to be dissipated as heat. Using MOSFET loss and thermal resistance values (shown later in chapter) MOSFET's case temperature is calculated to be at 31.2°C. Whereas the test results using a thermal camera (FLIR Thermal Camera) measured the temperature of 4 MOSFETS (of one leg) from 40°C - 42°C. Difference in thermal camera measurement and simulation results keeps on increasing as operating condition is increased. Hence another measurement method had to be employed to confirm.



Operating Condition

DC Link (V)	F_{sw}	I_{avg}	I_{pk}
36V	20kHz	89.8A	92.4A

MOSFET Case Temp.

Q1	Q2	Q3	Q4	Sim.
40°C	43°C	41°C	42°C	31.2°C

Figure 3.8: Thermal Camera output at 90A

Using a thermocouple seemed the right choice as another thermal camera would bring further uncertainties as we could not cover our heat emitting surface with black paint.

Knowing the MOSFET parameters, accurate estimation of loss at specific operating conditions can be obtained [4]. By using thermal conductivity information for each of the layer between MOSFET and heat sink, temperature expected on MOSFET's case can be calculated. Figure 3.9 below shows the structure of inverter PCB with heat sink and each of the layers in between.

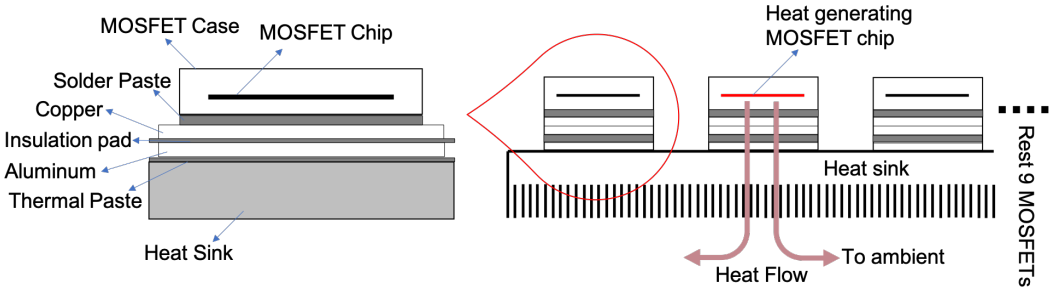


Figure 3.9: MOSFET heat dissipation via multiple intermediate layers

The heatsink selected by 2018 group is 890SP-02000-A-100 which has a thermal resistance of 0.07°C/W when used with a Papst type 3312 (12V) cooling fan. Taking into consideration the thermal resistance of the rest the of layers as showcased in figure 3.9 along with the estimated loss in MOSFET, temperature expected on the case of MOSFET can be calculated using the equation 3.7 below. The subscripts of thermal resistances in equation 3.7 stands for : JC (MOSFET Junction to case), CS (Case to Solder Paste), SC (Solder Paste to Copper), CI (Copper to Insulation pad), IA (Insulation pad to Aluminium), AP (Aluminium to Thermal paste), SA (Heat sink to air). T_J and T_a represents MOSFET junction temperature, room air temperature respectively. $P_{MOSLoss}$ represents MOSFET's power loss.

$$T_J - T_a = (R_{thJC} + R_{thCS} + R_{thSC} + R_{thCI} + R_{thIA} + R_{thAP} + R_{thSA}) \cdot P_{MOSLoss} \quad (3.7)$$

Ultimately, testing invterer leg at load current of 175A_{avg} and DC link voltage of 12V (using motor winding as RL load) at switching frequency of 10kHz. Again using the datatsheet parameters we calculate that the MOSFET is expected to generate 11.283W which translates to 44.13°C at chip junction and 39.6°C at the case using equation 3.7. Using the thermocouple in contact with MOSFET case gives us result of 43.6°C (figure 3.10), which is much closer to expected value as compared to thermal camera results of 61.0°C (figure 3.11).

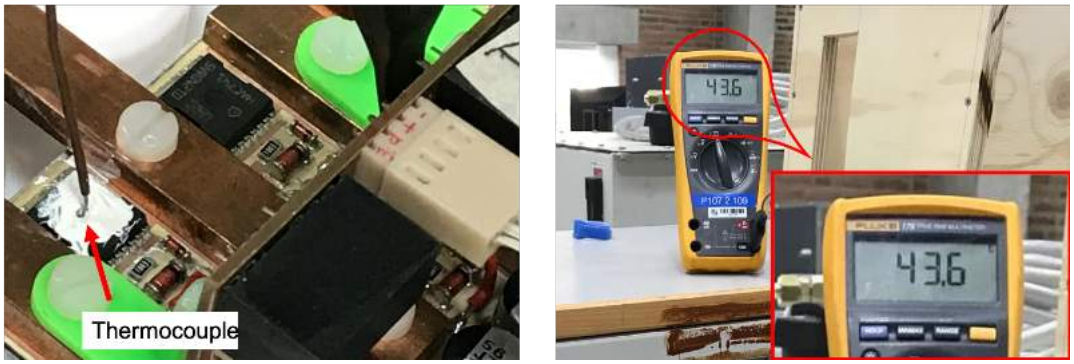


Figure 3.10: MOSFET case temp. measure using thermocouple at 175A load current

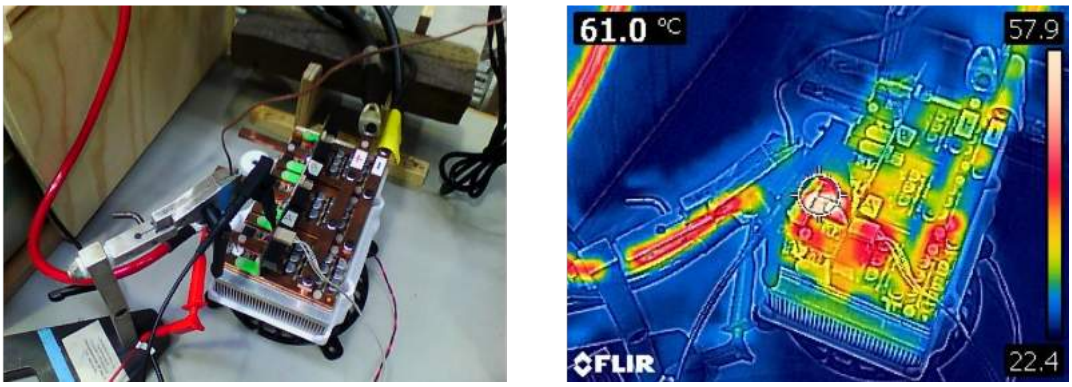


Figure 3.11: MOSFET case temp. measure using thermal camera at 175A load current

3.2.3 Test with signal generator

3.2.4 Test with DSP

3.3 Battery pack

Include introduction. AT

3.3.1 Obtaining parameters

An important part of an electric Go-Kart is the voltage supply. Since the inverter is designed for a DC supply of 36 V_{DC}, three lead-acid batteries of 12V will be used. The batteries are of the YellowTop S 4,2 type, produced by Optima.

add some other parameters that are present in datasheet like capacity etc.

Using the same batteries as the previous groups worked with, certain assumptions were made. First of all, one out of the total number of four batteries was excluded as it comes up it has the lowest capacity. Also, the other three were tested because they have not been used for a long time.

The main task of this project regarding batteries topic is to create a dynamic model of the batteries. Having a dynamic model provides advantages such as the ability to estimate how much current can be reintroduced in the battery when regenerative braking is occurring and provides the possibility to simulate how much time the batteries can be used at different current values. Dynamic model consists in obtaining the internal impedance of the batteries which can be used to validate the simulation model. The values for the components which make up the equivalent model of the batteries can be obtained by performing different set of experiments. [5]

Devices

- 3 12V YellowTop lead-acid batteries
- 1 KEPCO bi-directional power supply
- 1 National Instruments USB-6341 data acquisition device
- 1 laptop with MATLAB installed

Setup

The setup can be seen in Figure 3.12. It is placed in the student laboratory inside the University.

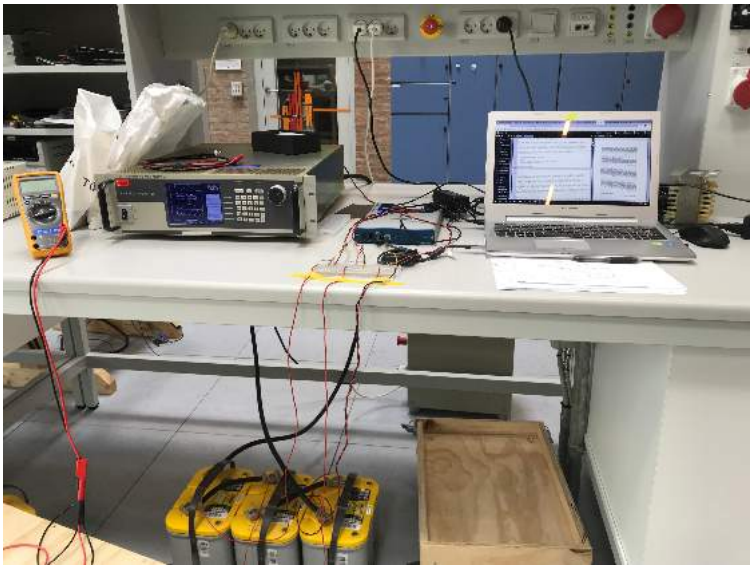


Figure 3.12: Practical setup for testing the batteries.

Procedure

The first step is to connect the batteries in series and to the power supply. Since they are drawing too much current, when the power supply is turned on, a resistor must be connected in series with the batteries to avoid over-current protection. After having everything connected, one has to verify if the batteries are fully charged and with about the same voltage at their terminals. In order to make the data acquisition of the voltages across the batteries and the output current of the power supply, the National Instruments device is used. This device has 16 analog input channels with the range of $\pm 10V$ but only four will be used. Because the voltage at the terminals of the batteries is higher than 10V, a voltage divider and a breadboard need to be used in order to scale the voltage down taking into account that every voltage is referenced to one of the batteries' terminal. The analog output of the power supply is a signal with the amplitude of 10V and because the current applied is in the same range, no scaling has to be done. Additionally, a thermocouple can be added to measure the temperature while the experiments are being done.

After having the setup ready and the batteries fully charged and balanced (small differences are accepted since the batteries are not new and they do not have exactly the same capacity) the impedance test can be done.

Obtaining the values for the internal impedance can be done by discharging the battery with 10A until 90% or 80% SoC is reached and then interrupt the current. Then they should be left to rest for at least half an hour so that the chemical reactions inside them settle down and they return to their open circuit voltage. The same procedure should be repeated until 0% is reached. By monitoring data

during discharge, parameters such as the time constant or the voltage drop across the internal resistance can be obtained. The same procedure can be repeated while charging. It is indicated that different current pulses are applied so that it can observe how the internal resistance varies with different values of the current. There will also be some small differences for measuring at different temperatures but this is not in the scope of the experiment. The values at which the bi-directional voltage supply will be set are 43.2V ($3 \cdot 14.4V$)and 31.5V ($3 \cdot 10.5V$) for maximum and minimum voltage values across the batteries. The capacity test of the batteries can be done by doing a complete charge-discharge cycle and monitoring the current and the duration of the cycles.

The characteristic parameters can be obtained experimentally. From the vertical jump of the voltage across the battery in the moment when the current is interrupted, the series resistor Rs' can be computed. The time constant is determined by selecting points from the voltage recovery curve in the region after the current was interrupted and analyzing it as a basic RC circuit. The transient resistance Rt can be obtained by measuring the open circuit voltage at the end of recovery region and subtracting Rs'.[6]

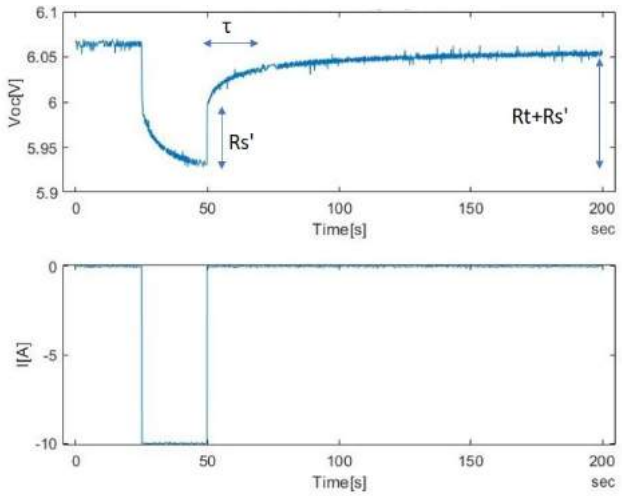


Figure 3.13: Obtaining values of equivalent circuit.

3.3.2 Model of the batteries

Why controllable

A battery can be modeled by its internal circuit. This is composed by a controllable voltage source an internal resistance and RC parallel blocks. All of the parameters are dependent on the current flowing, the SOC and the temperature. Increasing the number of RC parallel blocks will lead to a better approximation of the recovery region. It is assumed that the equivalent model of the batteries contains only one pair of resistor connected in parallel with a capacitor which will lead to an error but the result is close to reality.[5]

Model of internal structure

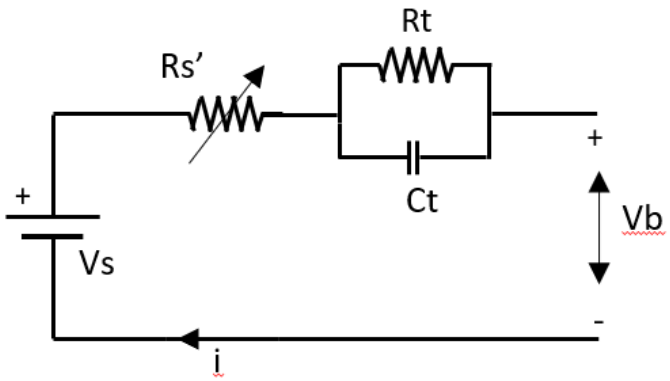


Figure 3.14: Equivalent model of internal structure

Remove red underline in Vb and i

Simulation model

This model is adapted after the one suggested in [6].

This figure is very small, NHF. Tried to make it bigger, still not very legible. AT.

The output of the figure needs to be clear and from here not much is understood. AT

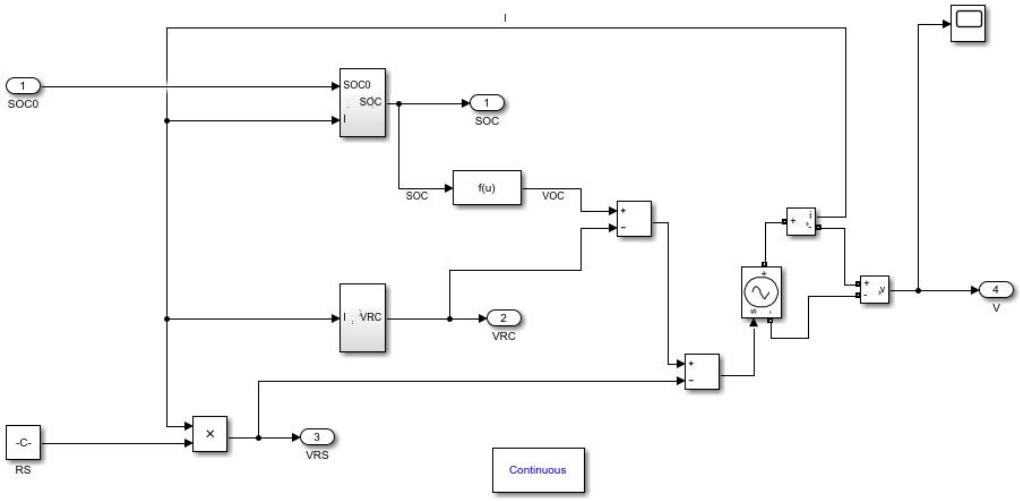


Figure 3.15: Simulation model in Simulink

3.3.3 Test results

Two different types of tests were done in this topic.

If an introduction is added it needs to contain a bit more information about the tests, maybe what is the purpose or the output of testing. AT

Charge-discharge test

This test was conducted because the capacity of the batteries needs to be obtained. It is because the batteries are old and due to some preliminary tests their capacity does not fit with the measurements done last year [3]. This subsection provides test results for only one of the batteries and the other results are presented in the Appendix.

link to actual appendix

Battery D

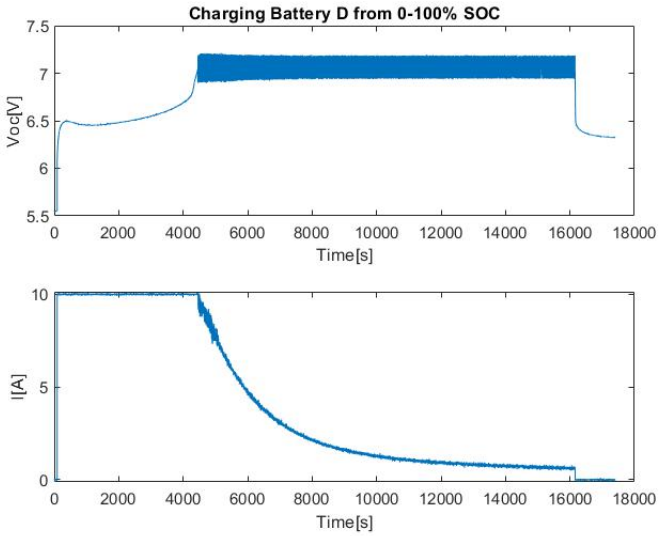


Figure 3.16: Charging battery D 0-100% SOC

From Figure 3.16 it can be seen that it takes around 16100 seconds to fully charge the battery at a constant current of 10A. Therefore, the charging capacity of battery D is 44.7Ahr

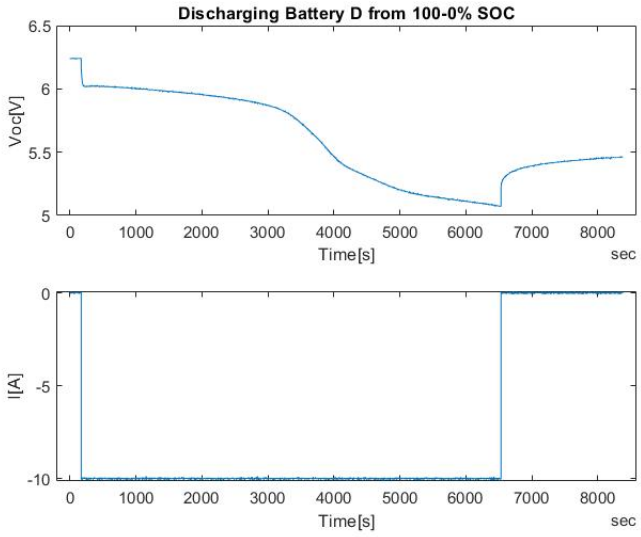


Figure 3.17: Charging battery D 100-0% SOC

On the other hand, it takes only 6363 seconds to discharge the battery with a constant current of 10A. Therefore, the discharging capacity is 17.675Ahr.

Partial discharge test

This test provides the curves needed to compute the values of the equivalent circuit components. The first experiments were done on the batteries connected in series since this is the configuration that will be used in the Go-Kart but because there is a noticeable difference in their capacity, one of the batteries discharged faster then the other two during the test. In order to avoid damaging the battery, the test was stopped. The results of this test are presented in the Appendix. The internal values of the components were then obtained by applying short current pulses (approx. 25 seconds) so that the SOC of the batteries was not highly modified, on each battery separately with the following results:

link to appendix

Applying pulse discharge to batteries @ 100% SOC

I think you should put the 3 voltage graphs in 1 plot, NHF

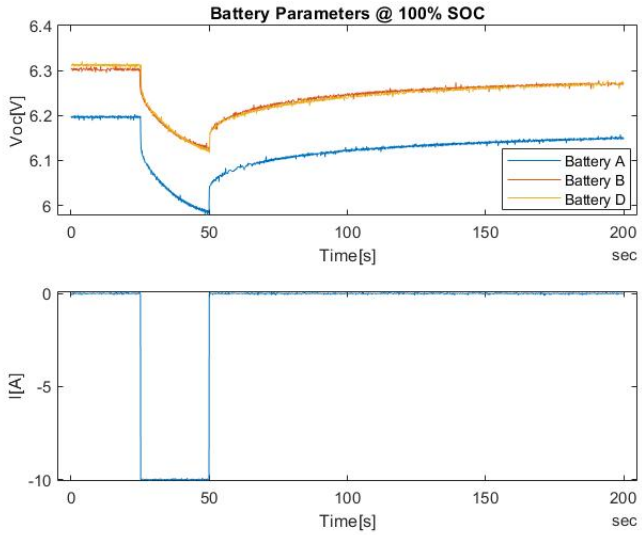


Figure 3.18: Pulse discharge @ 100% SOC

From experiments shown in Figure 3.18 the following values are calculated:

- $Rs'_A = 0.0059\Omega, Rt_A = 0.0108\Omega$
- $Rs'_B = 0.0034\Omega, Rt_B = 0.0108\Omega$
- $Rs'_D = 0.0025\Omega, Rt_D = 0.0116\Omega$

Also, it can be seen that the time constant of the battery is $\approx 10 seconds$.

Series connection pulse discharge

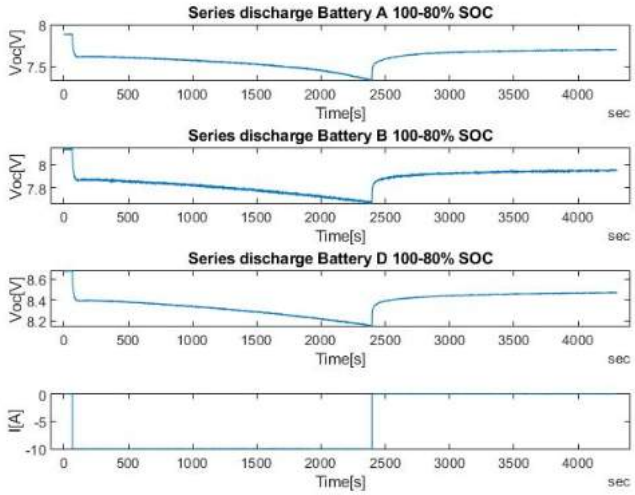


Figure 3.19: Internal impedance from series test for batteries @ 100-80% SOC

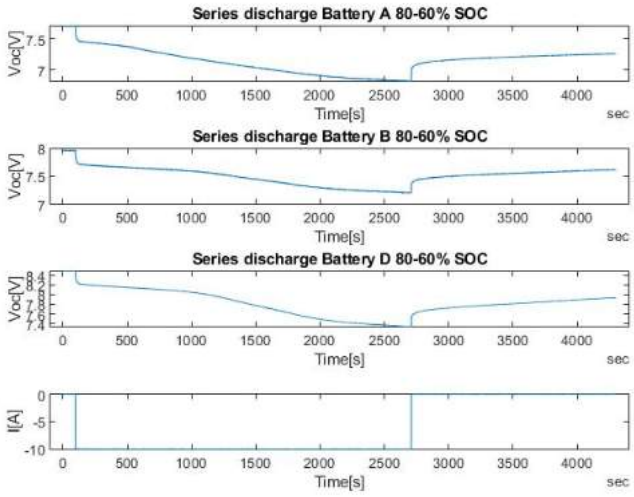


Figure 3.20: Internal impedance from series test for batteries @ 80-60% SOC

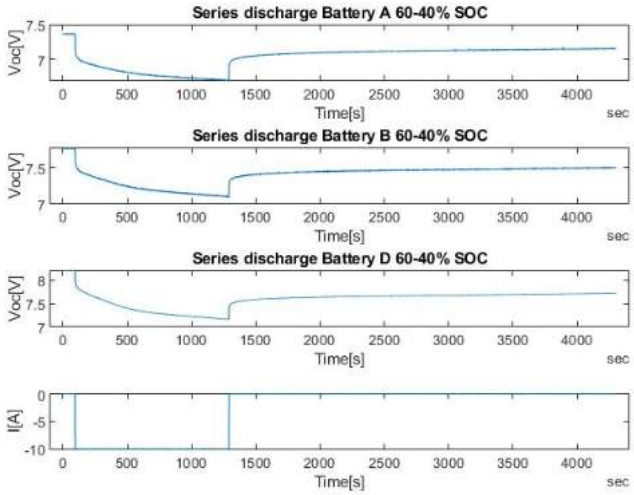


Figure 3.21: Internal impedance from series test for batteries @ 60-40% SOC

I would try to reduce the number of pictures and add some more informative text. AT

3.3.4 Conclusions

Because the batteries are not new there is a noticeable difference in their capacity and internal resistance. Being used in a series configuration, the available capacity will be the lowest one which is around 28Ahr. The application they are supposed to be used in implies drawing high currents $> 150A$ and this will lead to fast discharge ($\approx 11 minutes$). Also because the difference in capacity the charging

time is not the same and this will lead to different states of charge while they are supposed to be the same.

There exists some sources of errors. This can be both hardware and software such as errors in data acquisition(this being done at 1 sample/seconds because of the big amount of time requested for a complete charge-discharge cycle), inequalities in the ratio of the voltage divider, not selecting the points on the result curves at the same reference.

This is not a good place to have a conclusion, this should be placed in the final conclusion, NHF

I think this text is nicely placed here but the title conclusions should be changed.

3.4 Interface board

3.4.1 PCB design

3.4.2 PCB building

4 Control of the induction machine

I think this chapter will be very long, if it should include both modulation techniques and controller design. Maybe consider diving in two chapters, Erik

4.1 Modulation techniques

A wide variety of modulation techniques can be implemented for controlling the output voltage of power converters. The objective of the modulation is to control the switching of the inverter in order to obtain the desired magnitude and frequency at the AC output voltage, which has to be close to a sine wave [7]. Pulse-width modulation (PWM) generates the pulses for the inverter switches by comparing a reference voltage waveform with a triangular voltage waveform of higher frequency. In this section, three different pulse-width modulation (PWM) methods are introduced and their advantages and disadvantages are analyzed to select the most adequate modulation for this project. Sinusoidal PWM (SPWM), third-harmonic injection PWM (THIPWM) and space vector PWM (SVPWM) are the most commonly used modulation techniques for controlling three phase voltage source inverters (VSI)[8].

FOR ERIK: We have been thinking about how to split the chapter, for now we have included regenerative braking inside control and one option would be to split bw pure control theory and simulations/tests. However, we believe it also makes sense to have it all in one chapter in order to have tests right after explaining what each thing is. What do you think about this approach? If you think it's better to split it in two, how do you suggest to do this changes?

4.1.1 Sinusoidal PWM

Add intro. NM

Sinusoidal PWM generates the gate pulses for each of the inverter's legs by comparing a sinusoidal reference signal with a triangular signal (carrier). In the case of three phase VSI, the reference signal consists of three sinusoidal phase voltages each shifted by 120°C and at the desired fundamental frequency. The carrier signal is a triangular waveform which establishes the switching frequency of the VSI [8]. The switching of the semiconductor devices in each of the inverter legs is determined by the crossing points of these two waveforms. Thus, when the amplitude of the reference signal is greater than the amplitude of the carrier signal a high pulse is generated and the upper switch of the corresponding leg is turned on. The lower switch operation is complementary to the upper switch and hence it is turned off. On the other hand, when the reference is lower than the carrier, a low pulse is generated and the upper switch is turn off. Figure 4.1 shows the operation of the SPWM for one of the inverter's legs.

I think that a final sentence explaining that this is not exactly how it was implemented in hardware (as the voltages are not compared in the PWM HW) would be nice to explain the relationship between the theory and implementation. NM

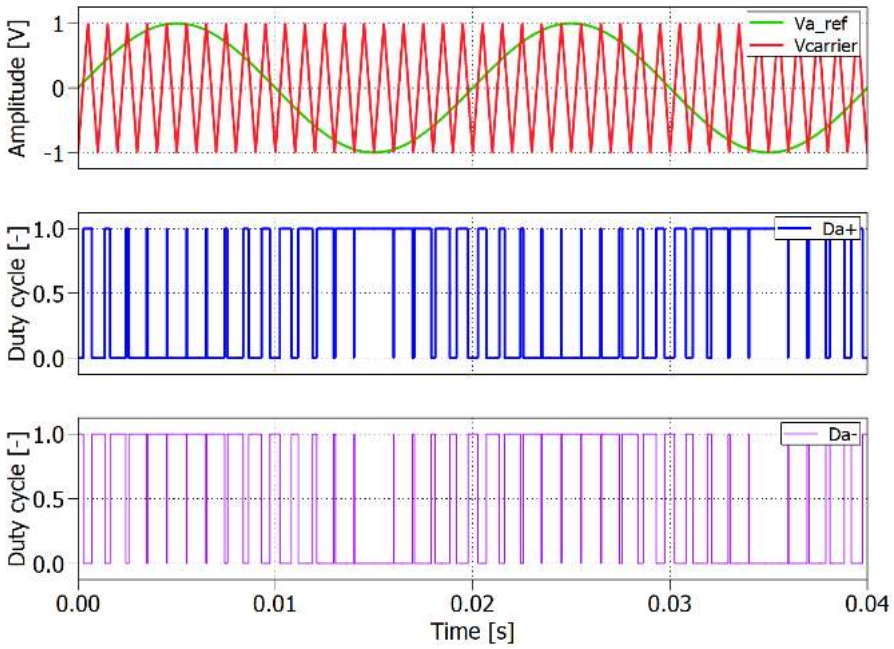


Figure 4.1: Sinusoidal PWM operation.

4.1.2 Third harmonic injection PWM

The third-harmonic PWM follows the same operation as SPWM of comparing a reference signal with a triangular signal to generate the gate pulses for the inverter. The difference is that the sinusoidal phase voltage reference is modified by adding the third harmonic component resulting in a flattened reference waveform shown in figure 4.2.

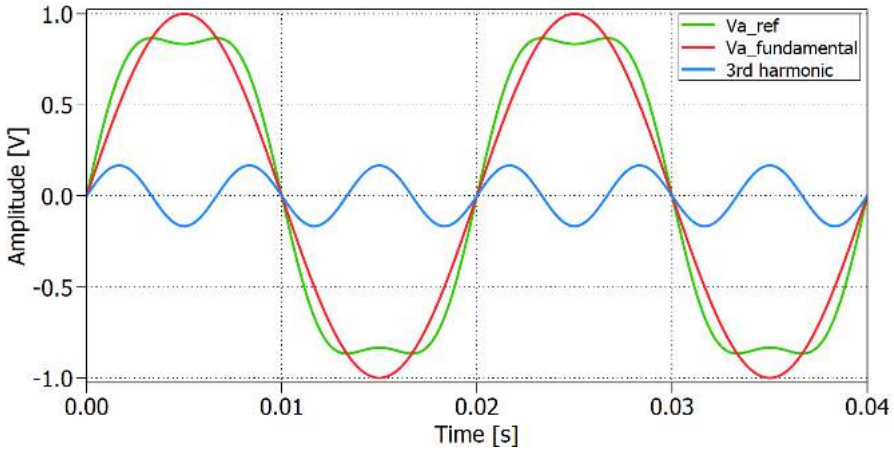


Figure 4.2: Third harmonic injection resultant reference waveform.

The third-harmonic component is equally added to each of the phase voltages, therefore, the line to line voltage remains undisturbed due to cancellation of the

third harmonic. As a result of the third-harmonic injection, the peak value of the output voltage is decreased and at the same time keeping the amplitude of the fundamental unchanged. This way it is possible to increase the inverter's output voltage up to approximately 15% higher than with SPWM without causing harmonic distortion of the line-to-line voltage [8].The reference waveform in THIPWM has lower amplitude than SPWM allowing a better utilisation of the DC voltage.

4.1.3 Space vector PWM

Space vector modulation is a modulation technique that differs from the two previous PWM methods because it does not modulate each of the three phase voltages independently. SVPWM considers the inverter as a single unit by representing the three modulating voltages as vectors rotating in the counterclockwise direction in a two-dimensional ($\alpha\beta$) reference plane [8]. Using this modulation technique, as with THIPWM, it is possible to increase the inverter's output voltage over that obtained using SPWM. This is because it reduces the peak amplitude of the modulating reference waveform, used for comparison with carrier waveform, resulting in a similar shape as with THIPWM (see figure 4.3). However, the implementation is completely different, and the advantages and disadvantages of these three methods will be discussed in the next section.

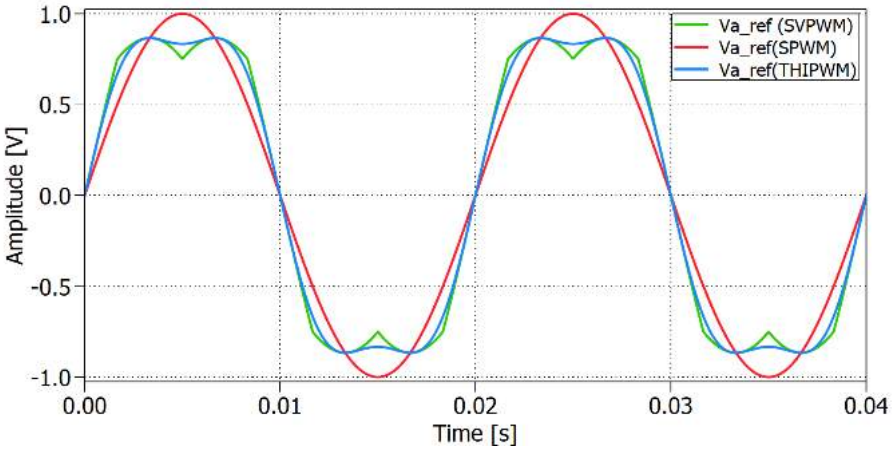


Figure 4.3: Space vector PWM resultant reference waveform

A VSI is controlled by six switches which allow the inverter to operate in eight different configurations according to the conduction states of the switches. Figure 4.4 shows the 8 possible configurations where '1' is used to represent the positive phase voltage level and '0' the negative. Therefore, each configuration can be represented as binary codes and has associated its corresponding space vector [8].

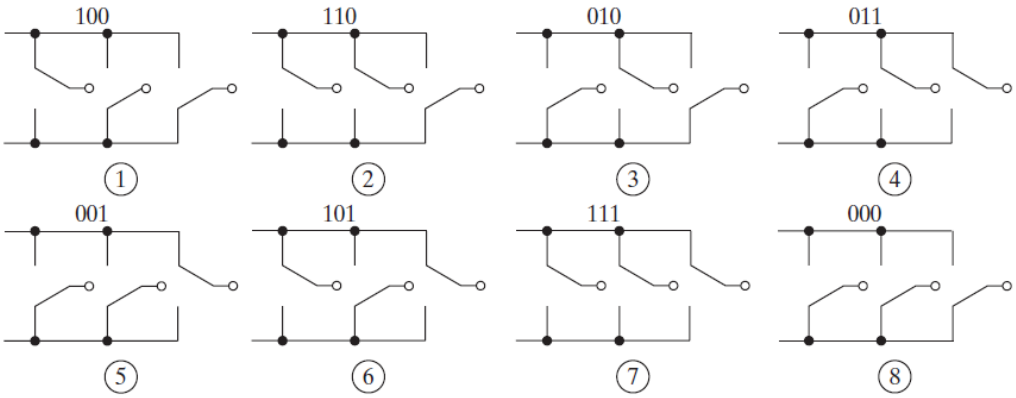


Figure 4.4: Switching states of a three phase voltage source inverter [8].

The space vector representation is sketched in figure 4.5, where vectors $V_1 - V_6$ are called active vectors and are phase shifted by $\pi/3$. These active vectors have fixed magnitude ($2/3 \cdot V_{DC}$) and form an hexagon [8]. Thus, the space vector can be located in any of the six sectors which is bounded by two active vectors. When the three upper or the three lower switches are conducting simultaneously, the DC supply line is short circuited resulting in the zero vectors (V_0 and V_7). These zero vector are at the origin of the hexagon and have zero amplitude. The eight space vectors are seen as stationary vectors because they do not rotate [8].

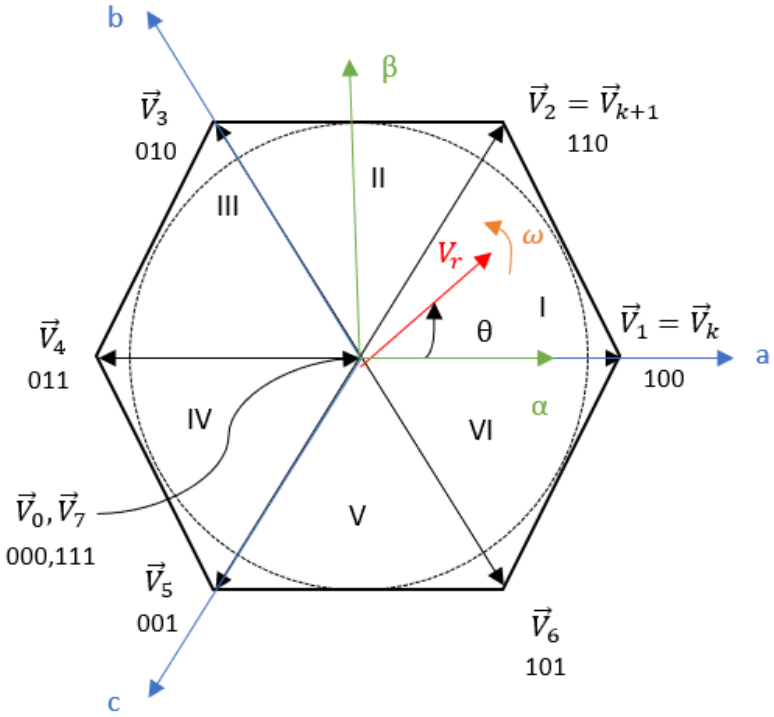


Figure 4.5: Space vector representation with reference voltage vector located in sector I.

Vr vector is not at origin (no big deal) AT

SVPWM operation is based on the rotation of a reference voltage vector $\vec{V}_r$ in the anticlockwise direction in a $\alpha\beta$ reference plane to generate the desired three phase voltages at the output of the VSI. The magnitude of V_r depends on the magnitude of the output voltage and the frequency of rotation is the same as the fundamental frequency [8]. Any space vector can be defined, according to its sector location, by the two adjacent active vectors ($\vec{V}_k$ and $\vec{V_{k+1}}$) and the zero vectors ($\vec{V}_0$ and $\vec{V}_7$). This way the switching sequence is determined to obtain the reference voltage waveform which will be compared with a triangular waveform in order to generate the pulses for the inverter's switches.

4.2 Selection of modulation technique

The three previous discussed modulation techniques are the most commonly used methods for controlling three phase voltage source inverters. SPWM presents the simplest implementation and thus is the most widely used for voltage control. However, this modulation technique has the disadvantage that the inverter's output voltage cannot exceed the DC supply voltage without entering the overmodulation region [8]. Overmodulation is reached when the amplitude of the reference voltage waveform is higher than the carrier's waveform amplitude which means that the relationship between the fundamental reference voltage component and the DC supply voltage becomes non-linear [8]. The overmodulation region is avoided in most applications that require low distortion because it adds more harmonics to the system.

THIPWM is used to compensate for this drawback as it adds a third harmonic component to the reference voltage waveform allowing an increase of approximately up to 15% of the DC supply voltage. This modulation technique is preferred in three phase applications due to cancellation of the third harmonic component leading to lower THD than with SPWM [8].

In contrast with SPWM and THIPWM, space vector PWM considers the inverter as a single unit and it requires reference frame transformations and, thereby, more complicated mathematical operations which lead to higher computational requirements. Nevertheless, SVPWM has the advantage of lower harmonic distortion than SPWM in both output voltages and currents applied to the induction machine [9]. Similar to THIPWM, this technique allows up to 15% increase in the output voltage compared with the conventional SPWM [8]. This way the linear operation region can be extended allowing the VSI to operate with higher voltages and, consequently, lower currents [9]. In addition to the advantage of lower THD and more efficient use of the DC link voltage, SVPWM can be implemented easily with DSP-based control systems [8][9]. Therefore, SVPWM is the modulation technique selected for this project due to its flexibility to be implemented digitally and because it presents lower THD than other modulation schemes.

4.3 Implementation of SVPWM

The space vector modulation scheme is implemented in this section to obtain the gate signals for the VSI. Figure 4.6 shows a block diagram with the subsystems

used for validating the modulation scheme in Simulink/Matlab. Three phase sinusoidal voltages are transformed from abc reference frame to $\alpha\beta$ frame by using Clarke's transformation. The $\alpha\beta$ voltages are the input for the SVPWM subsystem which will output the duty cycles for the inverter, which is connected to the induction machine.

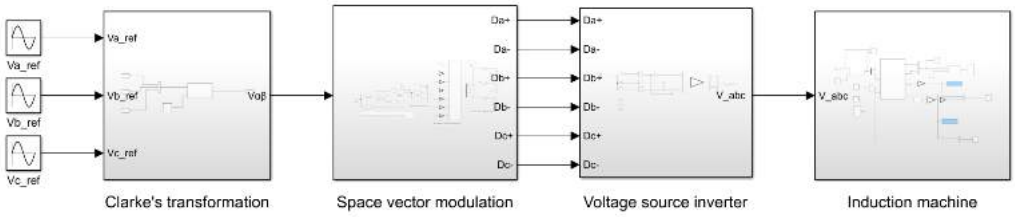


Figure 4.6: Block diagram for SVPWM implementation.

The implementation of SVPWM is shown in figure 4.7 and it is carried out by following the next steps [8]:

1. Clarke's transformation for the three phase sinusoidal reference signals in order to obtain the components of the reference vector in a two coordinate $\alpha\beta$ stationary reference frame.

2. Calculation of the magnitude V_r and the angle θ, with respect to the phase a axis, of the reference vector.

3. Determination of the sector number ($k = 1, 2, ..., 6$), in which the reference vector lies, and the angle γ with respect to the adjacent active vector ($\vec{V}_k$) of the corresponding sector.

4. Calculation of the time intervals for which the active vectors ($\vec{V}_k$ and $\vec{V_{k+1}}$) and the zero vectors ($\vec{V}_0$ and $\vec{V}_7$) are required to be on during one pulse-width modulation period.

5. Determination of the switching sequence to obtain the reference signal that will be compared with the triangular wave to generate the corresponding gate signals for the VSI.

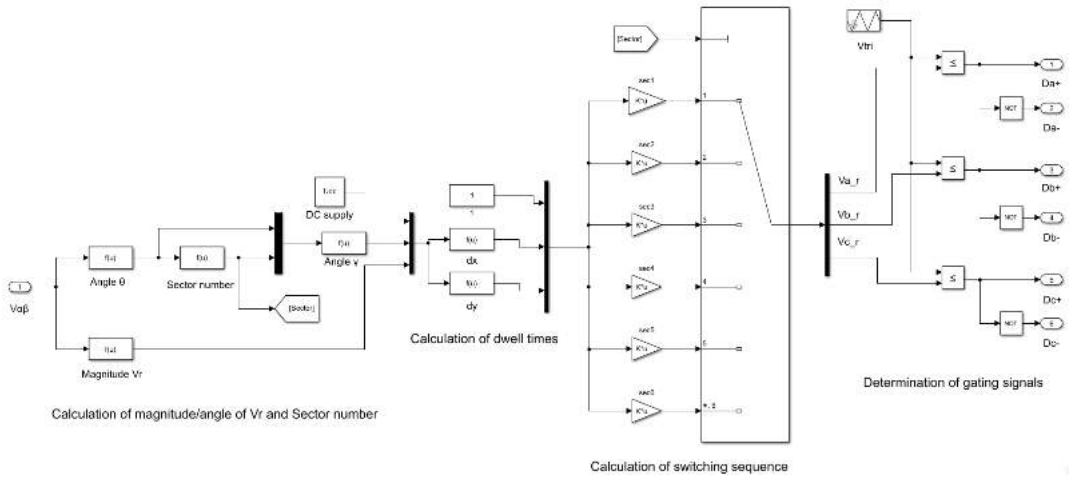


Figure 4.7: Subsystem SVPWM showing the steps followed for the implementation of the modulation scheme.

Clarke's transformation

The first step in the SVPWM implementation is the transformation of the three phase reference voltages from the abc reference frame to a two-dimensional reference frame. These two phase variables, which are stationary and orthogonal, are denoted as α corresponding to the real axis (aligned with phase a) and β for the imaginary axis. Therefore, the reference vector $\vec{V_r}$ is defined as:

$$\vec{V_r} = V_\alpha + jV_\beta = \frac{2}{3} \cdot \left[V_a \cdot e^{j0} + V_b \cdot e^{j\frac{2\pi}{3}} + V_c \cdot e^{-j\frac{2\pi}{3}} \right] \tag{4.1}$$

Applying Euler's formula (equation 4.2) to equation 4.1 and equating real and imaginary terms, the Clarke's transform is defined as in equation 4.3.

$$e^{j\theta} = cos\theta + jsin\theta \tag{4.2}$$

$$\begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \tag{4.3}$$

Magnitude and angle of $\vec{V_r}$

Figure 4.5 shows the space vector representation for the reference vector which rotates at an angular speed $\omega = 2\pi f$ in the $\alpha\beta$ frame, where f is the fundamental frequency of the inverter's output voltage. The space vector representation is divided in six sectors shifted by 60° forming an hexagon. In this case, the reference vector is located in sector I which is bounded by two active vectors ($\vec{V_k} = V_1$ and $\vec{V_{k+1}} = V_2$) and two zero vectors (V_0 and V_7) located at the origin of the hexagon. Depending on the sector the reference vector is passing through, different sets of switches will be turned on and off in the VSI.

The rotating reference vector $\vec{V_r}$ is defined as in equation 4.4 where V_r is the magnitude of the vector and θ is the angle with respect to the phase a-axis.

$$\vec{V}_r = V_r \cdot e^{j\theta} \tag{4.4}$$

The $\alpha\beta$ voltage components obtained in the previous step are used to compute the scalar magnitude of the reference vector and the angle θ as stated in equations 4.5 and 4.6, respectively.

$$V_r = \sqrt{{V_\alpha}^2 + {V_\beta}^2} \tag{4.5}$$

$$\theta = \arctan(\frac{V_\beta}{V_\alpha}) \tag{4.6}$$

Sector determination and angle γ

The reference vector $\vec{V}_r$ can lie in any of the six sectors that define the hexagon shown in figure 4.5. By using the angle θ its position can be determined as stated in Table 4.1.

Table 4.1: Sector determination from the reference vector angle

Angle	Sector Number
$0^\circ \leqslant \theta < 60^\circ$	1
$60^\circ \leqslant \theta < 120^\circ$	2
$120^\circ \leqslant \theta < 180^\circ$	3
$180^\circ \leqslant \theta < 240^\circ$	4
$240^\circ \leqslant \theta < 300^\circ$	5
$300^\circ \leqslant \theta < 360^\circ$	6

The angle γ is the angle between $\vec{V}_r$ and the adjacent active vector $\vec{V}_k$ at a particular time. From the corresponding sector number, where the reference vector falls, and the angle of the reference vector, θ, it is possible to calculate this angle as in equation 4.7. This angle can take values from 0° up to 60° and will be used in the next step for calculating the time during each of the active vectors should be on.

$$\gamma = \theta - (k-1) \cdot \frac{\pi}{3} \tag{4.7}$$

Calculation of the dwell times

Depending on the position of the reference vector, the active vectors that bound the corresponding sector should be active during certain time intervals which are known as dwell times. These time intervals are calculated based on the magnitude and angle of $\vec{V}_r$ and the active vectors. In a general form, the active vectors are defined as in equation 4.8 and the reference vector as in equation 4.4.

$$\vec{V}_k = \begin{cases} \frac{2}{3} \cdot V_{DC} \cdot e^{j(k-1)\frac{\pi}{3}} & \text{if k=1,2,...,6} \\ 0 & \text{if k=0,7} \end{cases} \tag{4.8}$$

As an example, if the reference vector is in sector I, the active vectors are $\vec{V_k} = \vec{V_1}$ and $\vec{V_{k+1}} = \vec{V_2}$. This means that $\vec{V_1}$ and $\vec{V_2}$ will be active in a sampling period during T_1 and T_2, respectively. The zero vectors will be active during a time interval T_0. These dwell times can be determined by applying volt-sec balance using the reference vector and the two adjacent active vectors [8]. The vector diagram showing the dwell times when the reference vector is located in sector I is depicted in figure 4.8.

$$\vec{V_r} \cdot Ts = \vec{V_1} \cdot T_1 + \vec{V_2} \cdot T_2 \tag{4.9}$$

Substituting the corresponding expressions for the reference vector and the active vectors (equations 4.4 and 4.8) in the previous equation it is obtained:

$$V_r \cdot e^{j\theta} = d_x \cdot \frac{2}{3} \cdot V_{DC} \cdot e^{j0} + d_y \cdot \frac{2}{3} \cdot V_{DC} \cdot e^{j\frac{\pi}{3}} \tag{4.10}$$

where $d_x = \frac{T_1}{T_s}$ and $d_y = \frac{T_2}{T_s}$ are the ratio between the dwell times (T_1 and T_2) and the switching period (T_s). Solving equation 4.10 by applying Euler's equation and equating real and imaginary terms:

$$V_r \cdot cos\theta = d_x \cdot \frac{2}{3} \cdot V_{DC} + d_y \cdot \frac{2}{3} \cdot V_{DC} \cdot cos\frac{\pi}{3} \tag{4.11}$$

$$j \cdot V_r \cdot sin\theta = j \cdot d_y \cdot \frac{2}{3} \cdot V_{DC} \cdot sin\frac{\pi}{3} \tag{4.12}$$

Thus, d_x and d_y corresponding to a reference vector lying in sector I is calculated as shown in equations 4.13 and 4.14. These will be used for finding the corresponding duty cycle for each of the inverter's legs.

$$d_x = \frac{\sqrt{3} \cdot V_r}{V_{DC}} \cdot sin(\frac{\pi}{3} - \theta) \tag{4.13}$$

$$d_y = \frac{\sqrt{3} \cdot V_r}{V_{DC}} \cdot sin\theta \tag{4.14}$$

Equations 4.13 and 4.14 can be written in a general form, valid for all the sectors, as:

$$d_x = \frac{T_1}{T_s} = \frac{\sqrt{3} \cdot V_r}{V_{DC}} \cdot sin(k\frac{\pi}{3} - \theta) \tag{4.15}$$

$$d_y = \frac{T_2}{T_s} = \frac{\sqrt{3} \cdot V_r}{V_{DC}} \cdot sin\gamma \tag{4.16}$$

In general, the zero vectors $\vec{V_0}$ and $\vec{V_7}$ will be active during the switching period for the same amount of time as depicted in figure 4.9.

$$T_0 = T_s - T_1 - T_2 \tag{4.17}$$

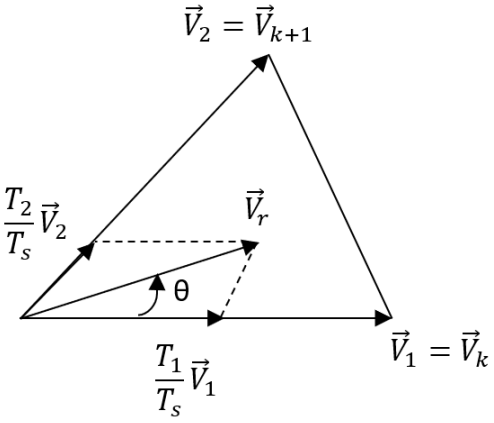


Figure 4.8: Vector diagram for dwell times calculation in sector I.

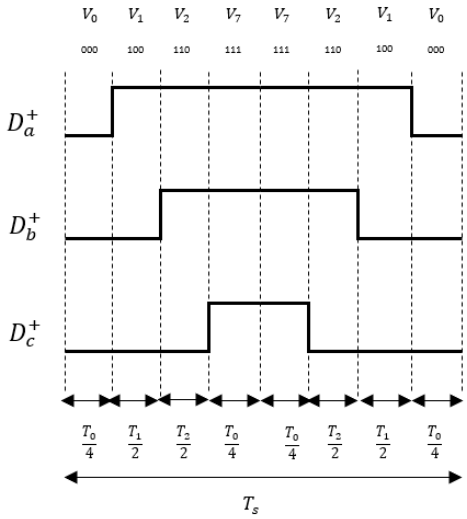


Figure 4.9: 7-segment switching sequence for sector I.

Switching sequence

The switching sequence for the space vector modulation scheme is arranged in a symmetrical manner around the center of the switching period. This is done in order to obtain at the inverter's output voltages with quarter-wave symmetry to ensure a reduction of the THD [8].

For obtaining the symmetric switching sequence the 7-segment technique is implemented in this project. The switching pattern consists of a zero vector followed by the two adjacent active vectors and ends with the other zero vector to complete half a switching period. The next half period is the mirror image of the first half as shown in figure 4.9. This figure shows the case of a modulating reference vector that falls in sector 1 in order to continue the previous example. It is observed that the sequence is $\vec{V}_0 \rightarrow \vec{V}_1 \rightarrow \vec{V}_2 \rightarrow \vec{V}_7$ for the first half period and $\vec{V}_7 \rightarrow \vec{V}_2 \rightarrow \vec{V}_1 \rightarrow \vec{V}_0$ for the second half.

The switching sequence is selected in this way to make sure that the transition from state to state involves the minimum number of switching. Thus, it is possible to reduce the switching losses of the inverter because each of the switches turns on and off only once per switching period. Therefore, by using the 7-segment switching pattern center-aligned pulses are obtained allowing a reduction in the THD and switching losses. Table 4.2 shows the switching sequence for all the six sectors.

Table 4.2: Switching sequence for SVPWM.

Switching sequence	Sector
$\vec{V}_0 \rightarrow \vec{V}_1 \rightarrow \vec{V}_2 \rightarrow \vec{V}_7 \rightarrow \vec{V}_2 \rightarrow \vec{V}_1 \rightarrow \vec{V}_0$	1
$\vec{V}_0 \rightarrow \vec{V}_3 \rightarrow \vec{V}_2 \rightarrow \vec{V}_7 \rightarrow \vec{V}_2 \rightarrow \vec{V}_3 \rightarrow \vec{V}_0$	2
$\vec{V}_0 \rightarrow \vec{V}_3 \rightarrow \vec{V}_4 \rightarrow \vec{V}_7 \rightarrow \vec{V}_4 \rightarrow \vec{V}_3 \rightarrow \vec{V}_0$	3
$\vec{V}_0 \rightarrow \vec{V}_5 \rightarrow \vec{V}_4 \rightarrow \vec{V}_7 \rightarrow \vec{V}_4 \rightarrow \vec{V}_5 \rightarrow \vec{V}_0$	4
$\vec{V}_0 \rightarrow \vec{V}_5 \rightarrow \vec{V}_6 \rightarrow \vec{V}_7 \rightarrow \vec{V}_6 \rightarrow \vec{V}_5 \rightarrow \vec{V}_0$	5
$\vec{V}_0 \rightarrow \vec{V}_1 \rightarrow \vec{V}_6 \rightarrow \vec{V}_7 \rightarrow \vec{V}_6 \rightarrow \vec{V}_1 \rightarrow \vec{V}_0$	6

Based on the dwell times and the switching sequence, the modulating reference signals for each of the three phases are generated and compared with the triangular carrier signal in order to generate the duty cycles for the inverter's switches. Following the previous example ($\vec{V}_r$ in sector I), the modulating reference signals are obtained from the dwell times by setting the condition that the on time for the zero vectors has to be the same in one sample period. Then, from the 7 segment switching sequence of figure 4.9 it is possible to obtain the relationship between duty cycles and the dwell times resulting in equation 4.18.

$$\begin{aligned} 1 &= D_a + D_c \\ d_x &= D_a - D_b \\ d_y &= D_b - D_c \end{aligned} \tag{4.18}$$

Writing this equation in matrix form and, calculating the inverse matrix, is possible to generate the reference signals from the dwell times. The same procedure is followed for the six sectors and the resulting matrixes are included in the gain blocks shown in figure 4.7.

$$\begin{bmatrix} 1 \\ d_x \\ d_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} D_a \\ D_b \\ Dc \end{bmatrix} \rightarrow \begin{bmatrix} D_a \\ D_b \\ D_c \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ d_x \\ d_y \end{bmatrix} \tag{4.19}$$

4.3.1 Simulation results

In this section the simulation results obtained when implementing the SVPWM in Simulink will be presented in order to validate the modulation scheme. From figure 4.10 it can be seen the three modulating reference signals, which are compared with the carrier signal, together with the sector where the reference vector is located. These reference signals, compared with the sinusoidal reference waveforms, have lower amplitude resulting in a better utilization of the supply voltage as it was explained in subsection 4.1.3. As an example, a zoomed view for a time instant when the modulating signal is in sector I is seen in figure 4.11. From this figure it can be validated the 7-segment technique for obtaining a symmetric switching sequence.

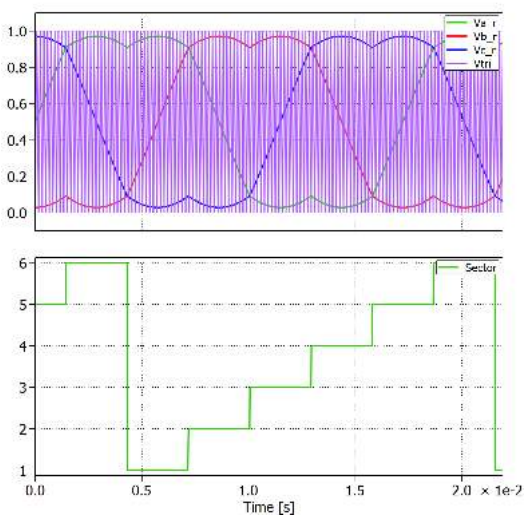


Figure 4.10: *Top graph:* Reference signals for the three phases compared with the carrier signal. *Bottom graph:* Sector number.

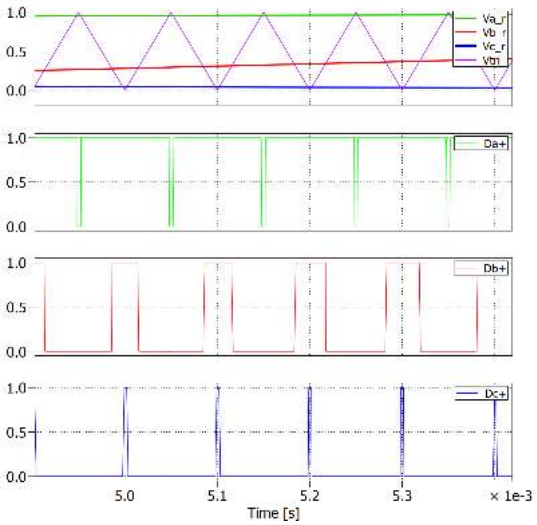


Figure 4.11: Zoomed view of the generation of the duty cycles for each of the inverter's legs in the case of $\vec{V}_r$ located in sector I.

Figure 4.12 shows the voltage for phase a and the line to line voltage between phase a and phase b at the inverter's output together with their corresponding sinusoidal reference signals. These sinusoidal reference signals are included in the system as inputs to model the voltages in the abc reference frame which are transformed to the $\alpha\beta$ through Clarke's transformation. However, at this point there is no controller included in the system as it is shown in the block diagram in figure 4.6. From figure 4.12 it is observed that, in both cases, the inverter's output voltages have quarter-wave symmetry in order to reduce the harmonic distortion. These voltages are in phase with the sinusoidal reference voltages. Moreover, it is observed that the phase voltage has a maximum amplitude of $\frac{2}{3}V_{DC} = 24V$ and it also has intermediate values of $\frac{1}{3}V_{DC} = 12V$ as it was shown in section/appendix "blablabla" . In the case of the line to line voltage, it varies between $-V_{DC} = -36V$ and $V_{Dc} = 36V$, as expected.

In the VSI section in chapter "Power train" or in appendix include the analysis of the phase and line voltages (as explained in the lectures) in order to refer here to it to show that the phase voltages can take values (2/3Vdc, -2/3Vdc, 1/3Vdc and -1/3Vdc) and the line to line voltages vary from -Vdc to Vdc. Stef

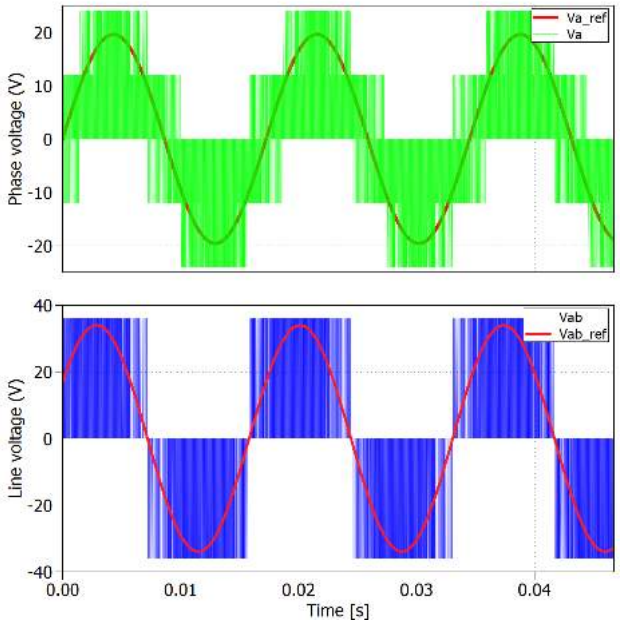


Figure 4.12: *Top graph:* Sinusoidal phase a reference voltage V_{aref} compared with the phase a voltage V_a at the inverter's output. *Bottom graph:* Sinusoidal reference line to line voltage V_{abref} compared with the line to line voltage V_{ab} at the inverter's output.

Finally, for validating the SVPWM the induction machine's behaviour is analyzed. The nominal parameters of the IM are listed on table 3.1. By applying a step load torque of rated value $T_{load} = 30.4Nm$ at $0.3s$, the electromagnetic torque, shaft speed and phase currents are obtained and plotted in figure 4.13. The electromagnetic torque T_e follows the load torque reaching its rated value and the rotor speed reaches a steady state value of 1681 rpm which is approximately the nominal rotor speed of 1685rpm. A slip value of 3.16% is obtained which is equal to the slip obtained when validating the IM parameters in section 3.1.2. The peak value of the stator phase currents after applying the step load torque is 264.5 A. The nominal phase current is 189A which corresponds to a peak value of $189 \cdot \sqrt{2} = 267.28A$. Therefore, the rated parameters of the IM are reached after applying the rated load torque and, consequently, the SVPWM can be validated.

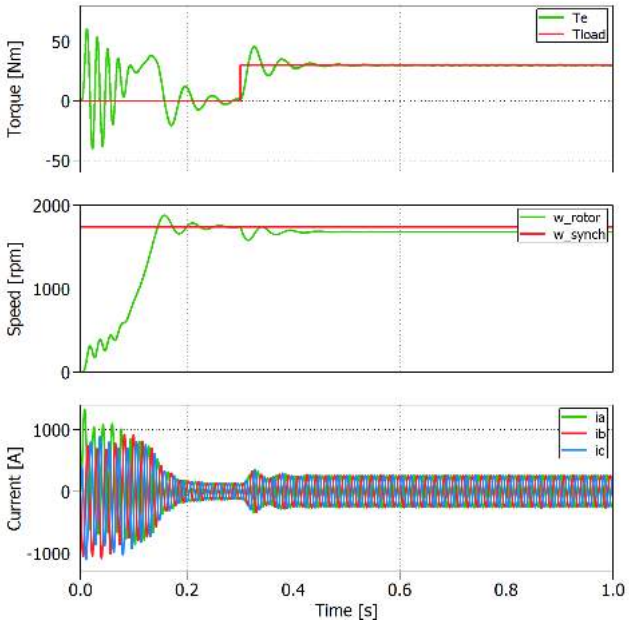


Figure 4.13: *Top graph:* Electromagnetic torque T_e and load torque T_{load}. *Middle graph:* Speed of the rotor and rated synchronous speed. *Bottom graph:* Stator phase currents i_a, i_b and i_c.

4.4 Field oriented control

Field oriented control (FOC), also known as vector control, is widely used for controlling the speed of induction machines due to its high dynamic performance and reliability [8][10]. The principle of the FOC technique is based on the decoupling of the stator currents into two components in order to achieve independent control of the flux and the torque [8][10]. The decoupling of the stator currents is done by implementing Clarke's anf Park's transformation ($abc \rightarrow \alpha\beta \rightarrow dq$) to the stator currents to obtain a two phase rotating dq reference frame aligned with the flux vector[8]. It is important to note that the dq system is rotating synchronously at electrical frequency. Usually, the dq reference frame is used for control purposes and the stationary $\alpha\beta$ frame for modelling, as it was done for the modelling of the IM.

The alignment of the dq reference frame with the flux vector can be carried out in different ways depending on which flux vector is chosen to be oriented. Two main techniques can be implemented: stator flux oriented control and rotor flux oriented control [9]. Stator flux orientation forces the stator d-axis current to be oriented with the stator flux linkage while rotor flux oriented control forces the d-axis to be constantly aligned with the rotor flux linkage. The latter is the selected technique to be implemented in this project and its principle is explained in detail in subsection 4.4.2. Rotor flux oriented control is the selected control technique because it has shown superior dynamic performance than stator flux oriented control [11]. Stator flux oriented control, in comparison with rotor flux oriented, does not present very accurate reaction to torque commands which could lead to instability [11].

The direct stator current (i_{ds}) is used for the rotor flux while the quadrature current (i_{qs}) is the responsible for controlling the torque. By holding the flux constant, the torque is directly proportional to the stator q-axis current as it will be shown later on in this chapter. Thus, the flux linkage vector will generate the reference for the d-axis current controller and, on the other hand, the applied torque will produce the reference for the q-axis current controller [8][9][10].

4.4.1 Indirect field oriented control

Besides the type of vector control regarding the flux orientation, there are also two different ways of implementing FOC for determining the rotor flux position. Depending on the possibility to measure or not the rotor's shaft position of the IM, FOC can also be divided into direct and indirect FOC.

The direct technique, also known as sensor-less technique, calculates the rotor flux position without using an encoder or tachometer in the IM's shaft. The rotor flux position is either measured by using flux sensors in the IM or it is estimated from the voltage equations [8].

On the other hand, indirect FOC utilizes the rotor speed/position measurement to determine the rotor flux position. As the rotor field rotates at synchronous speed in a dq reference frame and, the mechanical rotor speed is measurable, it is possible to obtain the rotor flux speed if the slip speed can be estimated. Using the mechanical rotor speed and the slip speed, the synchronous speed is obtained and, by integrating it, the rotor flux position is determined [8]. Indirect FOC is the selected technique because the induction machine available in the laboratory already has an incremental encoder mounted in the IM shaft.

4.4.2 Principle of rotor flux oriented control

The basic idea of rotor flux FOC is to use an arbitrary dq reference frame where the rotor flux is constantly aligned with the real part of the stator current i_{ds}. The rotor flux orientation is depicted in the phasor diagram of figure 4.14. This orientation allows decoupling of the IM torque and flux control variables making possible to operate the induction machine as a separately excited DC motor [8][9]. This means that by transforming the three phase stator currents into two current components, in a synchronously rotating dq frame, i_{qs} and i_{ds} are analogous to the field and armature current of a DC machine, respectively [8].

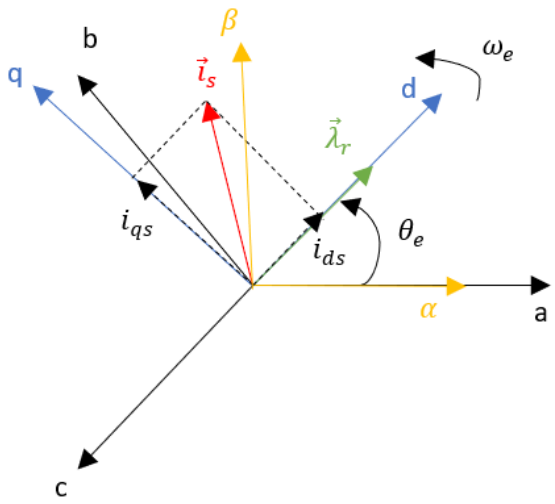


Figure 4.14: Phasor diagram showing rotor flux orientation.

The general torque equation for an induction machine in the dq reference frame is expressed as [8]:

$$T_e = \frac{3}{2} \cdot p \cdot L_m \cdot (i_{qs} \cdot i_{dr} - i_{ds} \cdot i_{qr}) \tag{4.20}$$

This equation can be written in different forms depending on the type of control that will be implemented. In this project, as rotor flux oriented control is the selected technique, the previous equation is defined in a vector form as in equation 4.21. The torque is a function of the rotor flux linkage $\vec{\lambda_{dqr}}$ and the stator currents $\vec{i_{dqs}}$:

$$T_e = \frac{3}{2} \cdot p \cdot \frac{L_m}{L_r} \cdot \mathrm{Im}(\vec{i_{dqs}} \cdot \vec{\lambda_{dqr}}^{\,*}) \tag{4.21}$$

where,

$$\vec{i_{dqs}} = i_{ds} + j \cdot i_{qs} \qquad \vec{\lambda_{dqr}}^{\,*} = \lambda_{dr} - j \cdot \lambda_{qr} \tag{4.22}$$

Substituting equation 4.22 in equation 4.21, it is obtained that the electromagnetic torque depends proportionally on the stator q-axis current:

$$T_e = \frac{3}{2} \cdot p \cdot \frac{L_m}{L_r} \cdot \lambda_{dr} \cdot i_{qs} \tag{4.23}$$

Therefore, it is possible to conclude that the quadrature stator component is responsible of producing torque. The direct stator current does not contribute in the torque production but in the flux, as it will be shown later on in this section. This is the basic operation of rotor flux oriented control which is illustrated in the block diagram of figure 4.15.

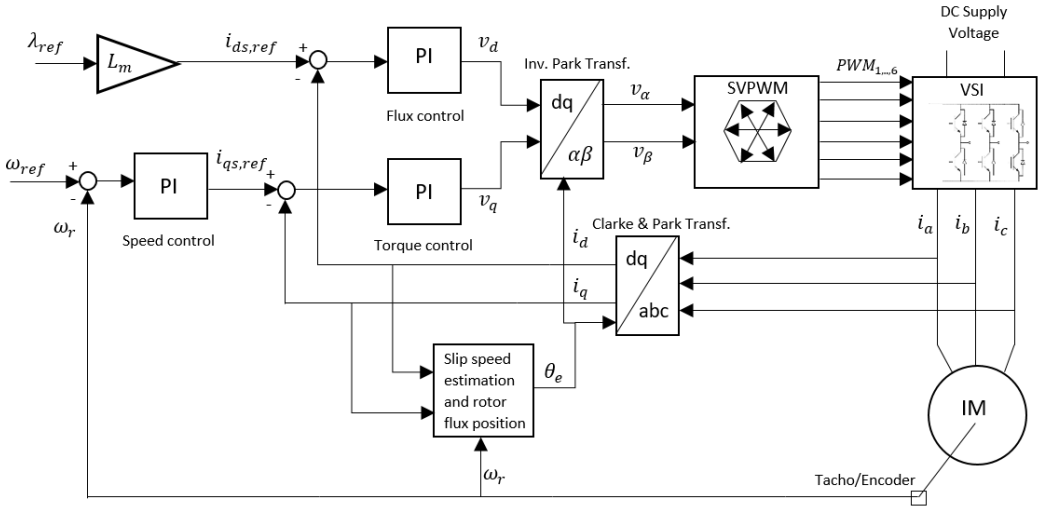


Figure 4.15: Block diagram of indirect rotor flux field oriented control.

From the block diagram it is observed that two PI current controllers are required for independently controlling the flux and the electromagnetic torque. The output of these controllers are the two voltage components in the dq reference frame. These dq voltage components are further transformed to the $\alpha\beta$ reference frame in order to implement the SVPWM. These reference frame are shown in figure 4.14. Besides the two PI current controllers, it is also possible to implement an outer speed controller that will generate the torque reference for the q-axis current controller [9][8].

In this project, the reference for the q-axis current $i_{qs,ref}$ can be generated in two ways which will be named torque control and speed control. Torque control will generate the required current reference directly from a torque profile reference using equation 4.23. Speed control, also known as cruise control, will generate $i_{qs,ref}$ from an outer PI control loop. For both types of control, it is necessary to determine the rotor flux position.

Determination of the rotor flux position

Field oriented control requires performing reference frame transformations because the controller is implemented in an arbitrary dq reference frame. The rotor flux position θ_e has to be determined in order to perform the transformations required for field orientation. This position will be obtained by integrating the electrical stator speed ω_e which is calculated as the sumation of the slip speed and the mechanical rotor speed. The induction machine voltage and flux linkage equations represented in a dq reference frame are [8]:

$$u_{qr} = R_r \cdot i_{qr} + \frac{d\lambda_{qr}}{dt} + (\omega_e - \omega_r) \cdot \lambda_{dr} \tag{4.24}$$

$$u_{dr} = R_r \cdot i_{dr} + \frac{d\lambda_{dr}}{dt} - (\omega_e - \omega_r) \cdot \lambda_{qr} \tag{4.25}$$

$$\lambda_{qr} = L_{lr} \cdot i_{qr} + L_m \cdot (i_{qs} + i_{qr}) \tag{4.26}$$

$$\lambda_{dr} = L_{lr} \cdot i_{dr} + L_m \cdot (i_{ds} + i_{dr}) \tag{4.27}$$

As the rotor windings are short-circuited ($u_{qr} = u_{dr} = 0$) and in rotor flux oriented control the d-axis is aligned with the rotor flux linkage ($\lambda_{qr} = 0$), the previous equations are reduced to:

$$0 = R_r \cdot i_{qr} + \frac{d\lambda_{qr}}{dt} + \omega_{sl} \cdot \lambda_{dr} \tag{4.28}$$

$$0 = R_r \cdot i_{dr} + \frac{d\lambda_{dr}}{dt} \tag{4.29}$$

$$0 = L_r \cdot i_{qr} + L_m \cdot i_{qs} \tag{4.30}$$

$$\lambda_{dr} = L_r \cdot i_{dr} + L_m \cdot i_{ds} \tag{4.31}$$

where $\omega_{sl} = \omega_e - \omega_r$ is the slip speed and $L_r = L_{lr} + L_m$ is the sum of the rotor leakage inductance and the magnetizing inductance.

The rotor quadrature current i_{qr} is obtained from equation 4.30 as a function of the stator quadrature current i_{qs} (equation 4.32) in order to eliminate the rotor currents as they cannot be measured. Substituting i_{qr} in equation 4.28, the slip speed is obtained as in equation 4.33, where $\tau_r = \frac{L_r}{R_r}$ is the rotor time constant. The derivative term of equation 4.28 is 0 because the rotor flux is kept constant.

$$i_{qr} = -\frac{L_m}{L_r} \cdot i_{qs} \tag{4.32}$$

$$\omega_{sl} = \frac{L_m}{\tau_r} \cdot \frac{i_{qs}}{\lambda_{dr}} \tag{4.33}$$

The rotor flux linkage, $\mid \vec{\lambda_{dqr}} \mid = \lambda_{dr} = \lambda_r$, is usually held constant at its rated value when the IM is operating below the base speed [9]. This approach is done to be able to vary the electromechanical torque from zero to maximum value without having to adjust λ_r. Hence, as the rotor flux linkage is kept constant, the torque will only depend on i_{qs} (equation 4.23).

From the d-axis rotor voltage equation (equation 4.29) it is obtained that the rotor d-axis current is zero because the rotor flux λ_r remains constant. Therefore, by including this in equation 4.31, the relationship between the flux linkage and the d-axis stator current is defined in equation 4.35.

$$i_{dr} = -\frac{1}{R_r} \cdot \frac{d\lambda_r}{dt} \rightarrow i_{dr} = 0 \tag{4.34}$$

$$\lambda_r = L_m \cdot i_{ds} \tag{4.35}$$

Thus, the estimation of the slip speed will depend upon the quadrature and direct stator currents as stated in equation 4.36.

$$\omega_{sl} = \frac{1}{\tau_r} \cdot \frac{i_{qs,ref}}{i_{ds,ref}} \tag{4.36}$$

A common simplification approach is to use the reference stator currents for the slip speed determination instead of the measured currents. This is because these currents are usually less noisier than the measured ones and the results achieved in steady state are almost the same. The reference quadrature current $i_{qs,ref}$ is obtained from the torque command (equation 4.23) and the reference direct current $i_{ds,ref}$ from the rotor flux linkage command (equation 4.35).

Once the slip speed has been estimated, the electrical stator speed ω_e can be calculated as in equation 4.37. The mechanical speed ω_r is obtained from the position sensor (encoder) that is attached to the motor's shaft as indirect FOC is implemented.

$$\omega_e = \omega_{sl} + \omega_r \tag{4.37}$$

The rotor flux position is found by integrating the electrical speed:

$$\theta_e = \int (\omega_{sl} + \omega_r) \tag{4.38}$$

Calculation of rated rotor flux linkage

The reference d-axis current $i_{ds,ref}$ for the PI flux controller is determined from the desired rotor flux level. Therefore, the next step is to calculate the rated rotor flux linkage $\lambda_{r,max}$. The induction machine voltage and flux linkage equations in vector form, in a dq reference frame, are defined as in equations 4.39 and 4.40, respectively.

$$\vec{u_{dqs}} = R_s \cdot \vec{i_{dqs}} + j \cdot \omega_e \cdot \vec{\lambda_{dqs}} \qquad \vec{u_{dqr}} = 0 = R_r \cdot \vec{i_{dqr}} + j \cdot (\omega_e - \omega_r) \cdot \vec{\lambda_{dqr}} \tag{4.39}$$

$$\vec{\lambda_{dqs}} = L_s \cdot \vec{i_{dqs}} + L_m \cdot (\vec{i_{dqr}}) \qquad \vec{\lambda_{dqr}} = L_r \cdot \vec{i_{dqr}} + L_m \cdot (\vec{i_{dqs}}) \cdot \vec{\lambda_{dqr}} \tag{4.40}$$

The stator flux linkage $\vec{\lambda_{dqs}}$ is obtained from the stator voltage equation as the induction machine's rated phase voltage and current are known parameters. Induction motors always run at lagging power factor (PF) which means that in a phasor representation the voltage is at the origin (0°) and the current is lagging by the angle defined by the nominal $PF = \cos\alpha = 0.76$. Therefore, the rated phase voltage and current are calculated as in equations 4.41 and 4.42.

$$\vec{u_{dqs}} = u_{pk,rated} \cdot e^{j0^\circ} = \frac{24V \cdot \sqrt{2}}{\sqrt{3}} = 19.6V\angle 0^\circ \tag{4.41}$$

$$\vec{i_{dqs}} = i_{pk,rated} \cdot e^{-j\alpha} = 267.28A\angle -40.53^\circ \tag{4.42}$$

The nominal frequency of the IM is 58 Hz, therefore, the rated electrical synchronous speed is $\omega_e = 2\pi f = 364.42$ rad/s. From this and the nominal voltage and current calculated previously, the rated stator flux linkage is found to be:

$$\vec{\lambda_{dqs}} = \frac{1}{j\omega_e} \cdot (\vec{u_{dqs}} - R_s \cdot \vec{i_{dqs}}) = 0.0524\angle -88.68^\circ \tag{4.43}$$

On the other hand, from the rotor flux linkage equation, $\vec{i_{dqr}}$ is obtained and included in the stator flux linkage equation and performing maths operations the rotor flux linkage results in equation 4.44, where $\sigma = L_s - \frac{L_m^2}{Lr}$ is the resultant leakage constant.

$$\vec{\lambda_{dqr}} = \frac{L_r}{L_m} \cdot (\vec{\lambda_{dqs}} - \sigma \cdot L_s \cdot \vec{i_{dqs}}) = 0.05671\angle -88.68^\circ \rightarrow \lambda_{r,max} = 0.05671 \tag{4.44}$$

From this rated value of the rotor flux linkage the direct stator current reference is obtained from equation 4.35:

$$i_{ds,ref} = \frac{\lambda_{r,max}}{L_m} = 149.22A \tag{4.45}$$

4.5 Implementation of indirect rotor flux oriented FOC

This section presents the procedure followed for implementing indirect rotor flux FOC. As it was mentioned in subsection 4.4.2, two types of control will be implemented in this project: torque control and speed/cruise control. In the case of torque control only the two inner control loops are required while for cruise control a cascaded controller needs to be implemented by adding an outer control loop for the speed. In any case, the controllers will be first designed in continuous domain and after they will be discretized in order to implement them using a DSP.

4.5.1 Design of PI controllers in continuous domain

In total three PI controllers will be designed in this section. It is important to consider the bandwidth of these controllers because when implementing cruise control a cascaded control architecture is necessary. Therefore, the inner current loops must be significantly faster (at least $3-5$ times) than the outer speed loop [12]. The reason for this is that the outer loop provides the set point for the inner loop and, making the inner loop faster, allows to compensate the inner loop disturbances before they affect the outer loop [12]. Therefore, the PI current controllers will be designed first and based on the selected bandwidth for the inner control loops the speed controller will be designed.

PI current controllers

First the design of the inner PI controllers loopd for the d- and q-axis currents will be carried out. It is necessary to obtain the transfer function of the plant to be controlled, which in this case is the induction machine. The plant's transfer function, with the stator current as output and voltage as input, is obtained from the IM equations. The stator voltage equations in Laplace form are:

$$V_{ds}(s) = R_s \cdot I_{ds}(s) + s\lambda_{ds}(s) - \omega_e \cdot \lambda_{qs}(s) \tag{4.46}$$

$$V_{qs}(s) = R_s \cdot I_{qs}(s) + s\lambda_{qs}(s) + \omega_e \cdot \lambda_{ds}(s) \tag{4.47}$$

Substituting the stator flux linkage equations in 4.46 and 4.47 yields to:

$$V_{ds}(s) = R_s \cdot I_{ds}(s) + s(L_s \cdot I_{ds}(s) + L_m \cdot I_{dr}(s)) - \underbrace{\omega_e \cdot (L_s \cdot I_{qs}(s) + L_m \cdot I_{qr}(s))}_{\text{decoupling term}} \tag{4.48}$$

$$V_{qs}(s) = R_s \cdot I_{qs}(s) + s(L_s \cdot I_{qs}(s) + L_m \cdot I_{qr}(s)) + \underbrace{\omega_e \cdot (L_s \cdot I_{ds}(s) + L_m \cdot I_{dr}(s))}_{\text{decoupling term}} \tag{4.49}$$

It is observed from the previous equations that it exists coupling between the d and q axis. Therefore, it is necessary to decouple them by adding or subtracting these terms in the current loops in order to have independent control of the torque and flux components and neglect these terms in the transfer function of the plant. These decoupling terms will be seen as disturbances in the system. Moreover, in order to simplify the transfer function of the plant, the rotor currents are neglected resulting in the transfer function of the IM.

$$G_p(s) = \frac{I_{ds}(s)}{V_{ds}(s)} = \frac{I_{qs}(s)}{V_{qs}(s)} = \frac{1}{R_s + L_s \cdot s} = \frac{\frac{1}{L_s}}{s + \frac{R_s}{L_s}} \tag{4.50}$$

The transfer function is the same for the d- and q-axis which means that the current loops for both will be exactly the same. Thus, the PI current controller will be design for the d-axis and the controller's parameters obtained will be also used for the quadrature current. The general form of a PI controller in the continuous domain is shown in equation 4.51 [12].

$$G_c(s) = K_P + \frac{K_I}{s} \tag{4.51}$$

When designing the current controllers it is necessary to include a time delay in order to obtain simulation results that can be compared with the results obtained in the laboratory. The delay appears in the system because of the digital implementation in a microcontroller which is not instantaneous. Therefore, as the controllers will be implemented in a DSP, the time for sampling the measured signals in the Analog to Digital Converter (ADC) together with the time it takes to perform the control algorithm have to be taken into account. The time delay is reduced as much as possible while programming the microcontroller achieving a total time delay of half a sampling period ($T_d = 0.5T_s$). The implementation in Simulink of the PI controller for i_{ds}, including the transfer function of the plant and the time delay, can be seen in figure 4.16. The transfer function for the time delay is:

Check this and explain it better if it isnt clear. Stef

$$G_d(s) = \frac{1}{T_d \cdot s + 1} \tag{4.52}$$

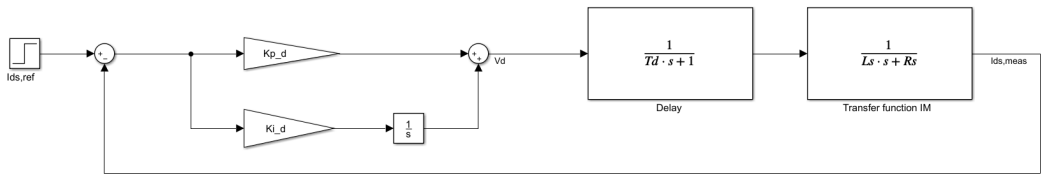


Figure 4.16: PI current controller in continuous domain for the direct stator current i_{ds}.

A common control approach is to select the zero of the controller in a way that it cancels out the pole of the original transfer function of the plant [12]. This zero-pole cancellation technique is done in order to obtain a closed-loop system with no zeroes and only one real pole to ensure continuous stable control. From equation 4.50 it can be seen that the pole of the plant is located in $s = -\frac{R_s}{L_s}$ and the zero of the PI controller in $s = -\frac{K_I}{K_P}$ (equation 4.51). Therefore, by equating these terms the integral gain of the PI controller is obtained as:

$$K_I = \frac{R_s}{L_s} \cdot K_P \tag{4.53}$$

The next step is to obtain the value of the proportional gain K_P. The open loop transfer function of the system is reduced to equation 4.54 after cancelling the plant's pole.

$$G_{OL}(s) = G_c(s) \cdot G_d(s) \cdot G_p(s) = \frac{\cancel{\left(s + \frac{K_I}{K_P}\right)} \cdot \frac{1}{L_s}}{\frac{s}{K_P} \cdot (T_d \cdot s + 1) \cdot \cancel{\left(s + \frac{R_s}{L_s}\right)}} \tag{4.54}$$

The closed-loop transfer function of the system is used to find the value of K_P. It is observed that the closed-loop is a second order system. Hence, equating the closed loop transfer function with the general form of a second order transfer function (equation 4.55), the value of K_P can be found.

$$G_{CL}(s) = \frac{G_{OL}(s)}{1 + G_{OL}(s)} = \frac{\frac{K_P}{T_d \cdot L_s}}{s^2 + \frac{1}{T_d} \cdot s + \frac{K_P}{T_d \cdot L_s}} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n \cdot s + \omega_n^2} \tag{4.55}$$

$$\omega_n = \frac{1}{2 \cdot \zeta \cdot T_d} \qquad K_P = L_s \cdot T_d \cdot \omega_n^2 \tag{4.56}$$

According to [13], the inner current PI controllers usually limit the bandwidth to approximately 10% of the switching frequency. The switching frequency is $f_s = 10KHz$, therefore, the requirement for the current controllers is that the bandwidth is $BW_c = 1KHz$. However, when discretizing the controller later on it was found that this bandwidth was very conservative and the currents showed high ripples. Therefore, the bandwidth requirement was reduced to half of the previous value ($BW_c = 500Hz$).

The proportional fain K_P depends upon the natural frequency ω_n of the system as the stator inductance and the time delay are known parameters. For a second order system, the bandwidth is related to the natural frequency as shown in equation 4.57 .

ref

$$\omega_{BW} = \omega_n \cdot \sqrt{(1 - 2\zeta^2) + \sqrt{4\zeta^4 - 4\zeta^2 + 2}} \tag{4.57}$$

Substituting the expression of ω_n obtained in equation 4.56 in the previous equation and setting the bandwidth requirement it is obtained that the damping ratio is $\zeta = 1.35$. This value corresponds to a natural frequency of $\omega_n = 7382 rad/s$. Thus, the values for the PI current controllers parameters are:

$$K_P = K_{P,d} = K_{P,q} = 1.12 \qquad K_I = K_{I,d} = K_{I,q} = 6.81 \tag{4.58}$$

Figure 4.17 shows the closed loop step response for the d-axis current for a step input reference of 1 A. The response to the step input does not show overshoot as it is an overdamped system ($\zeta > 1$). The rise time is the time the system takes to rise from 10% to 90% of its final value and it is found to be 0.7 ms. On the other hand, the settling time is the time required to reach 98% of the final value and it is, in this case, 1.27 ms.

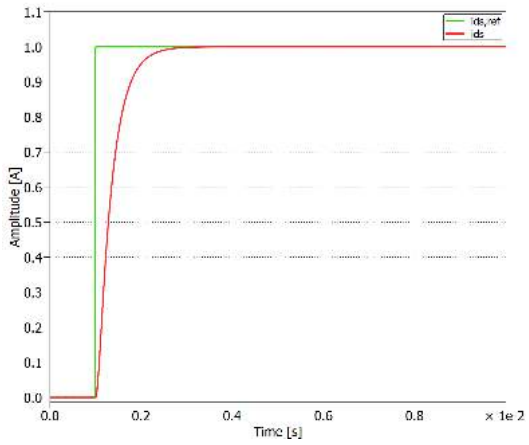


Figure 4.17: Closed loop step response for a direct reference value of $i_{ds,ref} = 1A$.

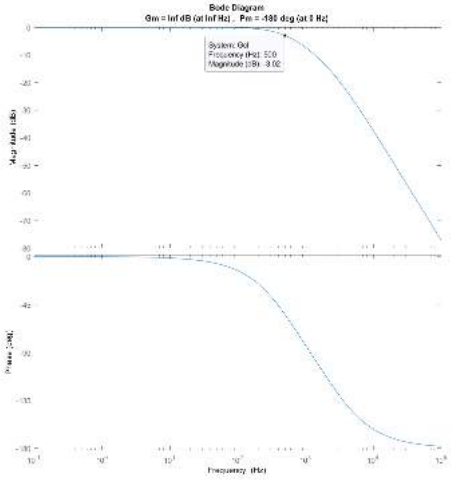


Figure 4.18: Bode diagram of the closed loop system including the PI current controller.

The bode diagrams of the closed loop system for the current controller is shown in figure 4.21. From this plot, the corresponding bandwidth is read at the point where the magnitude plot is -3dB, concluding that the controllers fulfil the bandwidth requirement $BM_c = 500$ Hz. Moreover, it is observed that the PI current controller corresponds to a second order system as its phase varies from 0° to -180° because the system has two poles.

PI speed controller

The outer speed loop is governed by the mechanical equation of the induction machine. The mechanical model of an induction machine is defined based on the load torque T_L, the motor's moment of inertia J, the viscous friction coefficient B and the rotor's speed ω_r [8]:

$$T_e - T_L = J \cdot \frac{d\omega_r}{dt} + B \cdot \omega_r \tag{4.59}$$

Laplace transforming equation 4.59, the transfer function of the plant for the design of the speed controller is found in equation 4.60. The load torque T_L is seen as a disturbance in the system.

$$G_{p,\omega} = \frac{\omega_r(s)}{T_e(s) - T_L(s)} = \frac{1}{J \cdot s + B} \tag{4.60}$$

The outer speed control loop will generate the reference electromagnetic torque $T_{e,ref}$ which provides the current reference for the quadrature current loop. As it was demonstrated at the beginning of this chapter, i_{qs} depends proportionally on T_e. Thus, as the rotor flux is kept constant at its rated value, the constant terms in equation 4.23 are grouped in a single variable $k = \frac{3}{2} \cdot p \cdot \frac{L_m}{L_r} \cdot \lambda_{dr}$. This constant is used to obtain the current reference $i_{qs,ref}$ from the electromagnetic torque reference $T_{e,ref}$.

Furthermore, the close loop transfer function of the current controller can be reduced to a first order transfer function because the outer speed loop is much slower than the inner loop which means that the time delay will not have significance effect in the speed controller. The value of the time constant τ_c that defines the delay introduced by the current loop is obtained from the step response of figure 4.17. It is found to be $\tau_c = 1.27$ms and it is this value the one used for the simplified transfer function for the inner current loop. Figure 4.19 shows the implementation of the speed controller in Simulink taking into account the inner current loop.

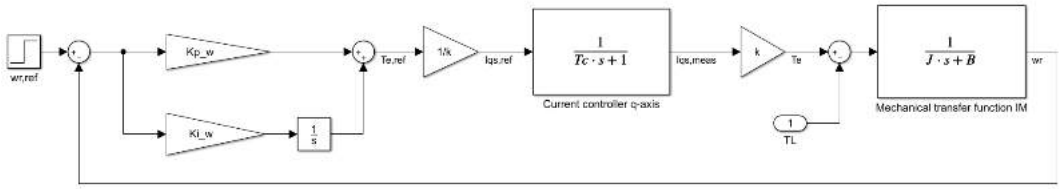


Figure 4.19: PI speed controller in continuous domain for generation of $i_{qs,ref}$.

Following the same zero-pole cancellation technique, as done during the design of the PI current controllers, the controller's zero is selected to cancel the induction machine's pole. This yields to the integral gain to be dependent of $K_{P,\omega}$.

$$K_{I,\omega} = \frac{B}{J} \cdot K_{P,\omega} \tag{4.61}$$

$$G_{OL,\omega}(s) = G_{c,\omega}(s) \cdot G_{p,\omega}(s) = \frac{\cancel{\left(s + \frac{K_{I,\omega}}{K_{P,\omega}}\right)} \cdot \frac{1}{J}}{\frac{s}{K_{P,\omega}} \cdot (1 + \tau_c \cdot s) \cdot \cancel{\left(s + \frac{B}{J}\right)}} \tag{4.62}$$

In cascaded controllers the inner loop has to be faster than the outer loop. Thus, it is decided that the speed controller must be 10 times slower than the current controller. This means that the bandwidth of the speed controller is $BW_\omega = 50$ Hz. The closed loop transfer function is reduced to a second order system. Identifying terms with the general expression of the transfer function of a second order system, as it was done during the design of the current controllers, it is found that the proportional gain $K_{P,\omega}$ depends on the natural frequency which at the same time depends on the bandwidth of the system (equation 4.57).

$$G_{CL,\omega}(s) = \frac{G_{OL,\omega}(s)}{1 + G_{OL,\omega}(s)} = \frac{\frac{K_{P,\omega}}{J \cdot \tau_c}}{s^2 + \frac{1}{\tau_c} \cdot s + \frac{K_{P,\omega}}{J \cdot \tau_c}} = \frac{\omega_n^2}{s^2 + 2\xi\omega_n \cdot s + \omega_n^2} \tag{4.63}$$

$$K_{P,\omega} = J \cdot \tau_c \cdot \omega_n^2 \tag{4.64}$$

Following the same procedure as before, it is found that the damping factor is in this case $\zeta = 1.01$ and the natural frequency $\omega_n = 495.05 rad/s$. Therefore, the speed controller's parameters are found to be:

$$K_{P,\omega} = 3.7 \qquad K_{I,\omega} = 0.37 \tag{4.65}$$

The value of Ki was found assuming a ration B/J = 0.1. However, this value needs to be found by experiment or if we dont get to do it it needs to be tuned when including it in the complete Simulink model.Stef

Figure 4.20 shows the closed loop response of the system for a speed input step of $\omega_{r,ref} = 1 rad/s$. It is observed that both, the rise time and the settling time are ten times higher than the obtained for the current controllers ($t_{r,\omega} = 7$ms and $t_{s,\omega} = 12$ms), as this was the requirement during the design of the speed controller. The bode diagram of the closed loop systems for the speed controller verifies the desired bandwidth of $BW_\omega = 50$Hz.

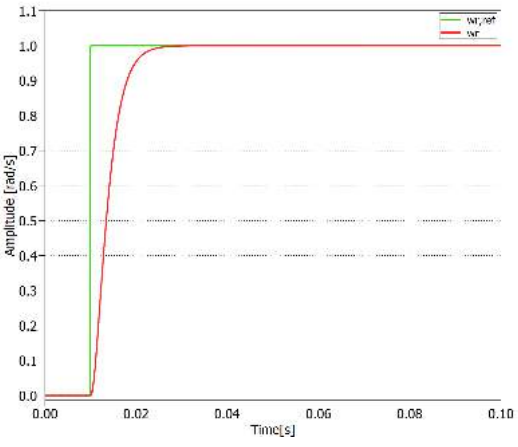


Figure 4.20: Closed loop response for a unit step rotor speed reference.

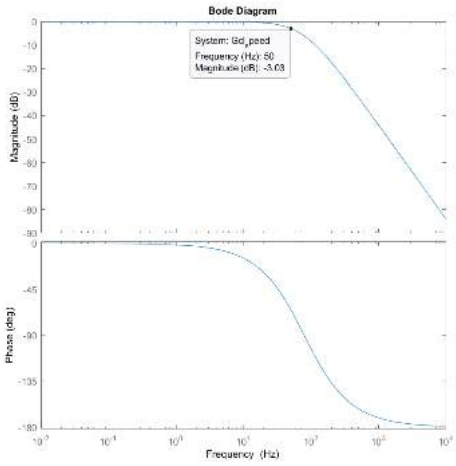


Figure 4.21: Bode diagram of the closed loop system including the PI current controller.

4.5.2 Discretization of the controller

mention here the anti-windup for the speed controller in order to avoid high level of currents. Stef

4.6 Verification of controller

4.6.1 Simulink model

4.6.2 Simulation results

4.7 Regenerative braking implementation

4.7.1 Simulation results

consider changing location of regenerative braking

4.8 Regenerative Braking

add more sections here don't know which ones include

Regenerative braking is just a special case of the overall vehicle control. Maybe you could have a chapter about overall vehicle simulation/control, Erik

In a conventional internal combustion engine, the only way to stop a vehicle in motion is by implementing mechanical brakes and dissipating the kinetic energy achieved during motion into heat. One of the biggest reason why electric vehicle is seen as a disrupter to IC engine is their ability to convert the kinetic energy of the vehicle into electrical energy which is stored back into the batteries when brake is applied. This phenomenon is popularly known in the industry as regenerative braking as the induction motor (in case our project) can be operated as generator during braking operation and recover the kinetic energy.

4.8.1 Types of regenerative braking

In spite of clear advantage of regenerative braking over conventional mechanical braking system, not all the braking requirement can be provided solely from regenerative method. Regenerative braking may not be used under conditions such as high state of charge (SOC), high temperature on battery. Furthermore, the required braking torque (deceleration) if implemented completely from regenerative braking might generate current amplitudes that will exceed the inverter or battery rating values. Furthermore, motor itself may not be able to provide the required braking force (deceleration).

From the above paragraph we conclude that there may be various constraints that will limit our regenerative braking capability and we will have to resort to frictional braking for the remaining brake requirement. There are two main control methodologies for splitting the required braking torque between regenerative and frictional braking.

- Parallel Regenerative Braking
- Series Regenerative Braking

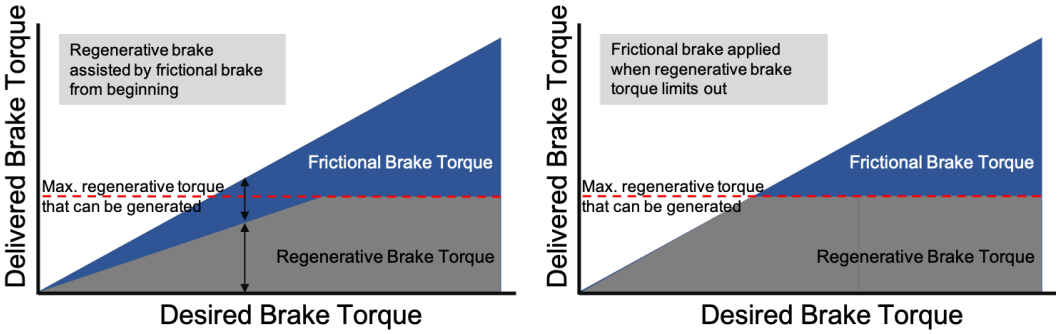


Figure 4.22: *Left:*Parallel and *Right:*Series Regenerative braking control

In figure 4.22 left side represents parallel regenerative braking strategy wherein both frictional as well as regenerative braking work in tandem sharing the required braking torque from $0Nm$. Ratio with which the two techniques share the braking load ($T_{friction}/T_{regen}$) depends on multiple parameter. As it can be seen that the ratio remains constant to the point where regenerative braking max out. Beyond this point, rest of braking torque is provided completely by frictional brakes.

In contrast to parallel braking strategy, in series regenerative braking strategy (4.22 *Right*) from $0Nm$ to the point where regenerative braking torque maxes out entire braking torque is provided by regeneration. Beyond this point frictional brake activates and provides the remaining required braking torque.

In commercial application it is preferable to employ parallel braking strategy or limit regenerative braking to be activated only after certain minimum threshold speed. Because at low speed the current generated by motor (generator mode) is not significant to ensure desirable battery recharge efficiency. Therefore at these low speed, frictional brake is employed to limit the battery charge/discharge cycles.

In this project our objective is to analyse the efficiency improvement due to regenerative braking capability. We will employ series braking control to ensure all the possible [to be further continued...]

4.9 Test results of the complete system in test bench

5 Firmware design, implementation and testing

In order to control the inverter, it is necessary to develop a control algorithm. Although the latest task is driving this motor by reading analog currents and generating the corresponding PWM signals, a relatively large framework must be developed to allow that task. For example, the user interaction with the system must be considered, the proper reference source must be used or safety techniques must be implemented to ensure that the system is behaving properly and react if it is not.

The software development procedure started with software design, where the system's features were described. After this initial stage the efforts were put into actual code development, in this stage the low level design was used as an input and the code was generated and tested to fulfil the design. After every software component was developed, the testing procedure of the specific software component was written in a common test description document. Finally the system was tested including the inverter and the motor. Significant attention has been put into the timing of the tasks and providing a framework of logging data. Also a Graphical User Interface (GUI) has been developed to provide a friendly communication channel between the controller and the user.

too many "develop"? AT. Only 2 right?NM

The used microcontroller consists of a Texas Instruments TMS320F28069M from the Piccolo family. The microcontroller is mounted in a development kit.

In this section the software architecture of the system is presented, showing the main blocks of the system and discussing execution triggers and time management of the tasks. After that, some software components which are of especial interest are examined. Finally the validation techniques used are reviewed.

5.1 High level design

The software has been designed prior to its development. In this section, the software design is discussed.

The concept of addressing the system as a whole and deciding which modules

will integrate the software is called software architecture. The software architecture divides all the functions to be performed into modules. After this modules and its specific tasks are described, it's necessary to design how are the module going to communicate and when will the modules be run.

The main reasons to follow the architectural design before implementation are the following:

1. In this case, the task to perform (control a motor) is a big task and will require several lines of code, where more than two developers will contribute. In this conditions, if the developers don't follow a master plan, the developed modules might not be aligned. And then additional work must be performed to align the modules, leading to extra development time. When every developer is following its own experience for developing, the result is usually known as *Spaghetti Code*. This code is usually unstructured, it is difficult to maintain and bugs are hard to find. If the developers can follow a design, the module will be coded just once and inputs, outputs and expected behaviour will be known from other modules before it's developed.

2. The development speed is increased.

3. Most design errors and bottlenecks are found in the design procedure instead of the development phase, then the implementation time is decreased.

This seems to be the same as 2, just more in depth. AT. I dont agree, you develop faster because you know your inputs and your outputs clearly and then you can have a higher focus level, in the 3 points it is discussed that you avoid bottlenecks.NM

4. If the modules are first designed and discussed, the resulting implementation will have a common development philosophy and style, leading to easier changes by any team member, the *ownership* of software modules is removed.

5. The system's scalability is increased by using a modular approach. Additional features can be easily added and there's a procedure established for that goal. Sometimes when the system is finished, new system requirements arise. For fulfilling this new requirement, additional features must be developed or existing features must be redesigned, by using a modular approach the overhead of adding the new feature to the system is minimised.

6. Reliability, by understanding the system as a whole, the reliability of the system is increased.

The procedure followed for the software design can be seen in figure 5.1. Every stage with the generated output and the approximate time spent relative to the whole procedure.

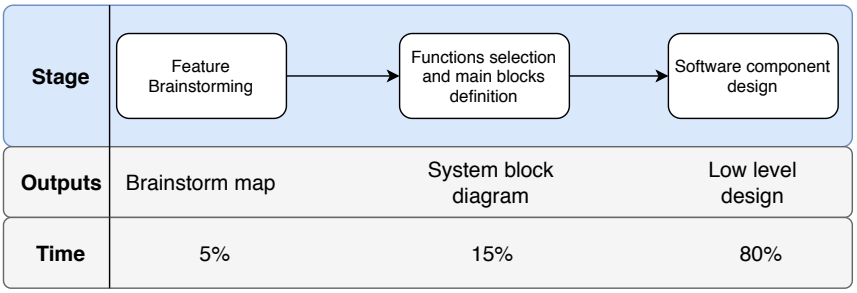


Figure 5.1: Software design procedure.

The designed system has 4 different blocks. Every block contains several software components. The blocks are System, Safety, Motor Control and User Interface, as seen in figure 5.2. After this introduction a more detailed explanation of every block will be performed later.

- System consists of modules which are necessary for the system to work, like the system initialization or modules that are used for debugging, like the UART interface.
- Safety is the block in charge of monitoring that the system as a whole is behaving as expected. If that is not the case, proper reactions must be done.
- Motor Control contains the modules which perform the actual controlling of the inverter by generating the duty cycles that will drive the motor as desired.
- User interface gathers the user input from the Interface PCB, the pedal or the Graphical User Interface and transforms that into a reference that will be fed into Motor Control

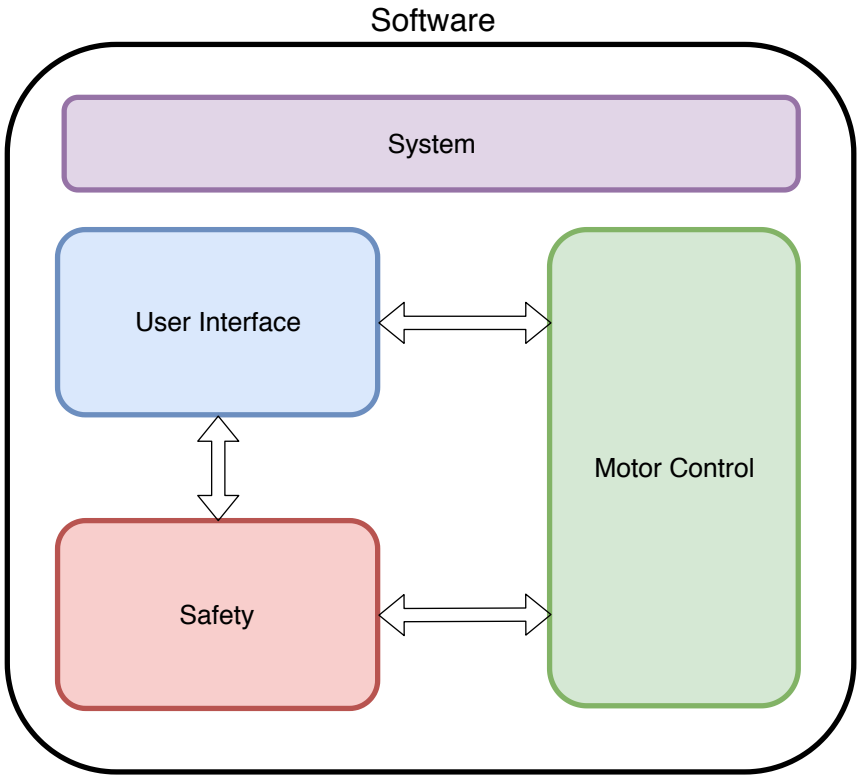


Figure 5.2: System main building blocks.

remember. AT

The Low Level Design output consists of a UML diagram. As seen in figure fig:.

In this UML diagram, every box represents a single software component, which will be developed in a single source code file. The main variables used and all the functions to implement can be seen.

Check this! AT

–>ADD UML and discuss.

5.1.1 Timing considerations

It is very important that motor controller has a precise execution frequency, as it performs calculations which consider the time. In addition, there are some tasks desired to run more often than others. A detailed design of the priority of the tasks and their execution frequency was performed. The result can be seen in figure 5.3. The color coding follows the one used in the software introduction.

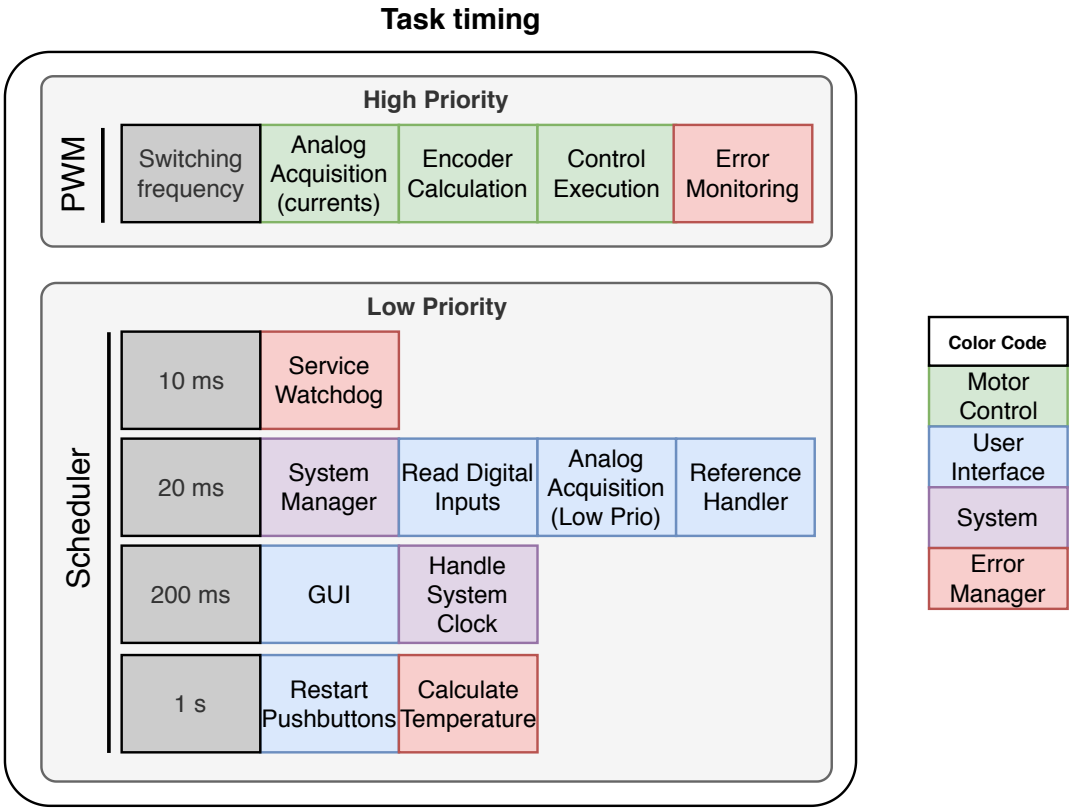


Figure 5.3: Timing design of tasks.

The implementation was performed by using the PWM as a trigger in the high priority tasks. The high priority tasks are those which must be ran every switching cycle and might contain calculations which include time as a variable. For the low priority tasks, the tasks are called by the Scheduler. The Scheduler implementation is described in section 5.2.2. In order to validate whether the tasks fit in the designed time span, the task's time consumption was measured. A table with all the results and further discussion can be found in section 5.6.1.

5.2 System block

The block System contains critical information for the correct operation of the control software. It is made of modules related to time managing or to the correct flow of commands.

System is the most unique block of the four, its main focus is the organization of the instructions that the DSP is working on at each moment. It works both in timing and in organization of the main state machine. For this reason it features the following modules:

- **System manager**, the main finite state machine of the program, section 5.2.1.
- **Scheduler**, organizes low priority tasks and calls the tasks at the correct periodicity, section 5.2.2.

What you write to each subsection is fine, but it's difficult to get the overview of how all the block and variables relate to each other. Maybe a flow-chart or similar drawing could help on that. Erik, This comment is for the first version when thinks werent complete but I think it is clear enoigh now. anyways let's see if more flow charts are needed to make it more clear.Stef

I think this whole paragraph is not saying anything.NM

I think that this is not explaining what system manager does

- **High priority tasks' execution**, ensures precise timing of high priority tasks, this tasks need to be called at switching frequency and with high priority, section 5.2.3.

5.2.1 System manager

The system manager is a module which goal is to oversee the system. It is based in a finite state machine that consists of four states; startup, standby, running and error. This is shown in figure 5.4.

After the system is initialized and startup is completed, the program is switching between the modules of standby and running, according to the input from the user. In standby, the inverter is switched off and the system is in hold. It is only in the running state that the inverter is enabled.

When an error is found the system rapidly goes into the error state, where the inverter is switched off to prevent further damage. Errors include overcurrent and overvoltage, but also an excess in the temperature of the inverter and batteries.

Only when the user has acknowledged and fixed the error, the state machine goes back to standby, and normal functionality is reestablished.

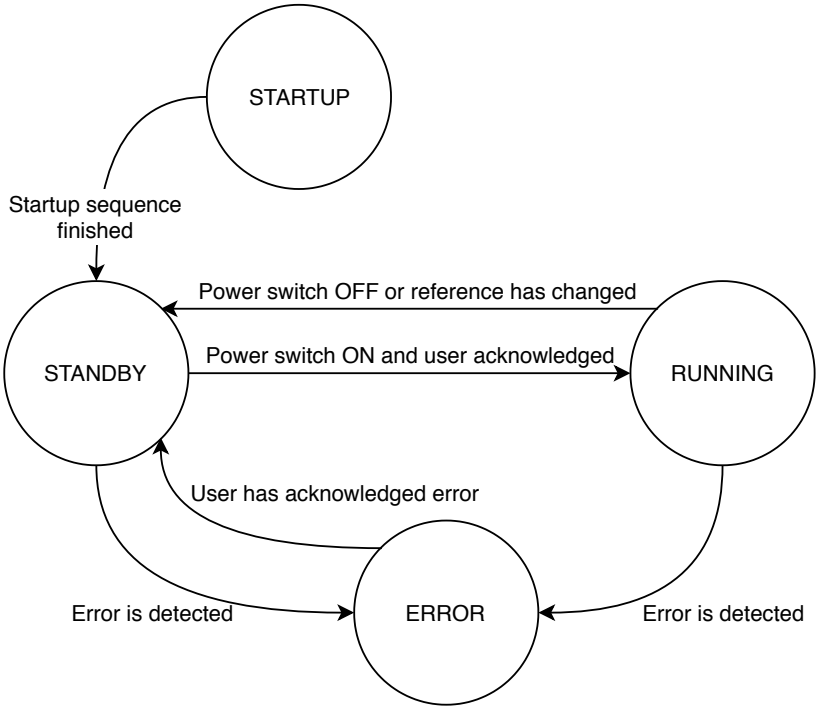


Figure 5.4: Finite state machine of the system manager.

5.2.2 Scheduler

As explained in the architectural design, the Scheduler is the module used for managing the execution frequency of the low priority tasks. In this section, the Scheduler, its main challenges and relevant pieces of code are discussed.

The system must perform several tasks, some of them are ran every few milliseconds, like servicing the watchdog. Other tasks only have to be ran once per second, like the temperature calculation. The Scheduler is a framework which allows the designer to change the execution frequency of the modules.

A scheduler must have a Task Control Block or TCB. This TCB includes the function pointer, the state, the cyclicity and the countdown until the next execution, for every task. So every task has a TCB of its own containing this essential information.

```
1     taskList[task5msItem].functionPointer = task5ms;
2     taskList[task5msItem].taskState = INACTIVE;
3     taskList[task5msItem].cyclicity = 5000;
4     taskList[task5msItem].timeLeft = 5000;
```

Listing 5.1: Task Control Block in 5 ms task.

The implementation has two main blocks. First there's an infinite loop analysing whether a task is ready to be ran. If that is the case, the task is executed. On the other side, the CPU timer is triggered every millisecond and the tasks' state is changed to READY when its countdown reaches 0.

```
1  void scheduleTasks(void)
2  {
3      Uint16 taskListIndex = 0;
4      for(;;)
5      {
6          for (taskListIndex = 0; endOfTaskListIsReached(taskListIndex);
       taskListIndex++)
7          {
8              if (taskIsReady(taskListIndex))
9              {
10                 runTask(taskList[taskListIndex].functionPointer);
11                 deactivateTask(taskListIndex);
12             }
13         }
14     }
15 }
```

Listing 5.2: Infinite loop which checks whether a function is ready to be ran and runs it if that is the case.

```
1  void updateTasksState(void)
2  {
3      Uint16 taskListIndex = 0;
4      for (taskListIndex = 0; endOfTaskListIsReached(taskListIndex);
       taskListIndex++)
5      {
6          if (taskMustBeScheduled(taskListIndex))
7          {
8              setTaskReady(taskListIndex);
9              restartTaskCountdown(taskListIndex);
10         }
11         else     decreaseCountdown(taskListIndex);
12     }
13 }
```

Listing 5.3: Function called by CPU timer to check if every task's countdown reached 0.

A module like the scheduler is necessary in a complex system where many developers interact, as the execution of tasks must be properly organized. The system also becomes easier to review, as the cyclicity of every module can be analysed at a glance. A limitation of the current implementation is the fact that there is not any kind of time sharing method. So if any task grew too much it will block other tasks' execution, eventually leading to a watchdog reset. In larger systems with a higher amount of tasks, a CPU time sharing technique becomes mandatory. Another limitation of the scheduler is the absence of priorities in the tasks, then the buttons' handling is given the same relevance as the reference calculation. Developing such system which considers the priority of every task might increase the reliability of the system. This additional features, priorities and time sharing, would raise the scalability of the system, allowing it to grow without a limitation on the task management side. However, this features would lead to increased overhead and thus a decrease of the useful computation time, as additional time will be spent on context switching among tasks. [14]

5.2.3 High priority tasks

In section 5.1.1 it was stated how some tasks have a higher priority than others and are triggered at PWM frequency. The high priority tasks consist of those in charge of running the control loop as well as monitoring critical errors. Running the control entails three main things; obtaining the values of the currents, calculating the position of the rotor and rotor magnetic flux vector and execute the control strategy (FOC or VF control). The currents are the most critical values to check since current peaks could damage the inverter in a short time. Therefore, these errors are monitored at switching frequency and are also a part of the high priority tasks.

An accurate measure of the currents is very important for having a reliable control feedback. For this reason, they are measured at the cycle point where the instant value of the currents is equal to its average. As seen in figure 5.5, this moment is at the mid point of the high or low parts of a PWM cycle.

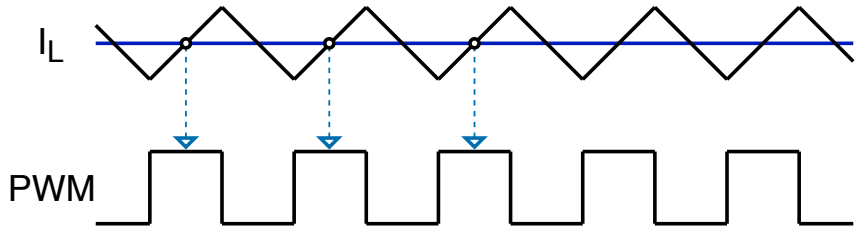


Figure 5.5: Representation of current behaviour with respect to PWM signal of one leg, in order to show the moments where it is measured.

An interrupt is triggered when the PWM signal is half way through the high section of the period, ensuring consistent and accurate currents measurements.

The triangular carrier is used for triggering the PWM signals, this carrier is functioning continuously, no matter if the PWM is sent or not to the inverter. Also, this triangular carrier signal is reliably consistent in its period, meaning that the

time between two triggers is always constant. This is useful for calculating rotor speed, since it is obtained by computing the change in position divided by the time passed, thus this time must be very accurate.

The order of execution of the tasks is then:

1. Measure the current values, ensuring it is measured at the correct moment.
2. Obtain rotor position and speed, time passed is always consistent and depends on switching frequency.
3. Run control, the time passed is still important but not as critical as for calculating the speeds.
4. Manage errors, final task to be performed when everything has been measured and computed.

I think this section is nice but I also think there's a bit of overlap with motor control intro, what do you guys thiink. NM

5.3 User interface

The Go-Kart must be in constant communication with the user since it receives information in order to update the reference parameters. Also, the user must be able to get information from the program to understand the current status.

Three different paths have been developed in order to communicate with the user, these are the interface board, a graphical user interface (GUI) and the throttle pedal.

- **Interface board**: The interface board features many different communication channels: switches, buttons and potentiometers as inputs and LEDs to show software status. It also has external connectors to receive information from sensors. Section 3.4.
- **GUI**: The interface board can have limitations, mainly with the information that it is able to send to the user. The lack of a display does not allow to show numerical information for speed or temperatures, also references can only be set to an approximated value with the built in potentiometer.

  For these reasons, a GUI was implemented, allowing an easy display of all the variables as well as the opportunity to set very precise references, section 5.3.1.
- **Pedal**: The throttle pedal is an extension of the interface board. It can replace the potentiometer at the board when the Go-Kart is being driven.

The interface board and the GUI can be displaying data at the same time. However, they cannot set references simultaneously. For this reason, a button in the interface board and another one in the GUI have the purpose of changing between the different input references. Going from the board, to the pedal, to the GUI and back again. When this happens, the motor is stopped and the user needs to acknowledge the change, making sure that a different reference value is not set by mistake.

The main modules of this section are related to data acquisition and display. These are the analog acquisition manager (section 5.3.2), digital input manager (in charge of managing switches and buttons) and the reference handler (section 5.3.3).

5.3.1 Graphical User Interface

A graphical user interface has been created to interact with the user. The goal of the interface is to have a simple plug-and-play way of showing information to the user, it is a way of testing that the interaction with the DSP would be possible without the buttons and potentiometers from the Interface PCB in the next iteration of the system. The available information is:

- General status of the system showing whether an error has been triggered, the system manager's FSM state and the status of the power enable switch.
- Input reference window, where the user can select the torque or speed reference and also the reference rate of change used by the control algorithm.
- Error status window, where every error source has a specific LED, providing an immediate way of knowing the failure reason.
- In the graphs and gauges, variables like rotor speed or MOSFET temperature, in addition to the actual currents as seen by the controller are graphically shown.
- In the reference handler window the actual reference fed into the controller can be seen.

The interface was created with *Texas Instruments GUI Composer* and a snapshot can be seen in figure 5.6. The main issue found is the low information frequency of the system, as the variables are updated just a few times per second. This leads to borderline performance in the case of plotting phase currents. Better performance might be achieved by changing the communication protocol or using another GUI Composer Widget which allows an array as an input. The GUI Composer needs the variables to show to be global. The whole software was designed avoiding global variables so a specific software component gathering the variables' value from other software components was necessary. This new SWC called Communication Manager uses the interfaces provided by the other modules to acquire the values that might be interesting for the user, and uses other interfaces to react to the user's input to the GUI. A drawback of the framework provided by the GUI Composer is the fact that every time that a new binary is created, it must be loaded to the GUI project. This is a technical requirement as the GUI Composer needs to map the variables to the memory. However the GUI will be most used when the software has reached a high maturity level, where this handicap will not disturb. The GUI has proved itself a useful approach for data communication when the software development is finished. However, the fact that small changes require a significant amount of time makes the GUI impractical for debugging most of the modules.

did we get to do this??

Is 'SWC' explaind before this, NHF.

This doesn't feel like a natural way to stop the section. Maybe add why the mapping is a drawback, NHF. What do you think now with the final paragraph.NM

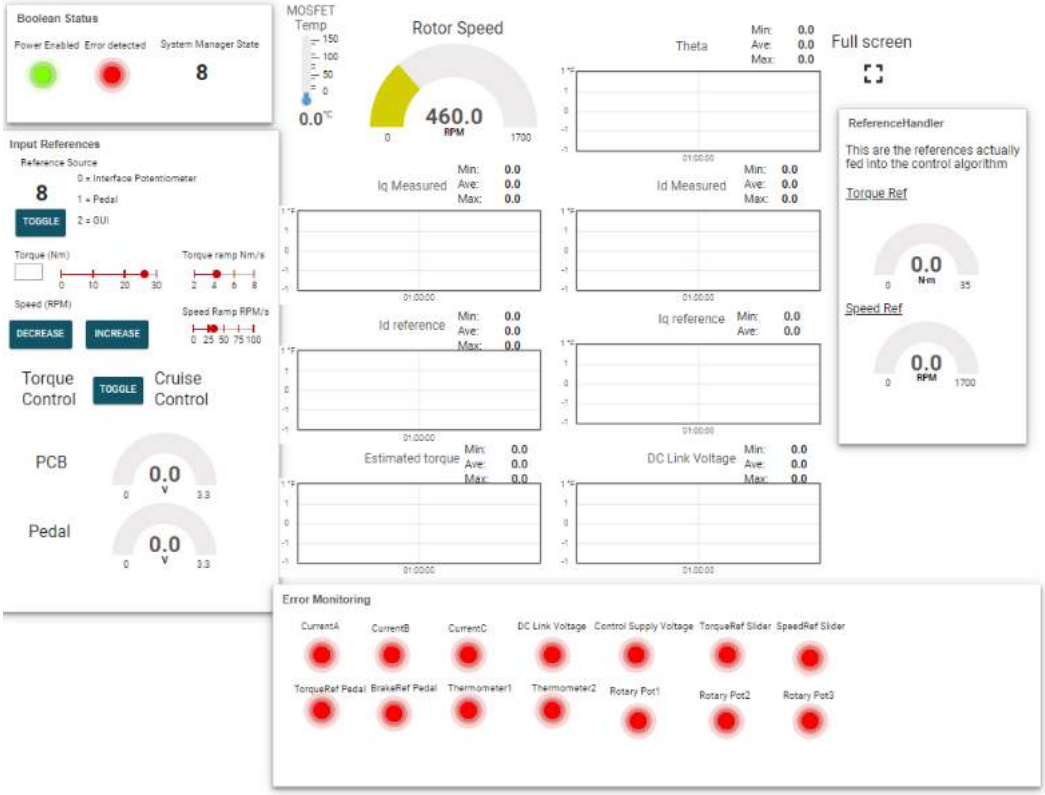


Figure 5.6: Graphical User Interface.

5.3.2 Analog acquisition manager

A big part of the base of the software is the analog acquisition manager. This block initializes the ADC's of the DSP and handles the measured values, for the use of upstream blocks.

The signal object

A 'Signal' struct type is created to handle the data for every analog signal in the system. To make the software more versatile and easy to adapt to the specific signal of interest, this struct includes selective parameters. These are parameters like ADC channel and cut-off frequency for the filter. Regarding the sampling, both the current value, the value at lag one and the filtered value are saved. Lastly, maximum and minimum thresholds for the signal are set to monitor the functionality of the system.

Two different structs are created to contain the analog signals, one with the currents and one with the rest. This is done to minimize the conversion time for signals within the time critical control loop. These structs will later be mentioned as the high and low priority signals, with the currents being the high priority.

ADC settings

For every signal three registers needs to be configured: ADC channel, trigger selection and acquisition period. The code snippet below shows an example of the setting of these registers.

The channel is received from the struct and is defined when creating the signal. The trigger defines when the conversion starts. The trigger source can be selected to be a CPU timer, a PWM signal or done by software. As PWM signals will be generated for the control, one of these is also used as the trigger. This ensures that new measurements are ready for every switching period. The acquisition period defines the the amount of clock cycles used in the sample and hold window. The total time to process the signal is then defined as the sum of the sample window and a constant conversion time of 13 cycles. Examples from the technical manual of the DSP are shown in table 5.1. [15, p. 492]

```
1    // Configure ADC for phase A current measurement
2    AdcRegs.ADCSOC3CTL.bit.CHSEL        = CurrentSignalList.currentMeasA.
     adcChannel;
3    AdcRegs.ADCSOC3CTL.bit.TRIGSEL      = TRIGGER;
4    AdcRegs.ADCSOC3CTL.bit.ACQPS        = SAMPLING_RATE;
```

Listing 5.4: Example of setting the ADC registers

ADC Clock	ACQPS	Sample window	Conversion Time	Total time to process
$45MHz$	6	$155.56ns$	$288.89ns$	$444.44ns$
$45MHz$	25	$577.78ns$	$288.89ns$	$866.67ns$

Table 5.1: Timing table for acquisition period selection

Signal acquisition

Reaching EOC(End-Of-Conversion) of the sampling, triggers an interrupt in the ADC block. When this interrupt is received, the analog signals are read from the ADC registers. To optimize the time consumption of the signal acquisition, only the high priority signals will be read at this point.

The total signal acquisition is executed by two commands, *readAnalogSignals()* and *calculateFilteredValues()*. Both commands takes a list of signals as inputs. They iterate through the list by the use of a pointer, to read the ADC registers and filter the readings respectively. The function *readAnalogSignals()* is shown below. It takes either the high or low priority list and the size of it as input arguments. By knowing the first and last position of the list, it is possible to iterate through it and update the value in the position of *structPointer*.

```
1 void readAnalogSignals(void *signal, int size)
2 {
3     AnalogSignal *structPointer;
4     AnalogSignal *initialMemoryPosition = signal;
5     AnalogSignal *finalMemoryPosition = initialMemoryPosition + size/
      sizeof(AnalogSignal);
6
```

```
7     for (structPointer = initialMemoryPosition; structPointer <
      finalMemoryPosition; structPointer++)
8         readADCValue(structPointer);
9 }
```

Listing 5.5: Function to read analog signals from ADC

Digital filter

To assure smooth signals without ripples and reducing the noise, a digital filter was implemented to filter the ADC readings. Implementing a digital filter makes the system more agile, as it can be removed if the hardware filter is sufficient. A first order low-pass filter was designed with an adjustable cut-off frequency. Equation 5.1 shows the calculation for the filtering.

$$y_n = x_n \cdot a_0 + y_{n-1} \cdot b_1 \tag{5.1}$$

The parameters a_0 and b_1 can be calculated from the execution frequency and desired cut-off frequency of the filter. The equation uses both the current reading and the previous filtered value, to calculate the current filtered value. The two parameters is calculated as in the following equations.

$$b_1 = e^{-\frac{1}{f_{filter} \cdot \tau}} \tag{5.2}$$

$$a_0 = 1 - b_1 \tag{5.3}$$

Where f_{filter} is the cut-off frequency and τ is a function of the execution frequency. [16]

Interfacing with other modules

For the use of the ADC readings in other modules, interface functions have been created. The purpose of these functions is to have an easy way of receiving the value in the correct unit of the measured signal. Then, although the temperature sensor has a voltage as an output, this interface will directly return the actual temperature. This will also ease the use of the ADC readings in other software blocks, as they also are a fundamental part of the *Safety* and *Motor control* blocks.

The Interface functions are divided into two functionalities: updating the signal lists with new readings, and receiving the readings from the list. The functions *readHighPrioritySignals()* and *readLowPrioritySinals()* should be called to update the signal lists. These are simple to use and clearly tells the user the priority of the signals. To receive the readings from the lists, *get*-functions have been created. These include the transfer function of the sensor circuits, to ease the use. The function to get the value of the DC-link voltage is shown below. It takes the filtered value, apply the transfer function and then subtracting an offset found by calibration. This simplifies the code and makes it cleaner where the measurements are being used.

```
1 float getDCLinkMeasurement(void)
2 {
3     return (AnalogSignalList.voltageMeas36.filteredValue *
      DC_LINK_MEAS_TO_VOLTAGE) - DC_LINK_OFFSET;
```

```
4 }
```

Listing 5.6: Function to receive the DC-link measurement from the signal list

5.3.3 Reference handler

The goal of the reference handler is to check what reference source is selected by the user. Once the source is known, the value of that reference must be obtained.

The possible reference sources are the pedal, the interface board and the graphical user interface. Also, open loop control (OL) or close loop field oriented control (FOC) can be selected and finally, when working in FOC, torque reference or cruise control are allowed.

The reference source can be selected from the interface PCB, obtaining references from the analog input manager and sources from the digital input manager, or everything can also be selected from the GUI.

If cruise control is selected, the speed can be updated digitally, either from the GUI or from the PCB. Here the rotor speed is required for getting the new reference, this is obtained from the control block.

When the torque reference has been obtained and with the goal of limiting large reference differences, a smooth adjust on the reference is performed, not allowing a reference to change suddenly, this is done in "Reference delta limiter".

Finally, in order to avoid references over the technical limits of the system, they are limited to a preset value in "Saturation limit".

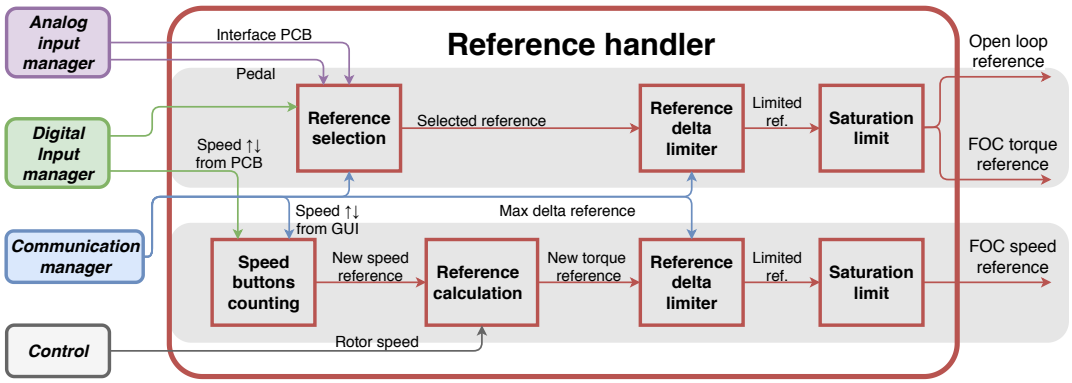


Figure 5.7: Reference handler work flow.

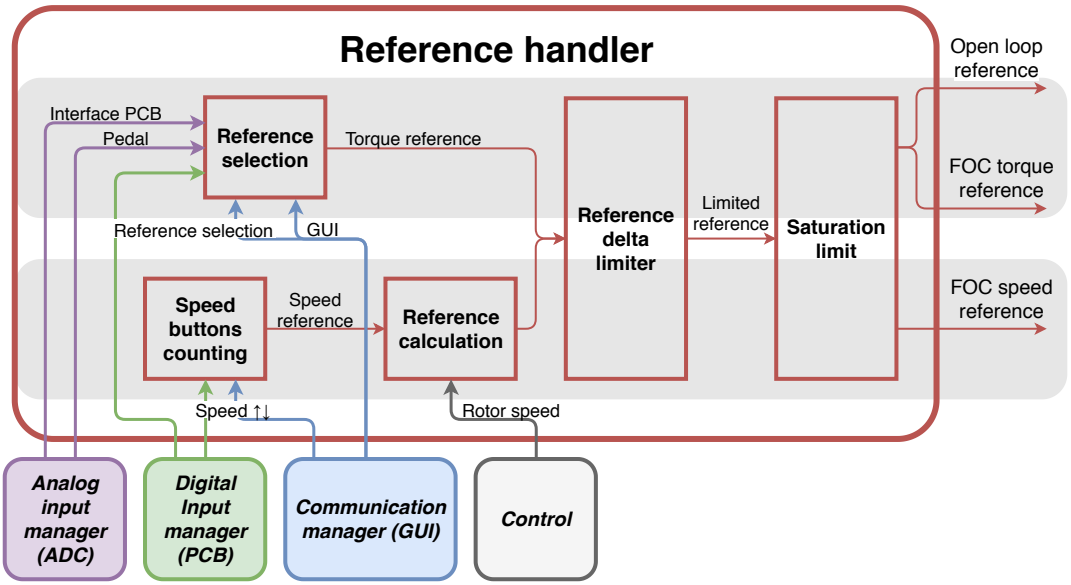


Figure 5.8: Reference handler work flow.

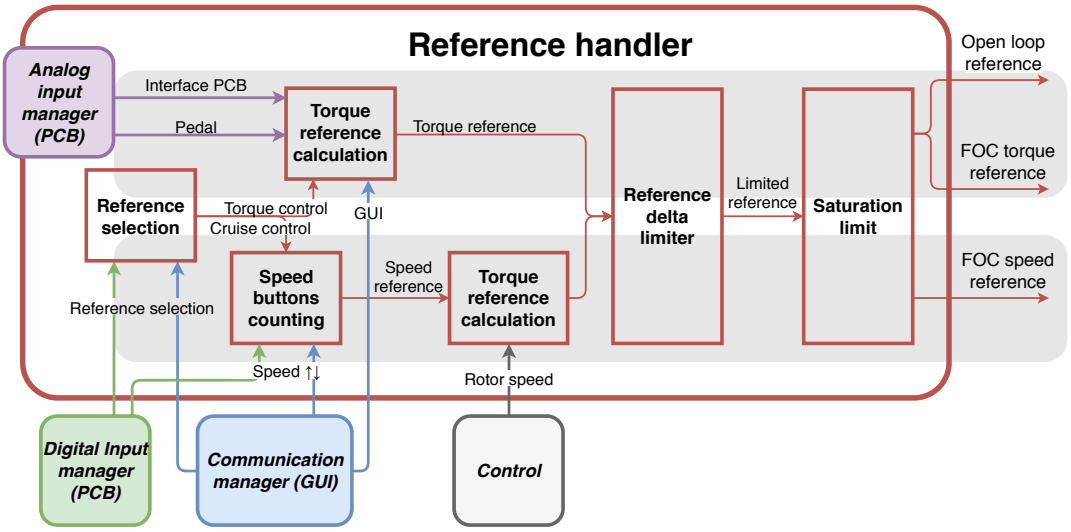


Figure 5.9: Reference handler work flow.

Which image is easier to understand? Make suggestions!! AT

5.4 Motor control

The motor control block includes the set of modules which have the user reference provided by the Reference Handler as an input and calculate and set the duty cycle as an output. This block has requiring time constraints as the delay between the ADC sampling and the duty cycle update will directly affect the control performance. For accomplishing the time requirements, code optimization techniques like removing divisions, have been done. In this section, θ calculation and the

list the optimization techniques

closed loop control modules are explained.

5.4.1 Position estimator

The control algorithm developed for the induction machine requires position and speed feedback from the motor. An encoder is attached to the rotor shaft, acquiring accurate information about the rotor position, speed can then be calculated.

There are three software modules that calculate rotor information. One module obtains both position and speed from the encoder measurements, another one estimates the rotor flux angle and its speed and finally the third module acts as a manager of these two and it is in charge of communication with the rest of the software.

Incremental quadrature encoder

The encoder used to obtain information about the motor shaft was selected during previous work [3]. It is an incremental quadrature encoder with 2048 steps. It also features index input to determine absolute position.

Incremental quadrature encoders have two channels, A and B. These channels provide square signals shifted 90º from one another. This way it is possible to obtain direction as well as angle difference. However, minor errors could pile up over time, introducing long term errors in the rotor position, to avoid this issue, the index term is included. This term provides one pulse for every lap of the encoder, when this happens position is reset assuring absolute precise position estimation in the long term.

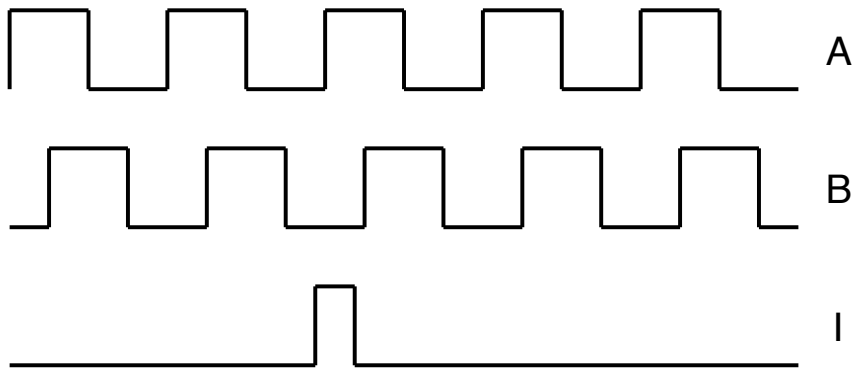


Figure 5.10: Signals produced by a general incremental quadrature encoder, including index.

In figure 5.10 the signals produced by an incremental quadrature encoder are shown.

Position estimator procedure

The information obtained from the encoder is the base to calculate four different parameters. These are position and speed of the rotor shaft as well as direction and speed of the rotor flux.

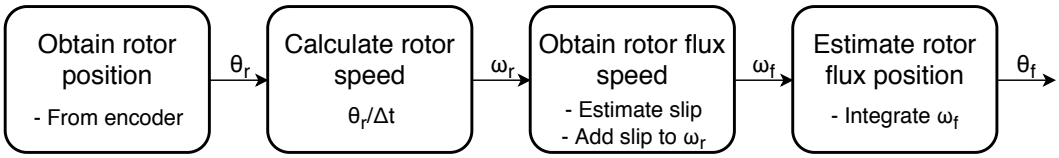


Figure 5.11: Encoder procedure for obtaining rotor parameters.

Figure 5.11 shows how position and speed related information are obtained.

The rotor position is directly obtained from the encoder signals. The encoder measures mechanical position of the rotor, which is then translated to electrical taking into account the number of poles.

After rotor position is obtained, its speed is calculated. For this, every 10 electrical degrees, speed is obtained. However, at low RPM, measuring every 10 degrees could lead to inaccurate results. Thus, if more than a certain time has passed since speed was last obtained, it is calculated anyway. This way, it is ensured that reliable values are measured both at high and at low speeds.

The rotor flux speed is the summation of the rotor shaft speed and the slip speed. An approximation is then performed to calculate this as shown in equation 4.33.

However, obtaining the instant direction of the rotor flux is not as trivial. The speed must be differentiated and an approximation of the direction would be obtained. The accuracy of the measurement could contain errors. Nevertheless, since the calculated value of the flux position is then used as a reference for controlling the machine, it should adjust its behaviour making minor errors unimportant.

5.4.2 Closed-loop control manager

The modules of the *Motor control* block are connected by the closed-loop manager. This is where the simulated FOC algorithm from section 4.5 is implemented. Further reading about the FOC can also be found in that section.

The flow of the closed-loop manager is shown in figure 5.12. The first step is to collect the reference values and the measured 3-phase currents, from the reference handler and the analog acquisition manager respectively. Then the currents are transformed into the dq-referenceframe, to be used in the PI-controllers. The next step is to transform the voltage references from the PI-controllers into the $\alpha\beta$-referenceframe. These can then be used as references for the SVM algorithm, which will generate the necessary duty-cycles for the three legs of the inverter.

Determine how much we should describe in this section. As all of it is already described in the control chapter, i think this should be at a minimum, . I think It's OK, we spent time in the implementation but I think describing the PI implementation for example might be a bit too much. NM

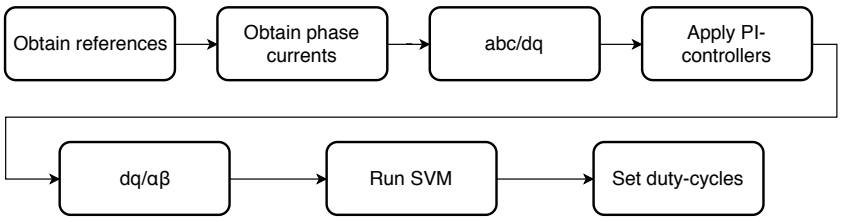


Figure 5.12: Flow diagram for the closed-loop control algorithm

5.5 Safety

To improve reliability and to increase safety, the error recognition and management must be carefully considered. This is specially important when working with high power devices where errors could entail major issues concerning the equipment or even the user.

The possible errors include overvoltage, overcurrents and high inverter and battery temperatures. The software also features a watchdog which force a restart if an unexpected behaviour is found or if the system is caught up in a loop. However, checking errors is a very time consuming task since these mainly come from the analog signals. For this reason, the most critical errors, overcurrents, are checked at switching frequency, while all the others, are checked from the scheduler at 1kHz.

5.5.1 Error manager

I think it would be nice to mention the reaction time and how it was measured, NM

The error manager is a crucial part of the safety feature in the software. It keeps track of the flow from error detection to execution of the necessary safety reactions. This flow is shown in figure 5.13. It shows that error monitoring can be disabled. This is a useful feature to avoid error triggering if some sensors are not connected. If an error is detected, the error manager performs the safety reaction, and switches to error mode. The safety reaction includes disabling the PWM drivers and turning an LED on to indicate that an error has happened. The error mode can only be exited by the user, so the error source can investigated before starting the control again. The error status of every signal is latched, such that the errors can be shown through the GUI and the UART for debugging. When the user acknowledges the error to return normal operation, the signals error status are reset, the drivers are enabled again and the LED is turned off.

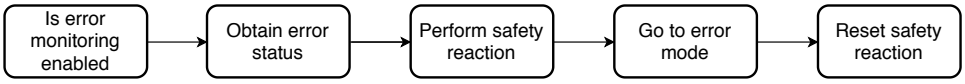


Figure 5.13: Flow diagram for the error managing algorithm

Obtaining the error status

The first step of the error manager is to obtain the error status of the analog signals. The code snippet below shows the implementation of this. By the use of pointers, it compares the measured value with the defined thresholds. If the value is outside these thresholds, the algorithm sets the corresponding bit equal to 1. This is continued through the entire struct, to generate a bit array. This array is then checked for 1's, to check if an error has happened.

```
1 Uint16 getHighPriorityErrorStatus(void)
2 {
3     AnalogSignal *structPointer;
4     AnalogSignal *initialMemoryPosition = &CurrentSignalList.currentMeasA;
5     AnalogSignal *finalMemoryPosition = initialMemoryPosition + sizeof(
      CurrentSignalList)/sizeof(AnalogSignal);
```

```
6     Uint16 errorStatus = 0;
7     int i = 0;
8
9     for (structPointer = initialMemoryPosition; structPointer <
      finalMemoryPosition; structPointer++)
10    {
11        if (structPointer->filteredValue < structPointer->threshold[0] ||
              structPointer->filteredValue > structPointer->threshold[1])
12            errorStatus |= 1<<i;
13
14        i++;
15    }
16
17    return errorStatus;
18 }
```

Listing 5.7: Function to get the error status of the three current signals. It Checks if the measured value is outside the user defined thresholds. If this is true the value of the corresponding bit is set to 1.

5.5.2 Watchdog timer

The watchdog is an important component in the error detection of the system. It is a peripheral with a clock independent from SYSCLOCKOUT, which is the clock used by the CPU, the DSP must acknowledge periodically the watchdog. If the acknowledgement is not performed, the watchdog will restart the DSP. This is a very useful feature to prevent the system to get stuck in an infinite loop, or if the DSP freezes. This is also a way to monitor whether all the scheduler tasks are being run and completed, as servicing the watchdog is the last task performed by the scheduler every $10ms$.

The time for the watchdog timer to overflow is defined as the watchdog counter clock. Here the default value is used, which is calculated as $OSCCLK/512/1$, where $OSCCLK$ is the system defined oscillation clock. [15, p. 107]

5.6 Validation

The validation phase goal is to determine whether the system is behaving as it was designed to. This is the final step after design and implementation and is of critical importance to achieve a reliable system. In this section the techniques used and some of the results will be discussed.

5.6.1 Task Timing

A main concern in real time systems is how often and how are the tasks executed. This was firstly introduced in section 5.1.1, where the tasks' timing behaviour was designed. Now it is desired to test whether the functions are executed with the proper cyclicity. For that, when the function was entered, a digital output was set to a high logic level and when the function was left the digital output was set to a low logic level. This signal was monitored with the oscilloscope. Setting the signal itself already takes some time, although very small: it was measured to be

$0.1\mu s$. That time is considered in the summary table 5.2. One of the concerns was to fit all the high priority tasks within $100\mu s$, which would be necessary if it is desired to run the control every switching period, at $10kHz$. The table shows that the total execution time of the high priority pipeline, without error monitoring is $48.2\mu s$. This means that the algorithm is able to execute the control with only half a switching period delay. The gain by minimizing this delay is faster response time in the control, and therefore smaller ripple in the currents.

High Priority pipeline	
Task to be performed	**Time**
Read and filter signals	**8.7μs**
Read signals	2.9μs
Filter signals	5.8μs
Calculate rotor position and speed	**7.6μs**
Calculate position	1.2μs
Calculate speed	2.2μs
Calculate rotor flux position	4.2μs
Execute closed-loop control	**31.9μs**
Read references and currents	8.1μs
ABC to DQ conversion	6.2μs
DQ to $\alpha\beta$ conversion	2.4μs
Apply PI-controllers	6.9μs
Run SVM	8.3μs
Perform error monitoring	**-**
Without error	26.2μs
with error	6.1μs

Table 5.2: CPU consumption time by tasks. SYSCLOCKOUT = 90MHz.

5.6.2 Debugging framework

The debugging framework consists of the set of techniques available to validate whether the developed code is working as it is intended to. If the developer doesn't have the proper tools for debugging, the code implementation might be slowed down or it might not be possible to check if the system is working properly. Thus, although creating additional scripts for validation might seem like an unnecessary or an overkill task, it is important that the management team evaluates what testing framework will be necessary for the system under development. In our case, most of the tests could be performed using on-chip debugging, where breakpoints can be set and the variables' value can be read. However, the Motor Control module includes time dependent signals and it is desired to analyse whether these signals evolve with time according to expectations. This was an especially critical module to validate, as misbehaviour could lead to damaging the hardware. During the control design phase, a Simulink model was used. There, the designer has access to every variable with respect to time, so a module which could replicate that practice was designed. The used developer kit includes a COM port within the USB connection, so using DSP's UART hardware over Serial communication was

the easiest option.

For implementing the serial communication, C features the widely known *printf()* function, however that function is highly memory and CPU demanding. So a module for using the UART hardware was implemented. A thing to consider was that the UART hardware buffer is relatively small and thus a software buffer is necessary if a high data rate is desired. A software buffer was implemented by a Circular Queue. If this software buffer was not implemented, it would be necessary to explicitly wait for the UART to send the messages. This wait results in a very limited communication framework. The developed module allows the user printing integers with a title. During execution, a timestamp is printed every 200*ms*, its goal is to allow the event analysis with regard of time. An example of the UART use can be seen in the snippet below, where the *Timestamp* is printed.

```
1    // UARTIntPrint([Title ], (int)[variable]
2    UARTIntPrint("TS ", (int)sysClock);
```

Listing 5.8: Serial communication example use: TimeStamp printing.

When these code is ran and the serial communication port is analyzed from the computer, the information can be collected. The result can be seen in figure 5.14. In the picture, aside of the *Timestamp*, the current in phase A is also printed. Although the data is available, it is necessary to provide a way to understand it, as the data will easily reach a size of hundreds of points. The chosen option consists of a Python script which has a text file as an input and generates the graph for every different graph for every different Title. Python was elected as it has powerful string processing capabilities. Also, in the case that real time plotting of the data is desired, there is plenty of literature available in the topic and the graph generation can be used as a base for a larger script. An example of the generated graph can be seen in figure 5.15. In the figure, the reference frame conversion was being validated. It's worth mentioning that even though in this case, the data frequency of the Timestamp and the variables under analysis that is not the mandatory.

```
TS 293,
PhaseA 98,
PhaseB 15,
PhaseC 37,
D 0,
Q 49,
TS 294,
PhaseA 98,
PhaseB 14,
PhaseC 37,
D 0,
Q 50,
TS 295,
PhaseA 98,
PhaseB 14,
PhaseC 38,
D 0,
Q 49,
TS 296,
PhaseA 98,
PhaseB 14,
PhaseC 38,
D 0,
Q 49,
```

Figure 5.14: Serial communication example.

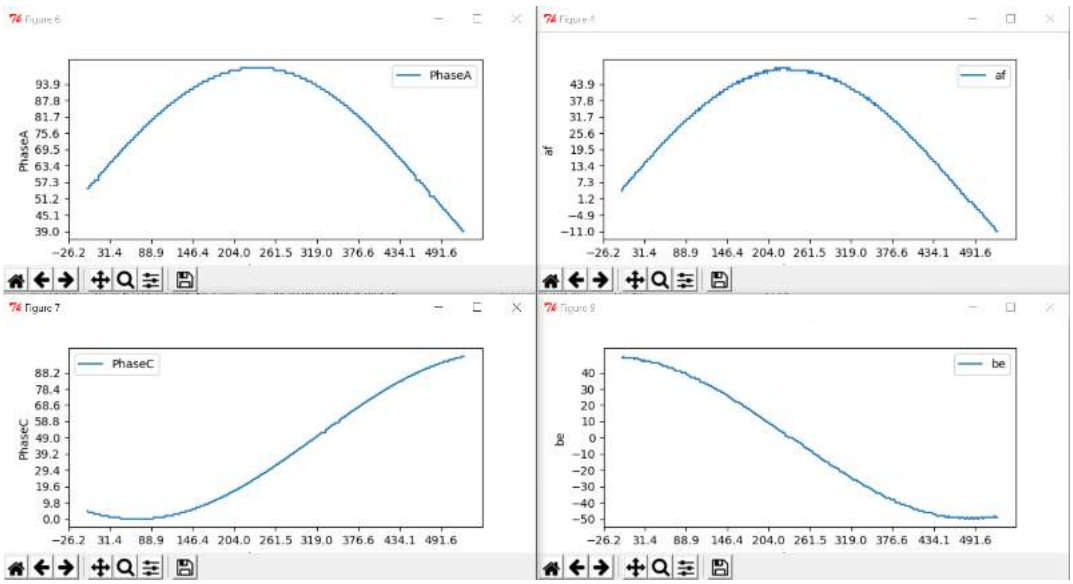


Figure 5.15: Offline plot of serial communication data. Test of reference frame conversion for currents.

Is it possible to add units to these plots + x-axis name, NHF

The limitations of the developed debugging system are divided into technical limitations of UART and serial communication and the fact that this feature was developed in a tight schedule. Regarding UART limitations, there are other communication protocols which could deliver a high data rate, but its hardware requirements are larger and the development time is also higher. On the other hand, as the development time was small, only integer communication is supported. De-

veloping a proper logger for supporting testing in the car would be the next step in validation and testing. Often, this systems use CAN communication, which is widely spread in the automotive industry.

6 Discussion

6.1 Integration test in Go-kart platform

6.2 Project management

6.3 Problems and limitations

6.3.1 Driver damage

6.4 Future work

7 Conclusion

In case you have questions, comments, suggestions or have found a bug, please do not hesitate to contact me. You can find my contact details below.

Jesper Kjær Nielsen
jkn@create.aau.dk
http://sqrt-1.dk
Audio Analysis Lab, CREATE
Aalborg University
Denmark

7.0.1 Key learning from this project

1. Hardware is tough
2. Software is tough
3. Life is tough

Bibliography

[1] *An industry based survey of reliability in power electronic converters*. Vol. 47. 2011, p. 1441.

[2] *Ensuring Success in Global Utility Solar PV Projects*. Tech. rep. 2014.

[3] J. Beer et al. *Development, Modelling and Implementation of an Electrical Drivetrain for a Go-Kart*. Student Project Report. Aalborg University, 2018. Chap. Inverter hardware. 116 pp.

[4] A. Kiep D. Graovac M. Purschel. *MOSFET Power Losses Calculation Using the Datasheet Parameters. Application note*. Tech. rep. Version V 1.1. Infineon, Automotive Power. 23 pp.

[5] Wenxin Peng. *Accurate circuit model for predicting the performance of lead-acid agm batteries*. Las Vegas: University of Nevada, Las Vegas, 2011.

[6] L. W. Yao et al. *Modeling of lithium-ion battery using MATLAB/simulink*. Vienna, Austria, 2013.

[7] T. M. Undeland N. Mohan and W. P. Robbins. *Power Electronics. Converters, Applications and Design*. Third Edition. John Wiley & Sons, Inc., 2003. 792 pp. ISBN: 978-0-471-22693-2.

[8] M. H. Rashid. *Power Electronics. Devices, Circuits and Applications*. 4th edition. Pearson, 2014. 1021 pp. ISBN: 978-0-273-76908-8.

[9] D. W.J. Pulle R. De Doncker and A. Veltman. *Advanced Electrical Drives. Analysis, Modeling and Control*. Springer, 2011. 438 pp. ISBN: 978-94-007-0181-6.

[10] V. G. Pagrut et al. *Internal model control approach to PI tunning in vector control of induction motor*. Mumbai India: International Journal of Engineering Research & Technology, 5 pp. ISRN: 2278-0181.

[11] M. Imecs, C. Szabó, and J. Incze. *Vector control of the cage induction motor with dual field orientation*. Cluj-Napoca, Romania: 9th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics. 12 pp.

[12] C. L. Phillips and R. D. Harbor. *Feedback Control Systems*. Ed. by M. Horton. 4th. Edition. Tom Robbins. ISBN: 0-13-949090-6.

[13] B. Fortman. *A faster current loop pays off in servo motor control*. Texas Instruments, 2017. 8 pp.

[14] S. Heath. *Embedded Systems Design*. Newnes, 2003. 451 pp. ISBN: 0-7506-5546-1.

[15] Texas Instruments. *Technical Reference Manual. TMS320x2806x Piccolo*. 2017. 1196 pp.

[16] S. W. Smith. *The scientist and Engineer's Guide to Digital Signal Processing*. 2nd Edition. California Technical Publishing, 1999. 688 pp. ISBN: 0-9660176-7-6.

[17] Lars Madsen. *Introduktion til LaTeX*. `http://www.imf.au.dk/system/latex/bog/`. 2010.

[18] Tobias Oetiker. *The Not So Short A Introduction to LaTeX2e*. `http://tobi.oetiker.ch/lshort/lshort.pdf`. 2010.

[19] Frank Mittelbach. *The LATEX companion*. 2. ed. Addison-Wesley, 2005.

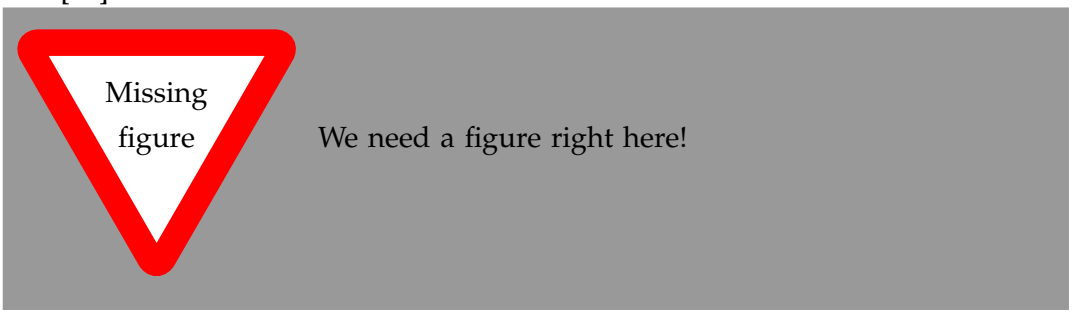
A

Space Vector Modulation

B Template

Here is chapter B. If you want to learn more about LaTeX, have a look at [17], [18] and [19].

I think this word is misspelled



Example of inserting a figure with a reference B.1:

Figure B.1: This is a nice looking go-kart

Example of inserting a table B.1:

Recommended ratings		
Supply voltages	V_{DD1}, V_{DD2}	5 [V]
Input voltage range	V_{in}	$0-2$ [V]
Other values of interest		
Voltage gain	G	1 [V/V]
Output common-mode voltage	V_{OCM}	1.23 [V]
Gain tolerance	–	± 3 [%]
Bandwidth	BW	100 [kHz]
Package	SSOP	[-]
Bandwidth	BW	100 [kHz]

Table B.1: Example of a table.

B.1 How Does Sections, Subsections, and Subsections Look?

Well, like this

B.1.1 This is a Subsection

and this

This is a Subsubsection

and this.

A Paragraph You can also use paragraph titles which look like this.

A Subparagraph Moreover, you can also use subparagraph titles which look like this. They have a small indentation as opposed to the paragraph titles.

Is it possible to add a subsubparagraph?

I think that a summary of this exciting chapter should be added.

Figure B.2 shows random stuff. This is done in chapter B.

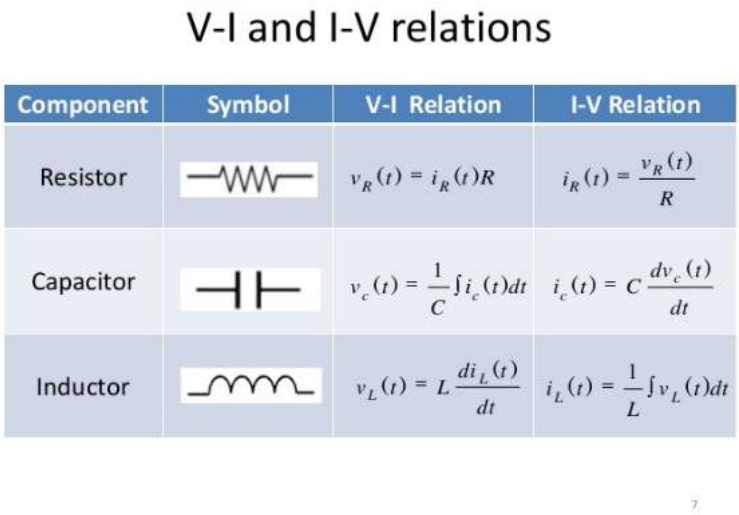


Figure B.2: This is a nice looking go-kart

$$\Delta = \frac{\pi}{\sigma \cdot 0.5} \tag{B.1}$$

You can also have examples in your document such as in example B.1.

Example B.1 (An Example of an Example)
Here is an example with some math

$$0 = \exp(i\pi) + 1\,. \tag{B.2}$$

You can adjust the colour and the line width in the `macros.tex` file.