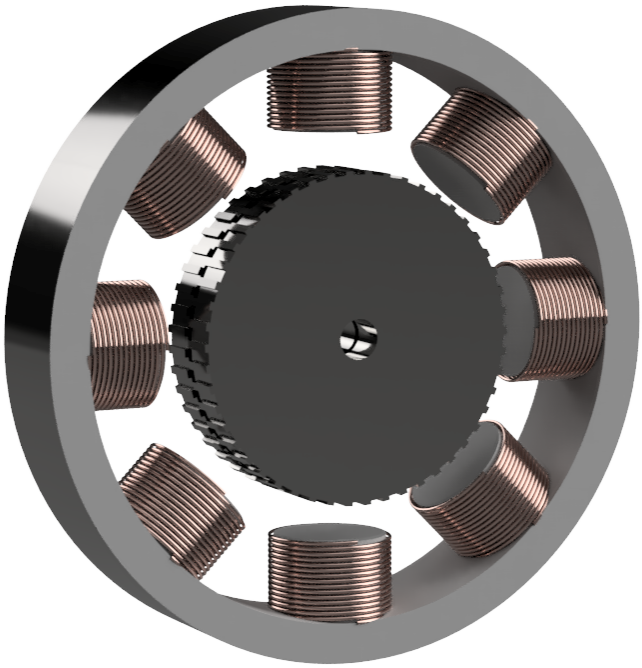


Model Based Control of PMSM using Field Oriented Control

Master's thesis in Systems, Control and Mechatronics

MICHAEL MARNE

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019:06

Model Based Control of Permanent Magnet Stepper Motors

using Field Oriented Control

MICHAEL MARNE

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2019

Model Based Control of Permanent Magnet Stepper Motors
using Field Oriented Control
MICHAEL MARNE

Supervisor: Mikael Bengtsson, Plejd and Jacob Andersson, Plejd
Examiner: Stefan Lundberg, Power Electronics

Master's Thesis 2019:06
Department of Electrical Engineering
Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Model of the principal structure of the stator and rotor of a PMSM, designed in Fusion360.

Typeset in LaTeX
Printed by
Gothenburg, Sweden 2019

Abstract

In this master's thesis report the feasibility of using model based control paired with field oriented control (FOC) for velocity and position control of permanent magnet stepper motors (PMSM) was investigated. PMSMs with a large number of poles (as most stepper motors are) are a challenging motor type for applying FOC due to the reduction in electrical rotor angle accuracy from the fact that the angle sensor measures mechanical angle which is divided into electrical angle. A FOC was first implemented in simulation along with a model based controller (Linear Quadratic Regulator, LQR) and a PID controller. After evaluation in simulation all controllers were implemented on the target hardware using the programming language C where their performance was compared to simulations. The FOC performed well at speeds under 10π rad/s with performance degrading at higher speeds due to a combination of implementation errors and lack of back-emf compensation. Both LQR and PID performed well in conjunction with the FOC, with the LQR outperforming the PID in both speed control and position control, despite speed limitations. The FOC was able to achieve a current rise time of 10 ms. The PID and LQR achieved velocity rise times of 0.06 s and 0.03 s respectively. For positional control the PID achieved a rise time of 0.2 s while the LQR achieved 0.15 s. The most suitable regulator to pair with the FOC is, if motor parameters are available, the LQR.

Acknowledgements

First I would like to thank my two supervisors, Mikael Bengtsson and Jacob Andersson at Plejd AB. Both have been very helpful, especially concerning the hardware implementation and changes to the hardware PCB to fit the needs of the project. I would also like to thank Iman Habib for letting me do this thesis at Plejd AB. Finally I would also like to thank Stefan Lundberg for being my examiner during this project.

Michael Marne, Gothenburg June 2019

Contents

List of Figures

List of Tables

Abbreviations

- **AC** - Alternating Current
- **ADC** - Analog Digital Converter
- **CNC** - Computer Numerical Control
- **CPU** - Central Processing Unit
- **DAC** - Digital to Analog Converter
- **DC** - Direct Current
- **FOC** - Field Oriented Control
- **GMR** - Giant Magneto Resistance
- **HW** - Hardware
- **ISR** - Interrupt Service Routine
- **KCL** - Kirchhoffs Current Law
- **LQR** - Linear Quadratic Regulator
- **PI** - Proportional Integral
- **PID** - Proportional Integral Derivative
- **PMSM** - Permanent Magnet Stepper Motor
- **PWM** - Pulse Width Modulation
- **RTOS** - Real Time Operating System
- **SRAM** - Static Random Access Memory

1
Introduction

1.1 Background

Permanent Magnet Stepper Motors (PMSM) are an integral part of many industrial and commercial products where precision movement is required. One of their biggest advantages is that they can be driven to an exact position without the need for positional feedback, as long as the speed, acceleration and load are low enough [21]. Because the lack of feedback requirement the most common control method of stepper motors is open-loop control where the phase currents of the motor are driven in such a way that they produce sequential steps [2]. As there is no feedback involved in open-loop control it is assumed that the motor always moves the specified number of steps for a given input signal. In order to maintain accuracy, open-loop control relies on low speed and torque requirements [21]. If a fault occurs, such as an overshoot, the controller is not made aware of this and the motor will forever be at the wrong position. The consequences of this can range from a non-event to disastrous depending on the application.
With the rise in popularity of 3D printers the need for good stepper motor drivers has increased. In 3D printing loosing steps is generally not acceptable as this results in the printed object not meeting its specifications. Some widely used stepper motor drivers used for 3D printing, A4982, DRV8825 and TMC2130 all lack positional feedback and instead operate as open loop controllers [3][4][20]. Because 3D printing can often take multiple hours (more than ten hours is not uncommon) a lot of time can be saved if the motors could be driven faster, given that they retain their accuracy. In a paper by Jeff Kordik it was shown that large gains in performance can be achieved by using closed loop control instead of open loop when stepper motors are used in servo-like applications [15]. Similarly in a paper by Jijo Paul improvements to vibrations and torque fluctuations were observed when comparing open loop control to vector control [12].

1.2 Aim

This master's thesis involves the development of a model based Field Oriented Controller (FOC) and Velocity/Position controllers for a Permanent Magnet Stepper Motor. The FOC will be implemented using standard PI current control while the Velocity/Position controllers will implemented both using PID and LQR in order to compare the two and select the algorithm most suited for stepper motors. A Simulink model of a stepper motor will be implemented and used to create and

simulate the controllers before they are implemented and evaluated on the target hardware.

1.3 Limitations

This master's thesis will focus on the software side of the motor driver, and will therefore not be addressing hardware development.

1.4 Ethical aspects

When producing any public document it is important to reflect over the uses to which it can be put. Firstly it is important that all reported results are true and not artificially fabricated, not only because this would be unethical, but also not to give false or inaccurate information to any work that would continue to build upon this.
Secondly one must consider to what this work could be used for and if this would enable unethical uses. This seems very unlikely as this thesis aims to improve upon methods already widely in use. However one possible outcome of making automation better and more affordable is that many low-skill jobs can be replaced by automated systems, which could lead to lost jobs.

2
Theory

2.1 Permanent Magnet Stepper Motors

A Permanent Magnet Stepper Motor (PMSM) is a type of motor which is similar to a synchronous AC motor, but with an increased number of poles. The rotor of the PMSM consists of a permanent magnet with a number of rotor teeth, see Figure 2.1, surrounded by the stator coils. The rotor teeth allow the motor to move in discrete increments, or steps, hence the name stepper motor.

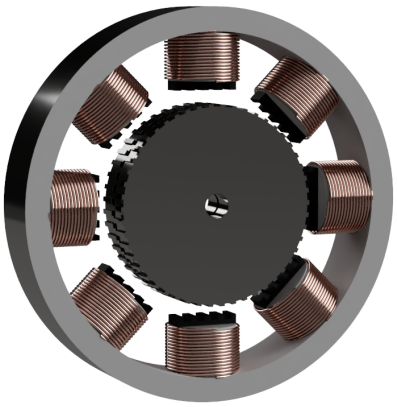


Figure 2.1: Principal model of the rotor and stator of a stepper motor

2.2 Principle of operation

The simplest way to achieve rotation of a stepper motor is to send pulses of current through the two stator phases in a specified pattern. For the bipolar stepper motor the coils are energized in alternating order, with the sign of the current changing after each step. Let the two phases of the motor be called A and B, then the phase currents should follow the pattern

$$A^+ \to B^+ \to A^- \to B^-, \tag{2.1}$$

where A^+ denotes a positive current through phase A and B^- is a negative current through phase B etc. The resulting (ideal) currents can be seen in Figure 2.2.

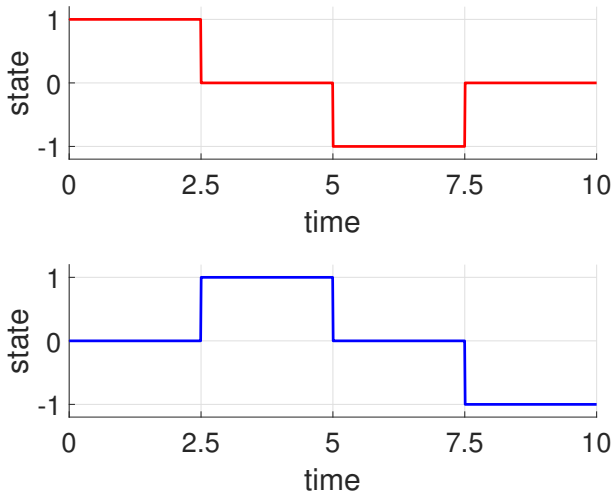


Figure 2.2: Ideal currents through the phases when driving the stepper motor using the scheme described by (2.1)

The current pattern shown in Figure 2.2 would result in the motor moving four discrete steps. The number of steps in a full rotation of the motor is determined by the number of teeth pairs (also known as N_r) and the number of phases according to

$$N_{steps} = 2N_{phases}N_{teethpairs} = 2N_{phases}N_r. \tag{2.2}$$

N_r represents the number of teeth pairs on the south and north poles of the rotor.

2.3 Modeling

The modeling of a stepper motor can be divided into two parts, mechanical and electrical. The mechanical part of the model describes the dynamics of how the rotor moves in the motor. The electrical part describes the behaviour of the voltages and currents inside the motor windings.

2.3.1 Mechanical model

The mechanical model is based in Newtonian physics for rotating bodies:

$$T = J\frac{d\omega}{dt} \iff \frac{d\omega}{dt} = \frac{T}{J}, \tag{2.3}$$

where J is the inertia of the rotor, T is the torque applied to the rotor, and ω is the mechanical rotational speed of the rotor. The torque acting on the rotor is divided into four separate terms as shown in detail in (2.5).

- Viscous friction
- Mechanical load
- Magnetic forces (two terms, as there are two phases)

The friction is modeled as proportional by some constant to the rotational speed of the rotor ($T_{friction} = B\omega$). The mechanical load (τ_L) can be selected to what ever the scenario requires and is applied as an external torque that is acting on the rotor. The magnetic forces represents the link between the mechanical and electrical dynamics. When current flows through the motor coils a magnetic field is created. This magnetic fields interacts with the permanent magnets in the rotor resulting in a force. The quadrature component (F_q) of this force produces torque on the rotor (and stator) while the direct component (F_d) does not produce any torque, as depicted in Figure 2.3. The magnitude of this force is determined by the currents and the mechanical angle of the rotor (θ), scaled by the torque constant (K_m) of the motor [17],

$$T_{quadrature} = K_m(-i_\alpha \sin(N_r\theta) + i_\beta \cos(N_r\theta)). \tag{2.4}$$

K_m can be found either directly in the data sheet of the motor, indirectly through current and torque ratings of the motor or by experimentation using a known torque load.

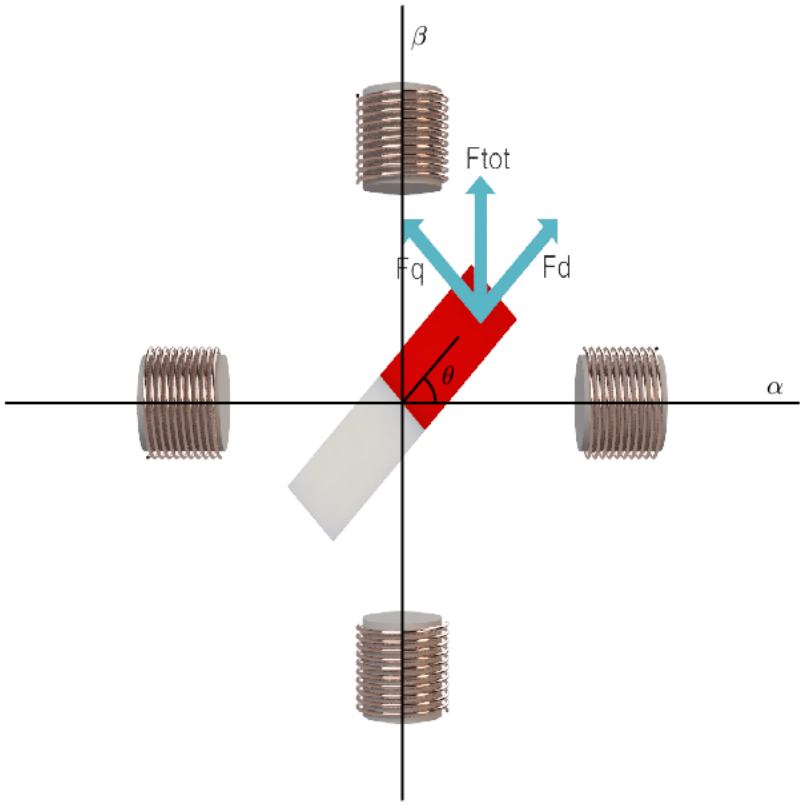


Figure 2.3: Simplified schematic of how the magnetic forces act on the rotor. In this schematic $N_{step} = 4$, as there are 2 phases, and one north-south tooth pair.

All this together gives the mechanical dynamics of the stepper motor as described in (2.5).

$$\begin{aligned}
\frac{d\omega}{dt} &= \frac{T}{J} = \frac{1}{J}(T_{quadrature} - T_{friction} - T_{load}) \rightarrow \\
\frac{d\omega}{dt} &= \frac{1}{J}(-K_m i_\alpha \sin(N_r\theta) + K_m i_\beta \cos(N_r\theta) - B\omega - \tau_L) \\
\frac{d\theta}{dt} &= \omega
\end{aligned} \tag{2.5}$$

2.3.2 Electrical model

The equivalent circuit for one phase of the stepper motor can be seen in Figure 2.4 [17]. The subscript **x** stands for either phase α or phase β in this case.

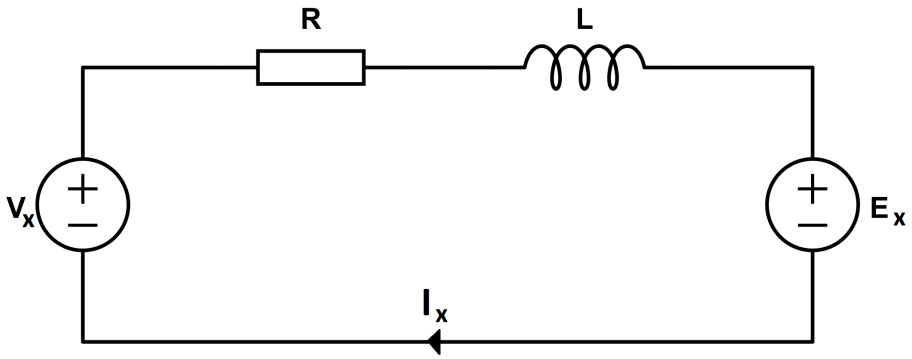


Figure 2.4: Equivalent circuit for one phase of a stepper motor. In the figure x denotes the phase and can be either α or β

By applying Kirchhoff's voltage law to Figure 2.4 the total voltage sum becomes

$$i_{s,x}R + L\frac{di_{s,x}}{dt} + E_x - V_{s,x} = 0 \tag{2.6}$$

where the voltage E_x represents the back-emf generated by the motor, which itself can be expressed as

$$E_\alpha = K_m\omega \sin(N_r\theta) \tag{2.7}$$

and

$$E_\beta = -K_m\omega \cos(N_r\theta) \tag{2.8}$$

for their respective phases. Rearranging this and changing the names to suit the phases results in the dynamics describing the current in the circuit,

$$\frac{di_\alpha}{dt} = \frac{1}{L}[v_\alpha - Ri_\alpha - K_m\omega \sin(N_r\theta)], \tag{2.9}$$

$$\frac{di_\beta}{dt} = \frac{1}{L}[v_\beta - Ri_\beta + K_m\omega \cos(N_r\theta)], \tag{2.10}$$

where v_α and v_β are the phase voltages which are the inputs to the system.

2.4 Control

2.4.1 Traditional control

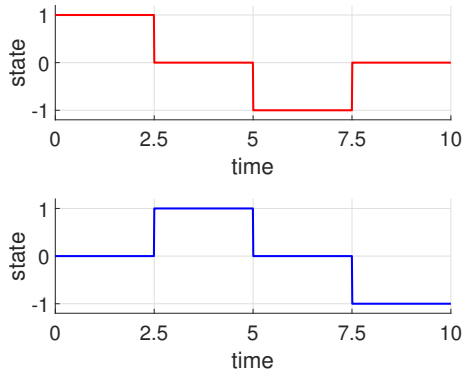
The traditional control method for stepper motors has usually been open loop, which due to its simplicity has been a contributing factor to the popularity of stepper motors [11][12]. There are several ways a stepper motor can be controlled in open loop configuration each with its own advantages.

2.4.1.1 Full stepping

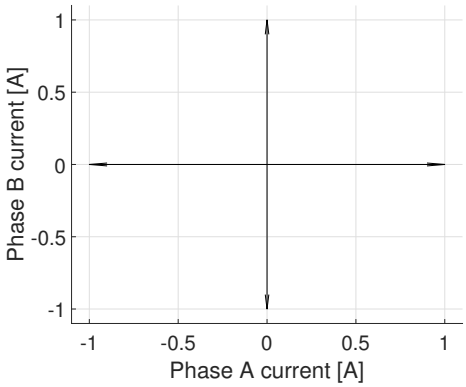
The simplest control method is full stepping. Full stepping a stepper motor involves sending square waves through each of the motor phases in the correct order to produce rotational motion. The pattern needed to rotate the motor is described by

$$A^+ \longrightarrow B^+ \longrightarrow A^- \longrightarrow B^- \tag{2.11}$$

where A denotes the phase and the sign $(+/-)$ denotes the direction of the current. Another visualization of this can be viewed in Figures 2.5a and 2.5b. The direction the motor turns can be reversed by switching the two phases. For a motor with 2 phases there are 4 full steps for each electrical revolution, each step separated by 90°. The number of motor poles determines the number of electrical revolutions for each mechanical revolution, as shown in (2.2). For the case with 2 phases (the most common) a phasor diagram of the phase currents has be constructed, showing these 4 full steps in each electrical revolution, and can be viewed in Figure 2.5b. The advantages of using full stepping is that it is very simple and requires simple hardware. There is no need for PWM as all signals are simply fully on or fully off. This simplicity results in the drawback of less resolution and jerky movement of the motor as it jumps between each step position.



(a) The phase currents in the two stepper motor phases required in order to achieve full stepping. Observe that the y axis depicts state and not actual current. This means that 1/-1 represents fully on and 0 is off.

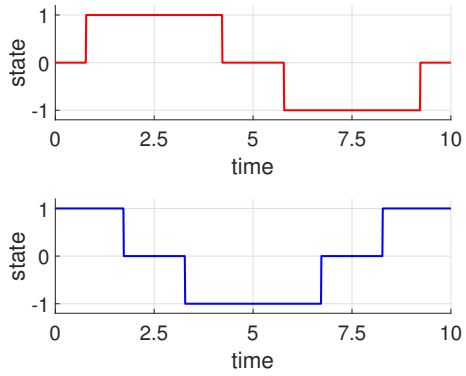


(b) Phasor diagram of the phase currents during full stepping. Each step, represented by an arrow is separated by 90 electrical degrees.

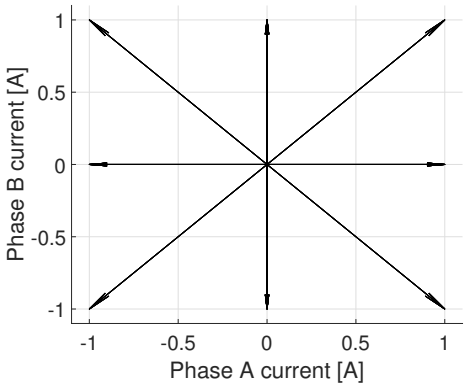
Figure 2.5: Waveform and phase for full stepping

2.4.1.2 Half stepping

The resolution of the stepper motor can be doubled by applying half stepping instead of full stepping. This method creates a new step position halfway between each full step by applying current of equal amplitude to each phase simultaneously, as can be seen in Figure 2.6a. This results in a phasor diagram with 8 distinct current vectors instead of 4, as shown in Figure 2.6b. This gives the advantage of higher resolution in the positioning and smoother operation, at the expense of more complicated control signals as well as more fluctuating torque. The torque produced at each half step position is $\sqrt{2}$ higher than the full step position, as is apparent in Figure 2.6b.



(a) Phase currents when driving a stepper motor using half steps. Note that there are overlaps where both phases are active. These overlaps corresponds to the new halfstep positions

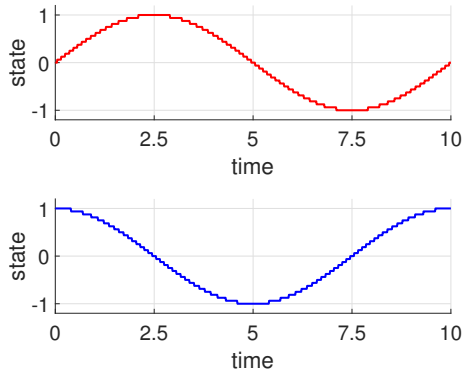


(b) Here the new half step positions are the longer vectors on the diagonals, placed halfway between the previous full steps.

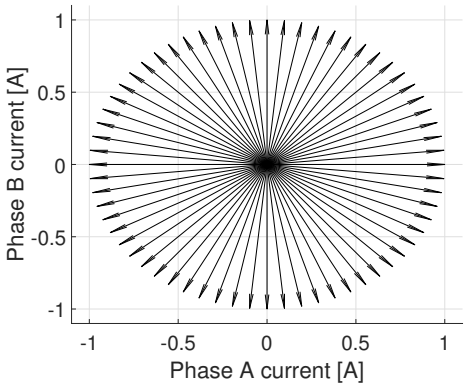
Figure 2.6: Waveform and phase for half stepping

2.4.1.3 Micro stepping / Wave stepping

In order to increase the resolution and smoothness of the motor further micro/wave stepping can be applied. This method builds on half stepping, but increases the number of step positions to an arbitrary amount, as well as keeping the torque more stable throughout the revolution of the motor. The control signals now looks more like sine waves and require more complicated hardware to create, such as an inverter to create a PWM signal. Figure 2.7a shows the sine shaped currents which corresponds to 16^{th}-stepping, a commonly used step size for cheaper drivers, such as A4982[3]. Modern drivers are now offering higher resolutions, 32-stepping for DRV8825 [4] and even 256-stepping using the TMC2130 from Trinamic [20].
The biggest drawback with micro/wave stepping, as well as all previously mentioned open loop methods, is that they are prone to losing steps. This means that the commanded position does not match the actual position. Steps will be lost when the angle between the stator field and the rotor position becomes larger than 180 electrical degrees, as this causes the quadrature force to change sign and the motor to snap back to the previous position, without it being registered by the open loop driver. If the cause of the large angle was transient the motor will continue operating normally after the snap-back event, but with an offset of one or more steps.
Situations where this occur is when the load torque applied on the motor exceeds the driving torque. This can be caused by a too heavy load, or when the motor is accelerated to quickly resulting a too large torque generated by the load inertia.



(a) Phase currents used for micro/wave stepping at the 16^{th}-step resolution. The signals looks close to a sine wave.



(b) In this figure 16 current vectors are shown in every quadrant of the electrical revolution. All of the vectors are of equal length which gives smoother operation due to the close to constant torque.

Figure 2.7: Waveform and phase for 16^{th} stepping

2.4.2 Closed loop control

Closed loop control can be achieved by adding positional feedback to any of the previous open loop control methods discussed above. One type of feedback that can be used is to add an encoder to the motor shaft. With an encoder the absolute position of the motor will always be known, and any lost steps can be compensated simply by taking additional steps until the positional error becomes small enough. However using feedback in this manner is not optimal, as some energy is wasted on creating magnetic forces that do not produce any torque. To make the control more efficient the angle of the magnetic field inside the motor needs to be controlled more precisely. With the *correct* magnetic flux angle the non torque producing component can be eliminated.

2.4.3 Field Oriented Control

Controlling the direction of the magnetic flux, or the magnetic field, is the purpose of Field Oriented Control, also known as vector control [9]. The optimal way of controlling the field can be found by studying the simplified schematic in Figure 2.3. This figure represents one electrical revolution of the motor, where the electrical revolution relates to the mechanical revolution according to

$$\theta = \frac{4\theta_e}{N_{steps}} = \frac{\theta_e}{N_r}. \tag{2.12}$$

In Figure 2.3 the upper and lower coils of the stator are energized in such a way as to produce an inward pointing magnetic north and south pole respectively. The rotor, represented by a bar magnet, is not magnetically aligned with the generated stator field. This misalignment will produce a force on the rotor, shown in the figure

as F_{tot}. This force can be divided into two components in the reference frame of the rotor. These components are called the *direct* component (F_d) and the *quadrature* component (F_q). As discussed previously in Section 2.3.1 it is only F_q that produces any torque on the rotor, while F_d produces an outward facing , radial force. The ideal situation is then apparent; F_d should be minimized and F_q should be maximized. Given F_{tot}, the ideal situation can be summarized as

$$F_q = F_{tot}, \ F_d = 0. \tag{2.13}$$

2.4.4 Controlling direct and quadrature forces

As discussed before, F_d and F_q are proportional to the magnetic field generated by the stator. This magnetic field is in turn proportional to the current flowing through the motor coils. This current can be controlled by the voltage applied to the motor phases, thus resulting in a clear path from the control signal to the variable to control.
However as F_d and F_q are located in a rotating reference frame (the dq-frame) they represent time varying currents in the reference frame of the stator (the $\alpha\beta$-frame), which also is the reference frame of the control signal. As the reference frame is rotating from the perspective of the stator the required stator currents are sinusoidal for a constant rotational speed. Instead of attempting to control the sinusoidal currents in the stator frame they can be transformed into the rotating reference frame, thus turning them into constants (for a constant speed), which are much simpler to control. This transformation effectively transforms the stepper motor into a DC motor, because there is now a current, i_q, which is directly proportional to the motor torque,

$$T_e = K_m i_q, \tag{2.14}$$

just as with a DC motor.

The field oriented control algorithm for a stepper motor can thus be summarized into the following steps:

1. **Measure phase currents**
The currents are measured in the reference frame of the stator as i_α and i_β.

2. **Transform current into the rotor coordinate frame**
Transforming the phase currents into the rotating reference frame, or *synchronous* coordinates, is called the *Park transform* and can be done through a simple matrix multiplication,

$$\begin{bmatrix} i_d \\ i_q \end{bmatrix} = \begin{bmatrix} \cos(N_r\theta) & \sin(N_r\theta) \\ -\sin(N_r\theta) & \cos(N_r\theta) \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix}, \tag{2.15}$$

where the matrix is a rotation matrix which projects the $\alpha\beta$-currents onto the dq-axes [17]. The relation between the two coordinate frames can be seen in Figure 2.8.

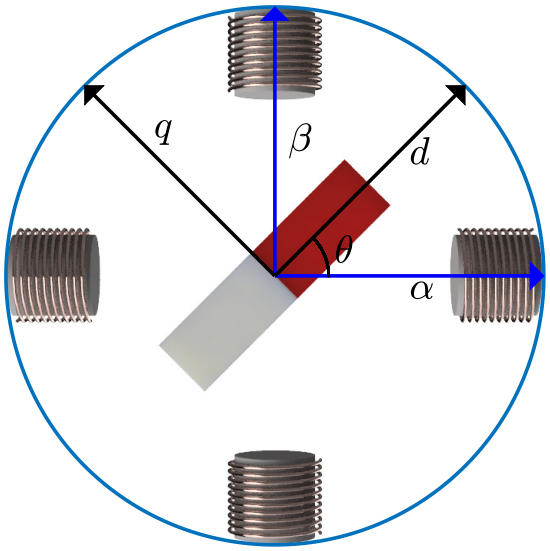


Figure 2.8: The two coordinate systems superimposed on the stepper model, showing the relation between the two reference frames. The black lines (dq) rotates together with the rotor, while the blue lines ($\alpha\beta$) are stationary and attached to the stator.

3. **Apply current control algorithm**
Because the currents needs to be controlled to constant levels, $i_d = 0$ and $i_q = \frac{T_e}{K_m}$, a PI-controller can be used for this task.

4. **Transform the controller output into the stator coordinate frame**
The PI-controller outputs two voltages, v_d and v_q, in synchronous coordinates that needs to be applied to reach the current set points. As the control signal can only be applied in stator coordinates the inverse to the transformation in (2.15) needs to be performed,

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \begin{bmatrix} \cos(N_r\theta) & -\sin(N_r\theta) \\ \sin(N_r\theta) & \cos(N_r\theta) \end{bmatrix} \begin{bmatrix} v_d \\ v_q \end{bmatrix}, \tag{2.16}$$

the *inverse* park transform. The resulting voltages, v_α and v_β, can be applied to the motor phases as the control signal.

The four steps described above will control the currents in the synchronous coordinate to meet a given current target. This target is calculated according to (2.14) from a desired torque. This algorithm will thus indirectly control the torque produced by the motor.

2.4.5 PI - Current control

The design of the PI current control is based on the dynamics of the current in the motor as described by (2.9) and (2.10). The current dynamics in the stator frame can then be seen as

$$L\frac{di}{dt} = V - Ri - E \tag{2.17}$$

which becomes

$$L\frac{d}{dt}\begin{bmatrix} i_d \\ i_q \end{bmatrix} = \begin{bmatrix} v_d \\ v_q \end{bmatrix} - R\begin{bmatrix} i_d \\ i_q \end{bmatrix} + L\begin{bmatrix} \omega i_q \\ -\omega i_d \end{bmatrix} - \begin{bmatrix} E_d \\ E_q \end{bmatrix} \tag{2.18}$$

when transformed into synchronous coordinates. The second to last term represents the cross coupling between the two systems. However, by selecting the input as

$$\mathbf{v} = \begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} v'_d \\ v'_q \end{bmatrix} + \omega L\begin{bmatrix} i_q \\ i_d \end{bmatrix} \tag{2.19}$$

the cross coupling can be cancelled [9].

Using the Laplace transformation the transfer function of (2.18), from the input $\mathbf{v}$ to the output $\mathbf{i}$ becomes

$$G(s) = \frac{1}{sL + R} \tag{2.20}$$

where the back-emf (E) has been modeled as a subtractive load disturbance on the input signal, as depicted in Figure 2.9.

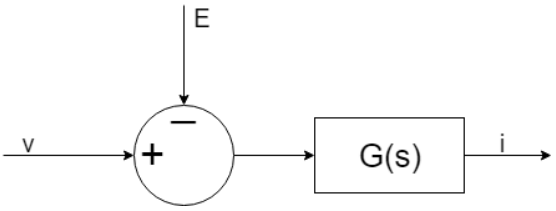


Figure 2.9: The back-emf E is subtracted from the input signal v before entering the transfer function $G(s)$ and becoming the output current i.

Because the system in (2.20) is of order one a PI controller,

$$F(s) = k_p + \frac{k_i}{s}, \tag{2.21}$$

is sufficient [9]. The closed loop transfer function then becomes

$$G_c(s) = \frac{F(s)G(s)}{1 + F(s)G(s)}. \tag{2.22}$$

The preferred closed loop transfer function is chosen as a first order system, and should therefore be on the form

$$G_c(s) = \frac{\alpha}{s + \alpha} = \frac{1}{sT_e + 1},\ \alpha = \frac{1}{T_e}. \tag{2.23}$$

The constant T_e in (2.23) is the closed loop electrical time constant, which is defined in terms of the rise time,

$$t_{re} = T_e \ln(9). \tag{2.24}$$

By combining (2.22) and (2.23) the closed loop transfer function becomes

$$F(s)G(s) = \frac{\alpha}{s}, \tag{2.25}$$

which in turn leads to the controller being formed by

$$F(s) = \frac{\alpha}{s}G(s)^{-1} = \frac{\alpha}{s}(sL + R) = \alpha L + \alpha\frac{R}{s} \tag{2.26}$$

In (2.26) k_p and k_i are identified as αL and αR respectively.

2.4.6 Model based motor control

Using the current control discussed in the previous section the motor input has be changed from being the phase voltage to being the torque, under the assumption that the current dynamics are fast enough. Because of this the electrical dynamics of the motor can be ignored, as they are taken care of by the current controllers. The motor can instead be considered from a purely mechanical standpoint,

$$\dot{\omega} = \frac{T - B\omega}{J} \tag{2.27}$$

In order to use a linear quadratic regulator (LQR) these dynamics must be converted into a state space representation.
By picking the state as

$$x = \begin{bmatrix} \theta \\ \omega \end{bmatrix}, \; u = T \tag{2.28}$$

the system can be represented as

$$\dot{x} = \begin{bmatrix} \omega \\ \frac{T - B\omega}{J} \end{bmatrix} \tag{2.29}$$

which in turn can be rewritten to matrix form

$$\dot{x} = Ax + Bu \tag{2.30}$$
$$y = Cx + Du, \tag{2.31}$$

using the matrices A,B,C and D

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -B/J \end{bmatrix}, \; B = \begin{bmatrix} 0 \\ \frac{1}{J} \end{bmatrix}, \; C = I_2, \; D = 0. \tag{2.32}$$

Given these matrices the LQR state feedback K is obtained by solving the algebraic Riccati equation

$$PA + A^T P + Q = PBR^{-1}B^T P, \tag{2.33}$$

for P, preferably using MATLAB. The state feedback coefficient K which minimizes the quadratic cost function

$$J = \int_0^\infty (x^T Q x + u^T R u) dt \tag{2.34}$$

is then given by

$$K = R^{-1}(B^T P). \tag{2.35}$$

The cost function is defined by the matrices R and Q which define the cost of the input and state respectively. The LQR controller is shown in Figure 2.10 where x is the current state, ref is the reference signal and u is the output from the controller.

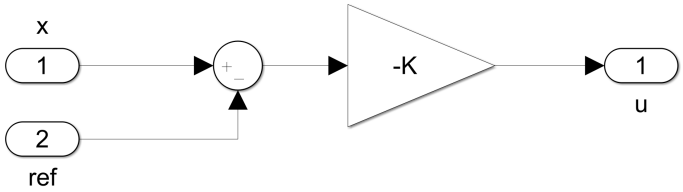


Figure 2.10: The LQR state feedback controller

LQR can be augmented with an integrating state for reference following. This is done by adding one or more states that integrates the error signals,

$$\begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} Ax + Bu \\ Cx - r \end{bmatrix}, \tag{2.36}$$

where r is the reference signal for the observed state (Cx) and z is the new integrating state. The integrating portion is the second row of the matrix equation ($Cx - r$). Here C is the same matrix as before (I_2) which transforms x to y according to (2.31) with $D = 0$. This means that the error being integrated is the observed error (Cx), which in this case happens to coincide with the state error (due to $C = I_2$).

3
Implementation

3.1 Model

The model of the stepper motor was implemented in MATLAB Simulink. The model equations used were those presented in (2.5). This model was deemed sufficient based on other projects using the same model, such as Kim [13] and Rusu [18]. In Figure 3.1 an overview of the model is displayed. The model was divided into electrical and mechanical dynamics as can be seen in the figure.

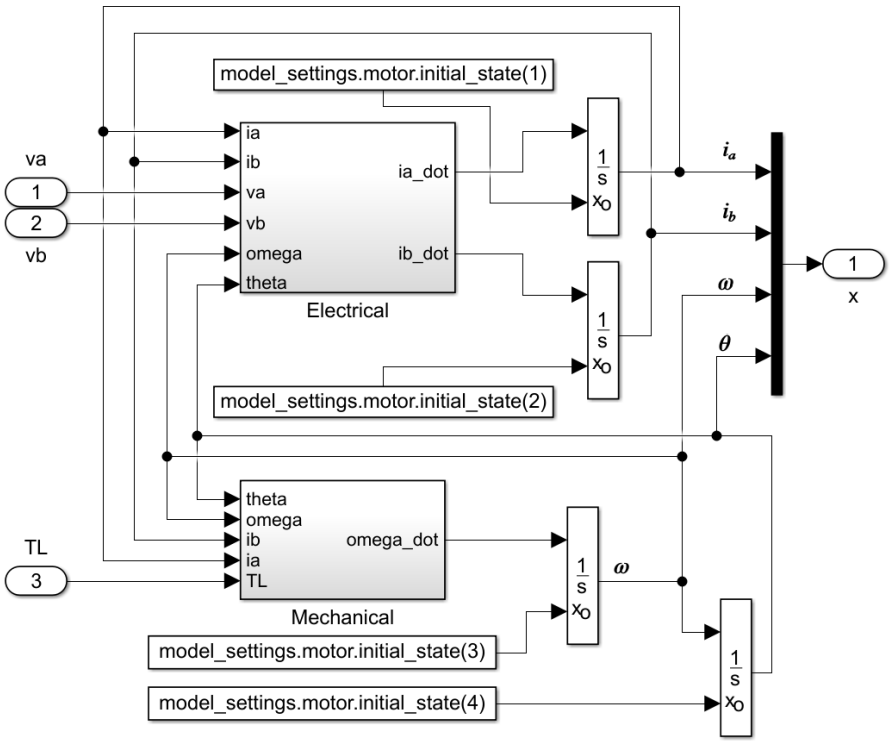


Figure 3.1: Overview of the stepper model in Simulink. The implementation is divided into the electrical and mechanical dynamics

The equations for the dynamics were implemented with fully customizable parameters as in all parameters were initialized outside of the model definition, as displayed in Figure 3.2. This gave the advantage of being able to change the motor specifications on the fly, without ever having to modify the model implementation. Because of this the performance of the controller could later be easily and rapidly evaluated for motors with different specifications.

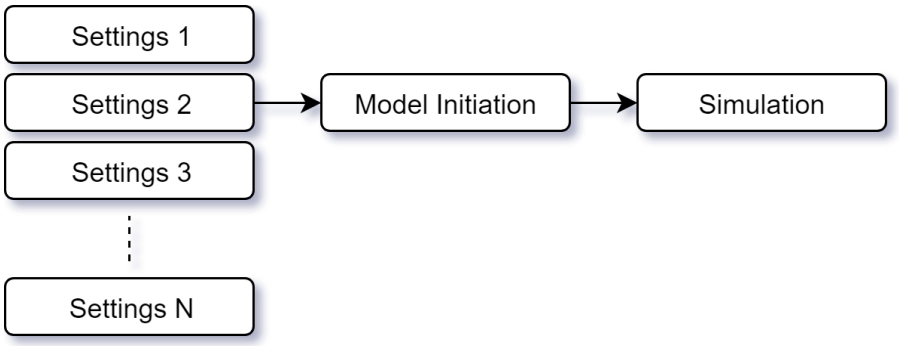


Figure 3.2: Streamlinging of the simulation process allowed for easy changes to model and simulation settings, as well as having multiple copies of the settings with slight variations

3.2 Field Oriented Control

The field oriented control implementation in simulink can be viewed in Figure 3.3. The input to the controller is the desired torque $T_{desired}$, which is transformed into a desired current according to (2.14), as well as the measurements of the currents (i_α, i_β) and rotor angle (θ). The PI current controller (labeled *Current Controller* in Figure 3.3) can be viewed in Figure 3.4 and as can be seen there are no decoupling terms included. These were not implemented because they had a negligible effect at the speeds used for the stepper motor (recall that the coupling in (2.18) contains ω) and allowed the code on the hardware to be simplified.

Using (2.26) and a rise time of 10 ms the current control gains were calculated to

$$k_p = 0.7251, \; k_i = 468.0. \tag{3.1}$$

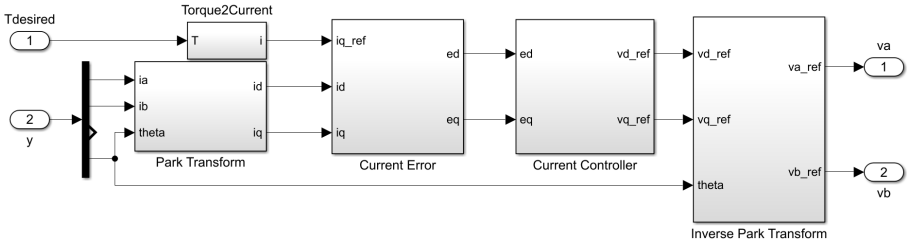


Figure 3.3: Field oriented control implementation in simulink.

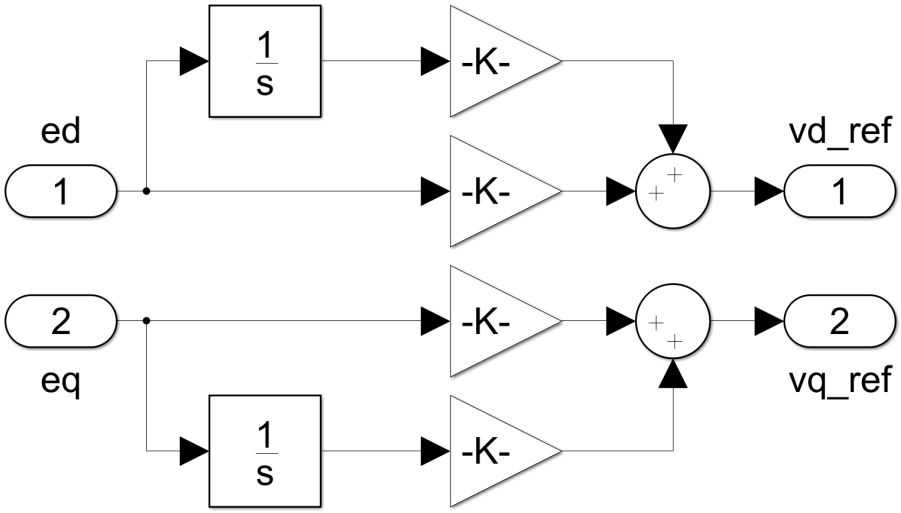


Figure 3.4: Implementation of PI-current controller in simulink.

3.3 Velocity and position control

Two separate control algorithms were implemented, PID and LQR. This allows for comparison between these two common control laws as well as finding a control law well suited for the task.

3.3.1 PID

Two different versions of PID were implemented. The first version controlled only the speed of the motor, while completely disregarding the position. This PID controller is presented in Figure 3.5, as the subsystem labeled "velocity controller". The velocity controller takes the current speed and the target speed as inputs and outputs the torque required to reach the given target. The inner workings of the velocity controller can be seen in Figure 3.6.
The second controller was a position-velocity controller. This controller was a cascaded controller, making use of the previously discussed velocity controller as well as a new position controller. The controller, in its full, can be seen in Figure 3.5. This controller takes a target position and the current position as input and outputs the velocity required to reach the target. This velocity is then fed to the velocity controller which outputs the required torque target.

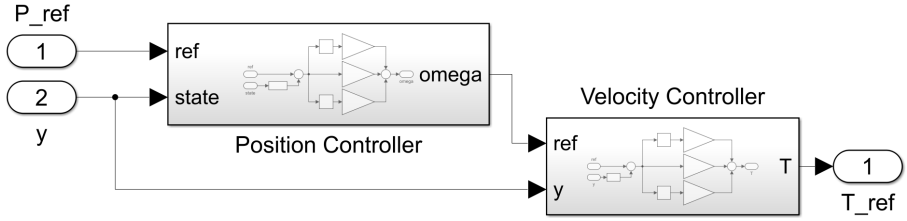


Figure 3.5: Cascading of PID controllers in order to control both speed and position

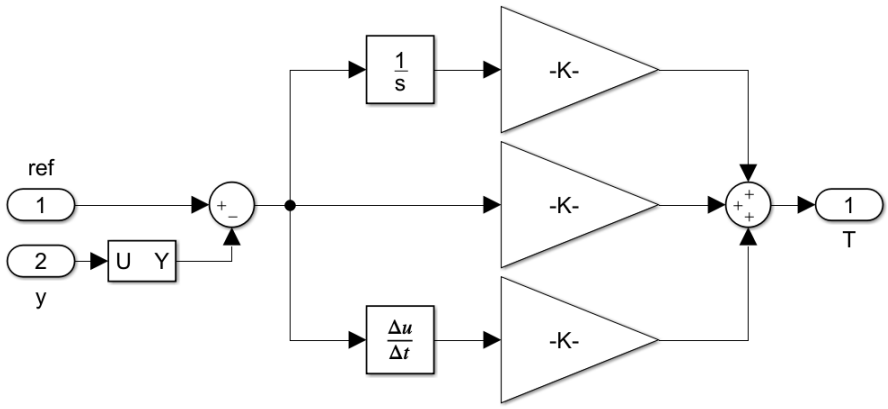


Figure 3.6: The PID controller used to control the motor speed. The input to the PID is the reference labeled *ref* and the measured state y with the controller outputting a desired torque T

Both controllers were tuned using a manual tuning technique. The tuning technique consisted of the following steps, as presented by [10]:

1. Set K_p, K_i and K_d to 0
2. Increase K_p until step response oscillates
3. Increase K_d until oscillations stops
4. Repeat 2-3 until oscillations cannot be stopped by K_d
5. Set K_p and K_d to the last stable values
6. Increase K_i until the steady state error disappears fast enough while avoiding oscillation

The resulting gains from the above mentioned method were

$$k_p = 0.01,\ k_i = 0.05,\ k_d = 0.0001 \tag{3.2}$$

for the speed controller and

$$k_p = 20.0,\ k_i = 0.0,\ k_d = 0.0 \tag{3.3}$$

for the position controller.

3.3.2 LQR

Similarly to the PIDs, two LQR controllers were also implemented, one for speed and one for position together with speed. The LQR, being a state feedback control law is very simple to implement in simulink, as evident by Figure 2.10 in the previous section.
The tuning of the LQR was done by adjusting the cost matrices, R and Q from (2.34). R was chosen as

$$R_{speed} = 500 \tag{3.4}$$

$$R_{position} = 1000, \tag{3.5}$$

as this resulted in the demanded control signals for the voltage staying within the operating range. Q was selected using the mindset that when both position and speed were regulated position was most important and therefore had a higher weight. However it turned out that penalizing speed was more beneficial when combined with actively selecting a speed reference, which is reflected by the higher weight of the speed state.
When only speed was regulated it was selected as high as possible while ensuring that the desired input voltages stayed below the maximum value of 24 V. The final values of the Q matrices were chosen as

$$Q_{speed} = \begin{bmatrix} 0.1 \end{bmatrix} \tag{3.6}$$

$$Q_{position} = \begin{bmatrix} 1.0 & 0 \\ 0 & 0.1 \end{bmatrix}. \tag{3.7}$$

When using an integrating state the Q-matrix was augmented with the costs of the integrated states as well,

$$Q_{speed,i} = \begin{bmatrix} 0.05 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.8}$$

$$Q_{position,i} = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 0.00001 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.001 \end{bmatrix}. \tag{3.9}$$

The gains for the integrating states had to be selected as very small numbers in comparison to the regular states in order for the system to remain controllable.

Table 3.1: Motor parameters used for the LQR

B	0.0008
J	4.5×10^{-5}

These choices of Q and R where used in combination with the matrices presented in (2.32) with the parameter values shown in Table 3.1 to solve the Riccati equation (2.33) in MATLAB using the *lqr* command and resulted in state feedbacks of

$$K_{speed} = 0.0316 \tag{3.10}$$

$$K_{position} = \begin{bmatrix} 0.0992 & 0.0316 \end{bmatrix} \tag{3.11}$$

for the normal LQR and

$$K_{speed,i} = [0.0226 \ \ -0.1] \tag{3.12}$$

$$K_{position,i} = \begin{bmatrix} 0.0439 & 0.0181 & 2.2 \times 10^{-11} & -0.0014 \end{bmatrix} \tag{3.13}$$

for the LQR with integrating states.

3.4 Hardware

The hardware used for the project was a custom driver board called *pmstep*, developed by Plejd AB. The final circuit board for pmstep can be seen in Figure 3.7. The board is based around the micro controller ESP32 [6] as the main processor with an angle sensor (TLE5012B [19]) and an H-bridge (DRV8876 [5]). The motor current is measured using a shunt resistor, an amplifier (LMP860 [16]) and the ADC of the ESP32. The stepper motor used was a NEMA17 format motor with the motor parameters shown in Table 3.2

Table 3.2: Motor parameters of the motor used

Phase resistance (R)	2.13 Ω
Phase inductance (L)	3.3 mH
Viscous friction (B)	0.0008 Nm/rad
Rotor Inertia (J)	4.5×10^{-5} kgm^2
Torque Constant (K_m)	0.23 Nm/A
Rotor teeth pairs (N_r)	50
Motor Phases (N_{phases})	2

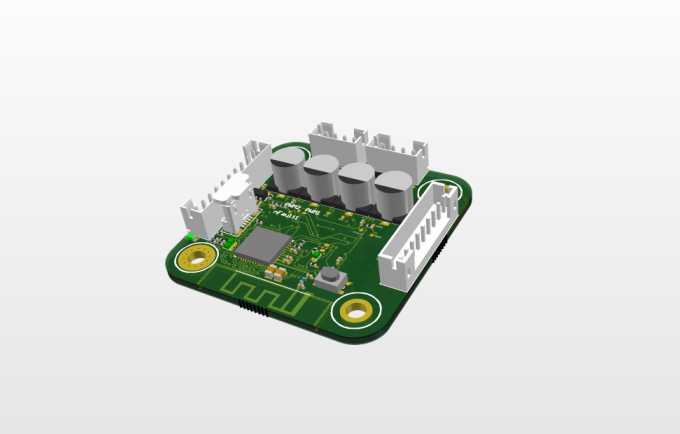


Figure 3.7: The final revision of the circuit board as of the completion of this project.

3.4.1 ESP32

The ESP32 is a fairly new micro controller produced by Espressif Systems and first released in 2016. The micro controller runs the real time operating system *FreeRTOS* [1] using a development framework called esp-idf [8] which in turn uses the CMake [14] build structure. Some of the hardware features used in this project are shown in Table 3.3. The CPU clock speed of 160 MHz is fairly high compared to other cheap micro controllers, which combined with a 32-bit address space allows the use of 32-bit number directly each clock cycle wastly improving speed compared to 8-bit systems. The ADC is a 12-bit ADC of the SAR type. 12-bit resolution is will allow

for accurate measurements of input voltage and motor currents. Finally the PWM module is very flexible, allowing up to 40 MHz frequency, and more importantly 16-bit resolution at 20 kHz. 20 kHz is a desireable frequency to run the PWM at as this moves the acoustic frequencies outside of human hearing range, while still retaining high resolution. For the full specification of the ESP32, see [7].

Table 3.3: Some relevant specifications of the ESP32 taken from the Technical Reference Manual for the ESP32 [7]

CPU clock	up to 160 MHz
Address space	32-bit
ADC	12-bit SAR
PWM	40 MHz at 1 bit to 20 kHz at 16 bits (scaling)

3.4.2 Angle sensor

The angle sensor is a GMR- (Giant Magneto Resistance) based sensor. It measures the rotation of an external magnetic field with an angular resolution of 0.01°. The sensor requires a magnet to be attached to the motor axle in order to generate a magnetic field which rotates in accordance with the rotor. While the sensor had a high resolution it was not completely linear over the revolution but had a varying offset of up to around one degree. However, as this offset was constant for a given orientation it could be compensated for by a lookup table. The lookup table was generated by positioning the motor using open loop control and thereafter reading the angle given by the sensor. This allowed an error term to be calculated, the error being how far the measured angle deviated from the expected angle,

$$lookup[\theta] = \theta_e - \theta. \tag{3.14}$$

This error was then saved in a lookup table which was stored in the flash memory of the micro controller. When the sensor is used after the calibration the correction term is added to the reading according to

$$\begin{aligned} \theta_c &= \theta + lookup[\theta] \rightarrow \\ \theta_c &= \theta + \theta_e - \theta \rightarrow \\ \theta_c &= \theta_e, \end{aligned} \tag{3.15}$$

where θ is the measured mechanical angle, θ_e is the expected angle and θ_c is the calibrated angle. As the calibration is saved in the flash memory there is only need to do the calibration once per driver/motor pair, though if either the motor or driver is switched or modified the calibration has to be redone.

3.4.3 H-bridge

The H-bridge is able to supply ± 3.5 A of drive current at a maximum voltage of 37 V. These maximum ratings are suspect to the thermal limits of the driver, which enters overheat protection mode at around 150° C, potentially before the other limits

are reached if the cooling of the chip is insufficient. Two of these chips are used, one for each motor phase. Additional specifications can be viewed in Table 3.4. The H-bridges were driven using PWM from the ESP32.

Table 3.4: Some relevant specifications of the H-bridge DRV7786 taken from the data sheet [5]

Supply Voltage	4.5-37 V
Output current	3.5 A peak
$R_{DS(on)}$	700 $m\Omega$
$T_{j(max)}$	150° C

Because the MOSFETs in the H-bridge are not ideal and many stepper motors have a low resistance the MOSFET drain-source resistance R_{DS} had to be taken into account. Figure 3.8 show a schematic of the H-bridge together with the motor (R_m) and the shunt resistor (R_s).

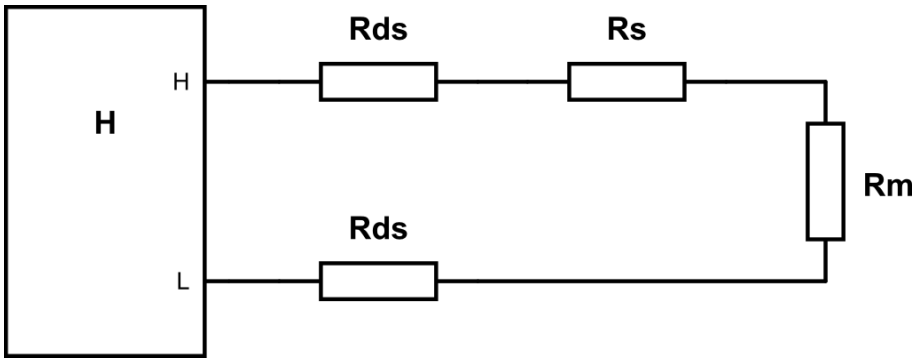


Figure 3.8: Resistances faced by the H-bridge. R_{DS} is the drain-source resistance of the MOSFETs, R_S is the shunt resistance for current sensing and R_m is the motor resistance.

When a voltage V is commanded this voltage is created before all the resistances in Figure 3.8, between the H and the L. The actual voltage over the motor is lower because of the voltage drops over the other resistances. By using KCL the voltage required between H and L in order to achieve the commanded voltage can be calculated as

$$V_{HL} = V\frac{1+\alpha}{\alpha} \tag{3.16}$$

where

$$\alpha = \frac{R_S + R_m}{2R_{DS}} \tag{3.17}$$

This correction to the H-bridge voltage is applied immediately before the H-bridge itself. V in (3.16) corresponds to the output from the current controllers after they have been transformed into stator coordinates, while V_{HL} is the voltage value sent to the PWM driver which drives the H-bridge.

It is important to note that these equations are based on the assumption that the current is stationary and that there is no back-emf. This was however deemed a justified simplification due to the low inductance and low speed of the motor. These two in combination makes this effect negligible.

3.5 C implementation

The controller was implemented on the target hardware using the C programming language. The project made use of the CMake build tool, as this is the same build environment used by esp-idf, the framework used for ESP32. With CMake code components can be defined, similar to classes in other high level programming languages. Each physical component was implemented as a component, such as the ADC driver and the H-bridge driver. Additionally, more abstract concepts were implemented as components as well such as the FOC controller and motor controller. The use of CMake components allowed multiple different code projects to be created, all referencing the components instead of having individual copies. This made sure that every test project had the newest version of the components at all times.

3.5.1 Motor control

The motor control was implemented as two parallel tasks running at different frequencies. For the field oriented control the task was invoked by an interrupt at 5 kHz. This task can be viewed as the right portion of Figure 3.9. The FOC task is responsible for ensuring that the motor produces the correct amount of torque as efficiently as possible. This is achieved through the FOC algorithm, described in detail in Section 2.4.5, which involves applying PI-control to the rotor synchronized stator currents. One exception was that compensation for neither coupling nor back-emf was implemented. The reason for this was that at the speeds the motor could be operated neither of these had any observable effect on the performance. Omission of this allowed the FOC loop to be shorter and more efficient. However it is still desirable to implement this in the future, which is why this is left for future work.
The other task, seen to the left in Figure 3.9 controls the speed and position of the motor and is executed as a FreeRTOS task running at 1 kHz. The interactions of these tasks are depicted by the large arrows in the same figure. The motor control task retrieves the angle and speed estimations from the FOC task while the FOC task is given its torque target from the motor control task. Both of these interactions are performed by atomic operations in order to prevent data corruption,as these tasks are running in parallel.

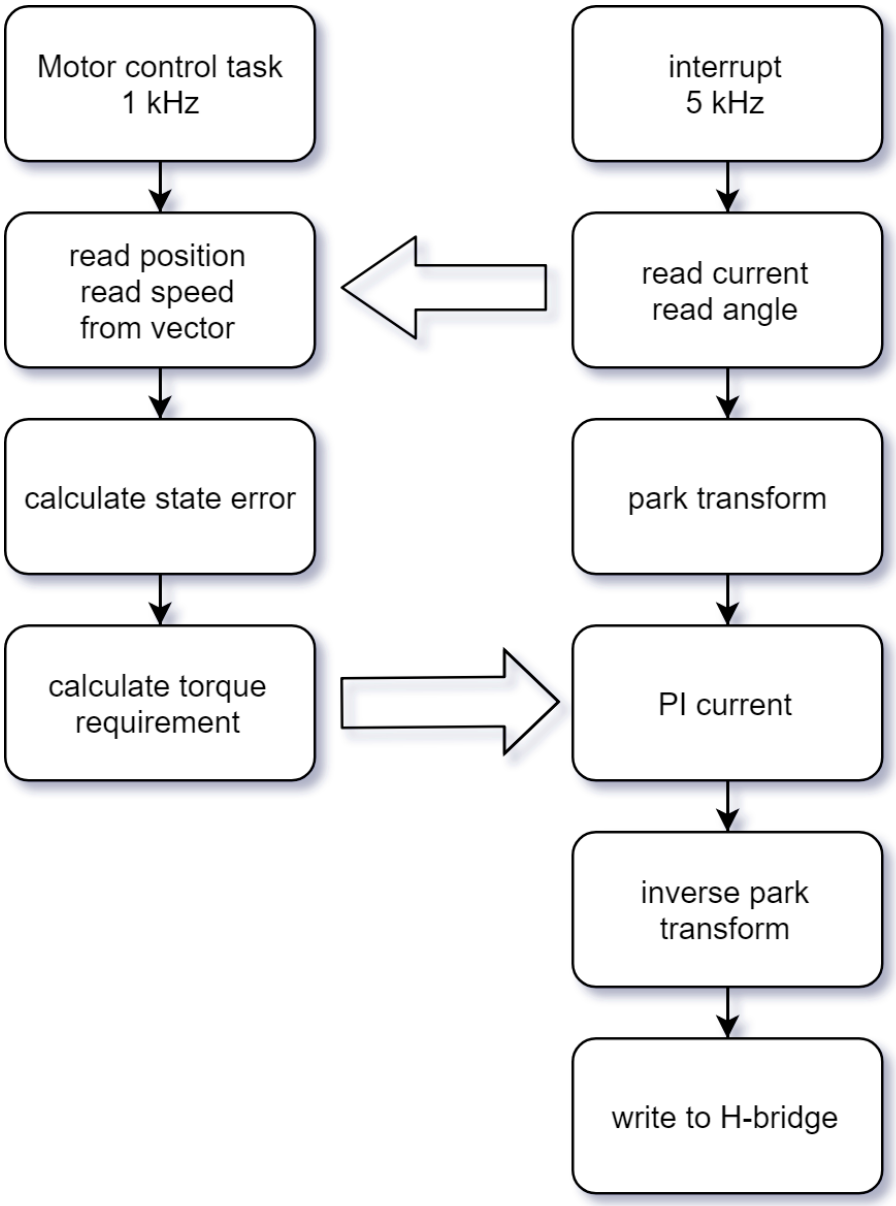


Figure 3.9: This figure shows the two parallel controller for the motor. To the right is the FOC controller which runs as 5 kHz and to the left is the motor controller running at 1 kHz. The large arrows represent data exchange. The top most is the exchange of angular information while the lower is the setting of the torque target

3.5.2 The Floating Point Unit

The code performing all calculations for the motor control made heavy use of floating point numbers. These were chosen over fixed point because the ESP32 contains a Floating Point Unit (FPU) in its hardware. Fixed Point Arithmetics is normally much faster than floating point, as these calculations can be done directly in hardware, while floating point is often computed in software. However, as the ESP32 has access to a hardware FPU the floating point arithmetics are speed up significantly. Due to the ESP32 being relatively new there are still some issues that has not been

corrected when using the standard framework. One of these issues lies with the FPU. Using the FPU in an interrupt context causes any calculations that were in progress to be corrupted. This is caused by the FPU registers not being saved by the interrupt handler which means that any calculations that are in progress while an interrupt (which also has to use the FPU) fires will be corrupted and cause a crash. There are three main workarounds to this issue:

1. Avoid using floating point in interrupts
2. Manually save the FPU registers in each interrupt
3. Move the code execution away from the interrupt context

Of these three alternatives option one was ruled out because this would require large changes to already written code. Alternative two was briefly considered but ultimately discarded as this resulted in too much overhead. It is likely that faster operation of option two could be achieved by optimization, however option three proved to be both fast and relatively simple to implement using FreeRTOS features. A flowchart of how the interrupts were handled is show in Figure 3.10. The hardware interrupt uses the *TaskNotify* feature of FreeRTOS. This feature sends a notification to the selected task, potentially unblocking it immediately, or in this case, immediately after the ISR has yielded. The task which is being sent the notification has a very high priority which causes the RTOS to initiate a context switch as soon as this task is unblocked. As the task is waiting for the notification it will be in the blocked state until the ISR sends the notification. This solution also had the advantage of making the code virtually platform independent, with only the interrupt being platform specific.

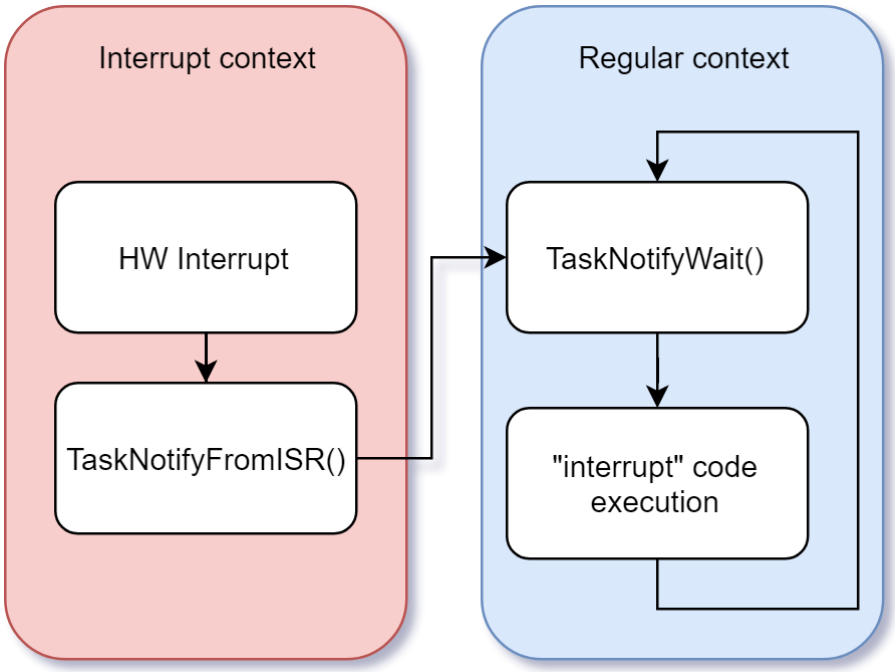


Figure 3.10: Interrupt handling using *TaskNotify* in FreeRTOS. The hardware interrupt handler sends a notification to the task containing the code that should be executed. This allows for a context switch directly from the ISR, and thus prevents the corruption of the FPU registers.

4
Results & Discussion

4.1 Simulation results

4.1.1 Current control

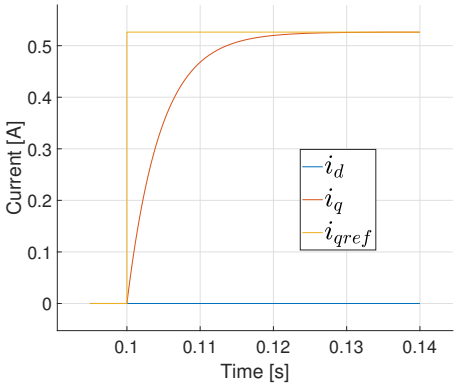
The current controllers were chosen as PI-controllers with gains according to (2.26), with a desired rise time of 10 ms. When simulating the controllers three different cases were simulated, fixed, loaded and free rotor. The fixed case shows the performance of the controller when the dynamics match reality, as the speed is 0 and therefore the coupling and back-emf are 0, as assumed. The loaded case shows how the controller would perform in a more realistic setting when the speed is not 0 and therefore the assumption of no coupling and back-emf is not true. Finally the free rotor test shows an extreme case when the speed is allowed to increase dramatically and therefore also the coupling and back-emf. In Figure 4.1a the simulation with the fixed rotor can be seen, Figure 4.2 depicts the load test and Figure 4.1b shows the free case.

In both the fixed case and the case with the load the controllers are able to keep both i_q and i_d at their setpoints, with a rise time of i_q matching the desired 10 ms. However when the motor is allowed to move freely this is not the case. The i_q controller initially follows the same trajectory as the fixed case, but starts to deviate after only a few milliseconds and subsequently fails to reach the setpoint. This is caused by the fact that the simulation takes into account that there isn't infinite source voltage. The point where the current starts to deviate coincides with the time at which the commanded voltage amplitude exceeds the source voltage (in this case 24 V).

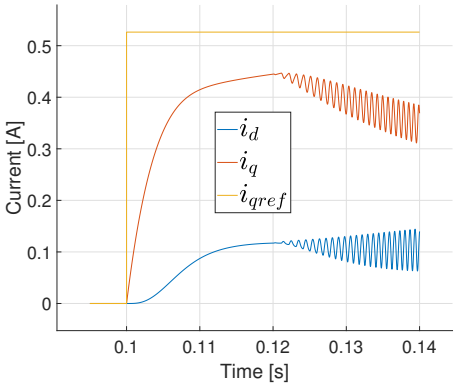
Figure 4.3 displays what happens if there is infinite source voltage, the controller eventually reaches the setpoint, but it does not do so in the specified rise time and requires a voltage amplitude of almost 90 V. The increased time required to settle is due to the fact that the calculation of the gains (2.26) does not include the back-emf. Because of this the current dynamics does not match the actual motor dynamics when the motor is moving. However as the normal use cases of stepper motors involve relatively low speeds and even holding the rotor fixed this deviation was not seen as a problem.

Finally in Figures 4.1b and 4.3 it can be seen that the direct component of the current leaves its setpoint at 0. This is because the decoupling of the controllers

were not implemented, however when the speed of the motor is low this coupling is negligible, as can be seen in Figure 4.2a.

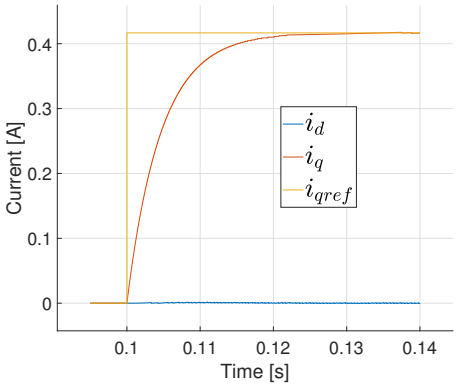


(a) Step response for the PI controller in the FOC. In this case the motor is simulated as stalled, meaning the rotor is fixed. The gains, K_p and K_i were chosen according to (2.26)

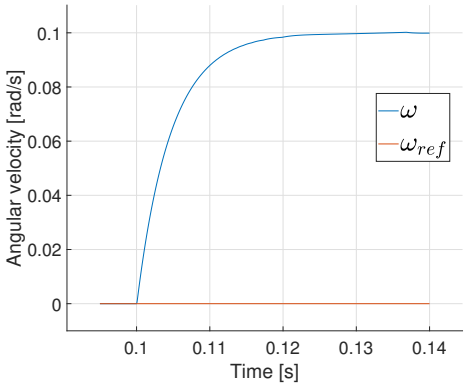


(b) The same step response as in 4.1a, except without a fixed rotor. The high frequency oscillations are due to the clipping of the control signal which was limited to ±24 V.

Figure 4.1



(a) Park currents



(b) Angular velocity ω

Figure 4.2: Step response for the PI controller in the FOC. In this case the motor is attached to a load which is proportional to the velocity ($T_L = B\omega$). The gains, K_p and K_i were chosen according to (2.26)

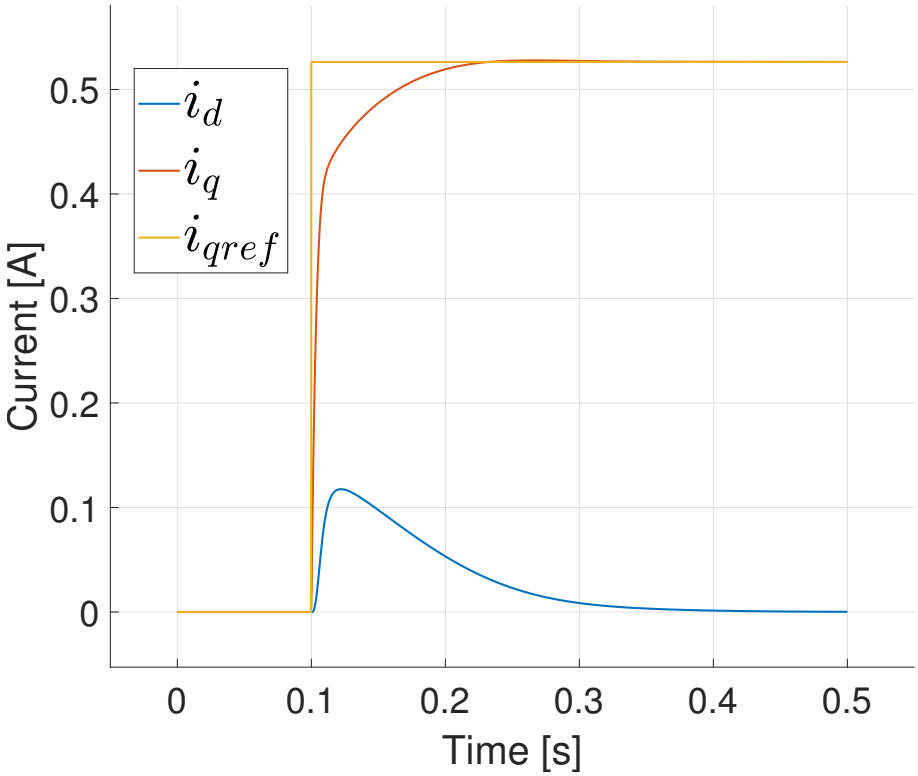
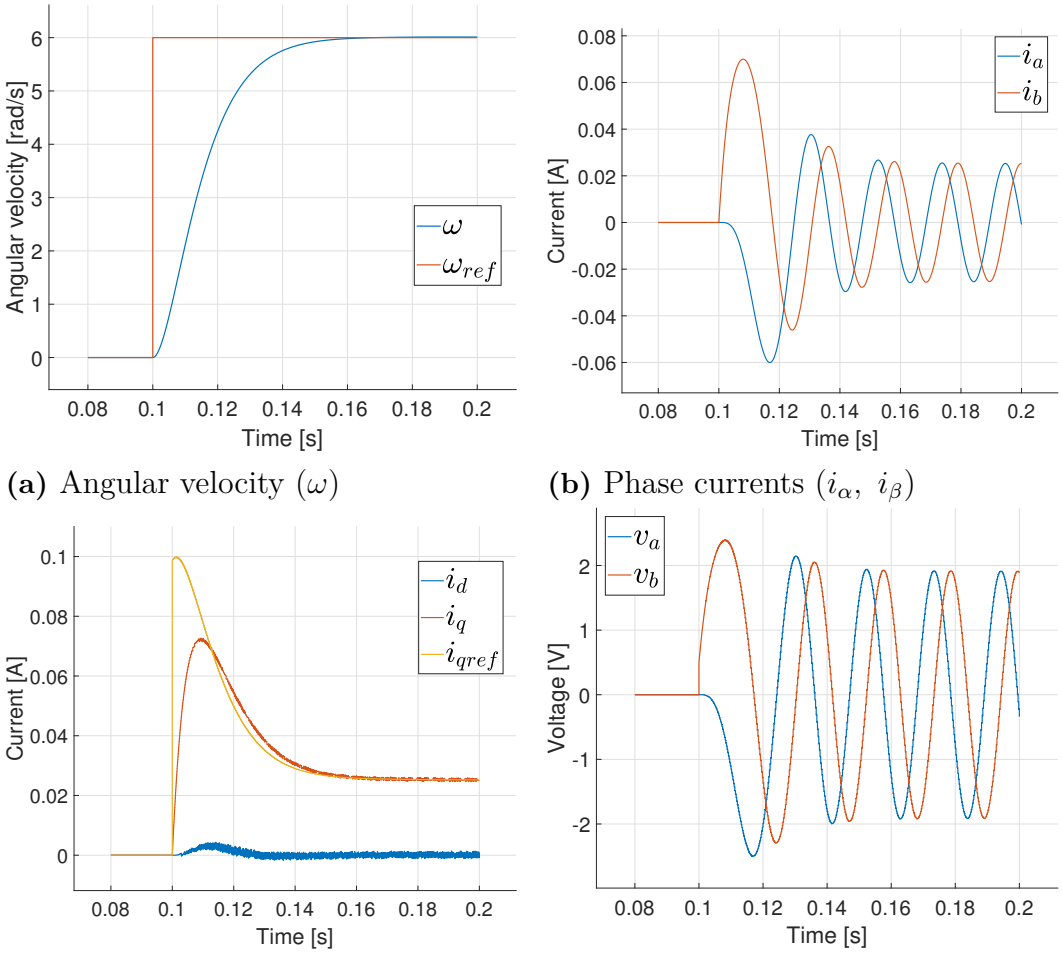


Figure 4.3: The same step response as in 4.1a, except without a fixed rotor and with ideal input (i.e. infinite source voltage amplitude). Here the effect of the coupling can bee seen as the i_d current does not stay at 0, however in this simulation the motor quickly reaches a speed outside its specification.

4.1.2 PID velocity control

The tuning of the PID controllers was done using the method described in Section 3.3.1 which resulted in the gains presented in (3.2). In Figure 4.4a the result of a speed step response can be seen. In this test and subsequent ones the motor was allowed to move freely. Figure 4.4a shows that the speed controller reaches the speed setpoint within 0.06 s and that the required voltages and currents are well within limits. In Figure 4.4c there is an initial spike in quadrature current as the motor starts rotation. The current then settles at a steady state value representative of the losses in the motor. This figure also exemplifies the advantage of the park transform, the quadrature current is controlled as a DC value for a constant torque while the actual control signals and stator currents (Figures 4.4d and 4.4b) are sinusoidal.



(a) Angular velocity (ω)

(b) Phase currents ($i_\alpha,\ i_\beta$)

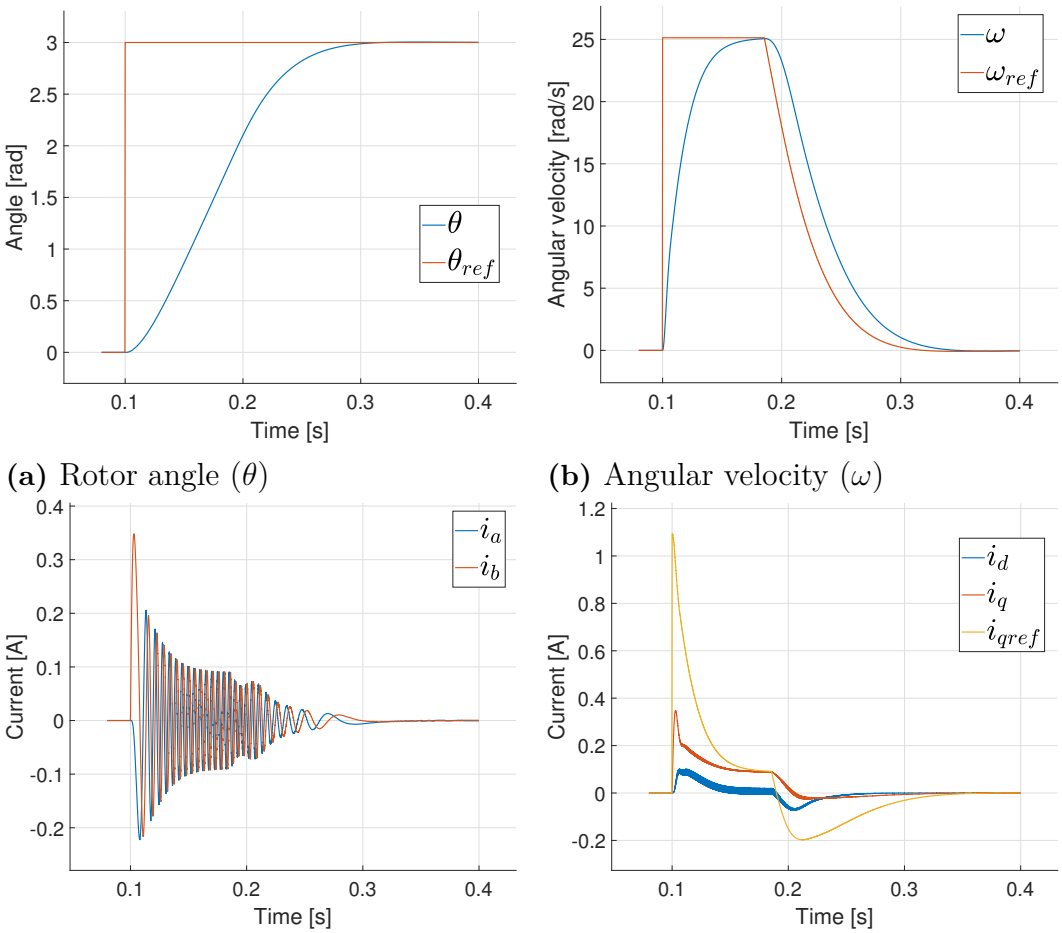
(c) Park currents ($i_d,\ i_q$)

(d) Phase voltage ($v_\alpha,\ v_\beta$)

Figure 4.4: Step response for the speed PID controller

4.1.3 PID position control

As with the speed PID controller of the previous section this controller was also tuned using the method in Section 3.3.1 with the resulting gains presented in (3.3). Figure 4.5a shows the step response for the position, where the controller is able to move the motor from 0 to 3 radians in around 0.2 seconds. When using the position controller the speed had to be limited due to hardware restrictions which will be explained later in Section 4.3.3. In Figure 4.5b it can be seen that the speed reference is selected proportionally to the error in position, as the position approaches the target the reference speed decreases. It is also possible to see the effect of the speed limitation as the reference signal is clipped at 8π rad/s.



(a) Rotor angle (θ)

(b) Angular velocity (ω)

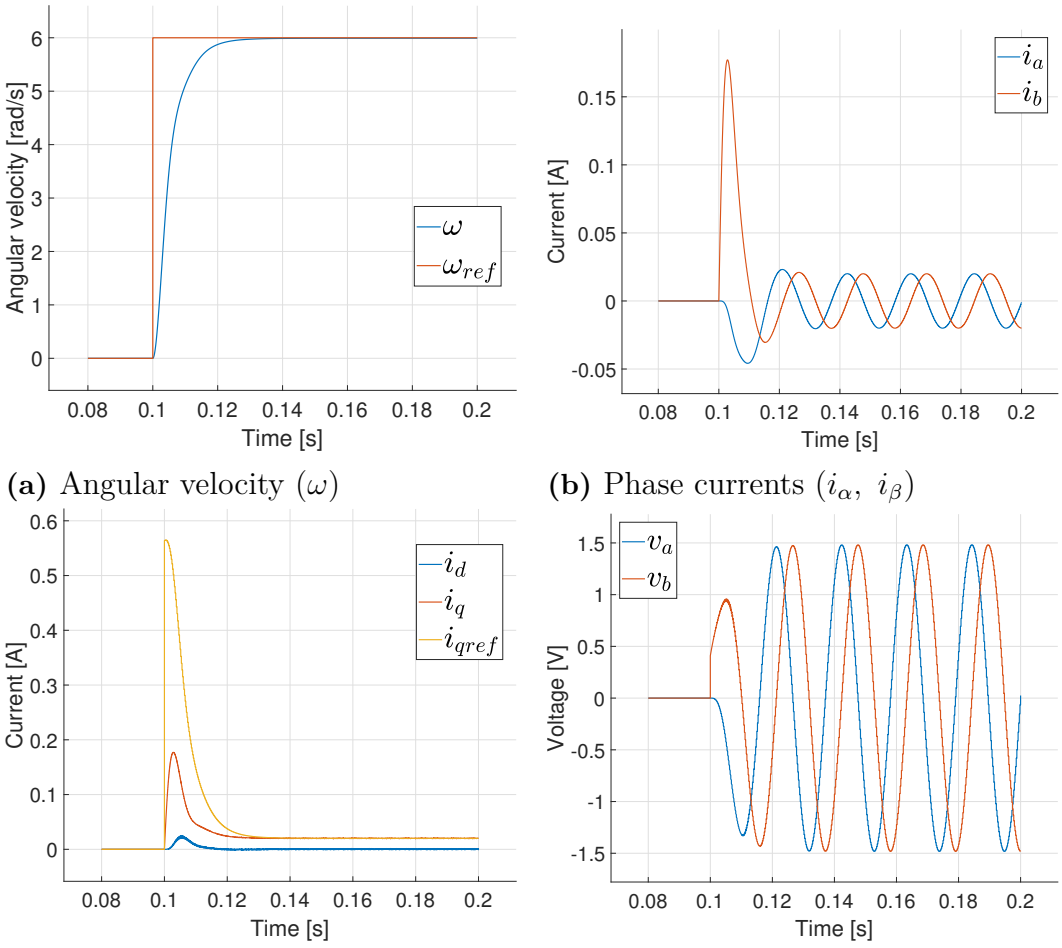
(c) Phase current ($i_\alpha,\ i_\beta$)

(d) Park currents ($i_d,\ i_q$)

Figure 4.5: Step response for the position PID controller

4.1.4 LQR velocity control

The LQR velocity controller was tuned by modifying the state and input costs, Q and R. When controlling the speed there is only one state (ω) and one input (T) which directly affects that state, which means that the dimensions of R and Q are both 1. R and Q where chosen by a trial and error process where R was used to limit the input to stay within the limits and Q was used to influence the rise time of the velocity state. Figure 4.6a shows the resulting step response, which outperforms the PID step response shown previously in Figure 4.4a (0.03 s vs 0.06 s). This indicates either that PID is not suitable for this task, or that it is not properly tuned. It is also important to keep in mind that the LQR is only as good as the model. In the simulations the LQR has access to a perfect model of the stepper motor that it is controlling. In reality the model will likely not be as good as in simulations.



(a) Angular velocity (ω)

(b) Phase currents ($i_\alpha,\ i_\beta$)

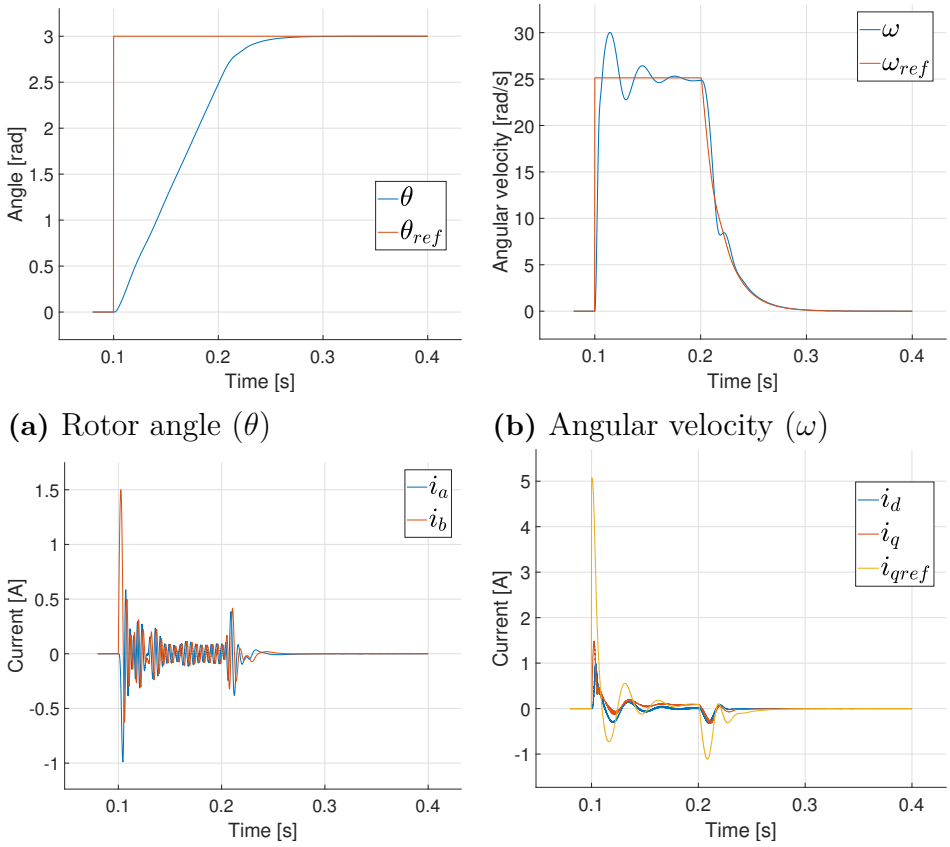
(c) Park currents ($i_d,\ i_q$)

(d) Phase voltage ($v_\alpha,\ v_\beta$)

Figure 4.6: Step response for the speed LQR controller. The LQR outperforms the PID shown in Figure 4.4 by a significant amount.

4.1.5 LQR position control

For the LQR position control the state was expanded to include both speed and position, while the input being the same and thus affecting the speed state directly and the position indirectly. This results in a Q matrix with dimension 2 and an R matrix of dimension 1. The purpose of this controller was to achieve a given position setpoint as fast as possible, while keeping speed restrictions. Using this mindset, the costs for the state (Q) was chosen to penalize the position state and the velocity state roughly equally. This type of control requires a trajectory planner to work properly, as the reference is only for one state. The trajectory planner selected a setpoint speed at each point that was proportional to the positional state error. This results in a control law that tries to keep both states at their references. In Figure 4.7a the controller is able to reach the setpoint for theta in about the same timespan as the PID controller in Figure 4.5, though the LQR is marginally faster, taking roughly 0.15 s as opposed to 0.2 s for the PID. However the speed is again limited to $\pm 8\pi$ rad/s due to hardware restrictions which will be explained in Section 4.3.3. The choice to limit the speed in simulation was made to enable easy comparison to the hardware results later on. Figure 4.7b shows the tracking for the velocity state, where the step response is similar to the LQR velocity controller in terms of rise time, though with some overshoot in this case. It is also possible to see how the trajectory planner picks the reference for the velocity state (ω_{ref}) in Figure 4.7b as the reference changes dynamically as the position error changes.



(a) Rotor angle (θ)

(b) Angular velocity (ω)

(c) Phase current ($i_\alpha,\ i_\beta$)

(d) Park currents ($i_d,\ i_q$)

Figure 4.7: Step response for the position LQR controller

4.2 Measurement results

4.2.1 Angle sensor

The angle sensor is specified with an angular resolution of 0.01°, however the sensor is not completely linear over the revolution of the motor, as depicted in Figure 4.8. The blue graph (which almost coincides with the red) represents the deviation of the sensor measurement versus the expected angle based on the requested field orientation. This discrepancy had to be compensated, which was achieved by the lookup table labeled as *calibration* in the same figure. The compensated measurement is shown as the yellow graph labeled *calibrated* and shows a significant improvement in the linearity of the angle (in the ideal case the yellow line would be constantly 0). Furthermore the *perceived* error, that is the difference between the calibrated angle and the expected angle, can be viewed on its own in Figure 4.9. While there is some error remaining over the revolution it is worth noting that the magnitude of this error is approaching the resolution of the sensor.

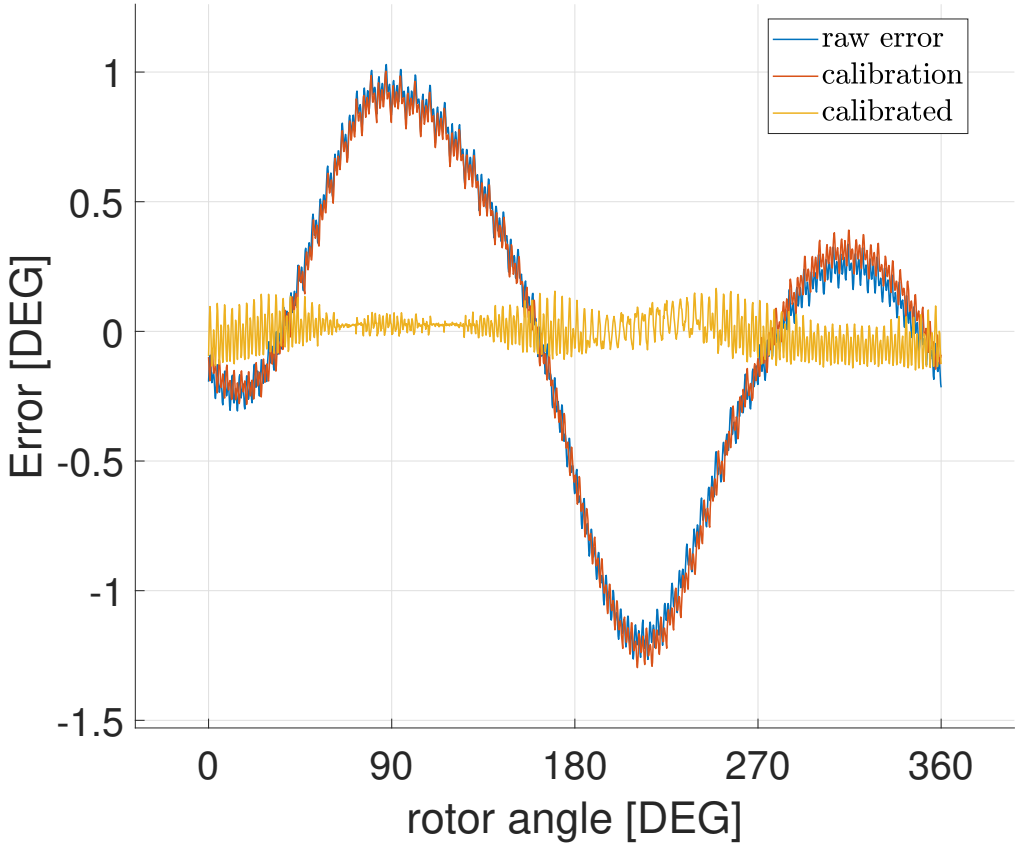


Figure 4.8: Result of the sensor calibration. The *raw error* (blue) is the measured angle compared to the expected open-loop angle. The *calibration* (red) is the lookup table used to compensate the error. The *calibrated* (yellow) is the result of using the lookup table to compensate the error.

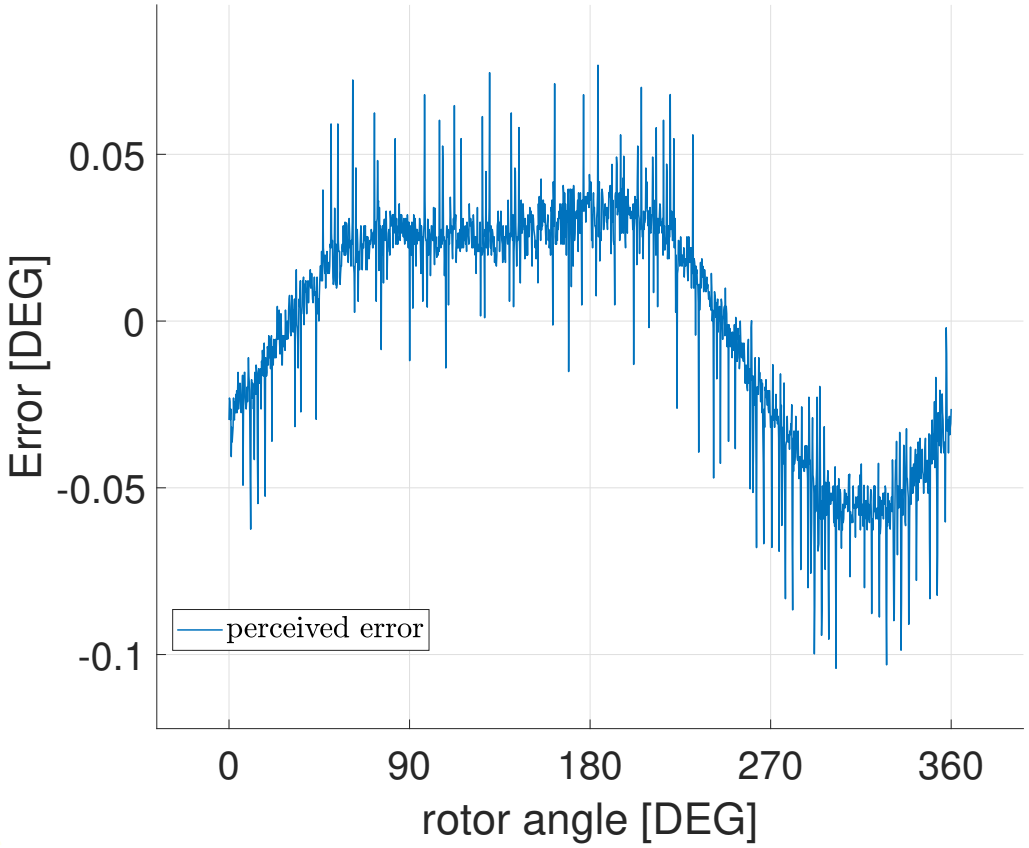


Figure 4.9: This figure depicts the measurement error calculated during the calibration test, the expected open-loop angle versus the compensated measurement.

4.2.2 Current measurements

Current measurements were achieved by the use of a shunt resistor connected in series with the load. The voltage over the resistor was monitored using a 20x-amplifier. From this voltage the current through the resistor is calculated using Ohm's Law. The sensing resistance was selected as 40 $m\Omega$, which amplified by 20 gives 0.8 V/A. In Figure 4.10 a plot of a voltage sweep over a motor can be viewed. The plot shows the current through the motor with regards to the commanded H-bridge voltage.

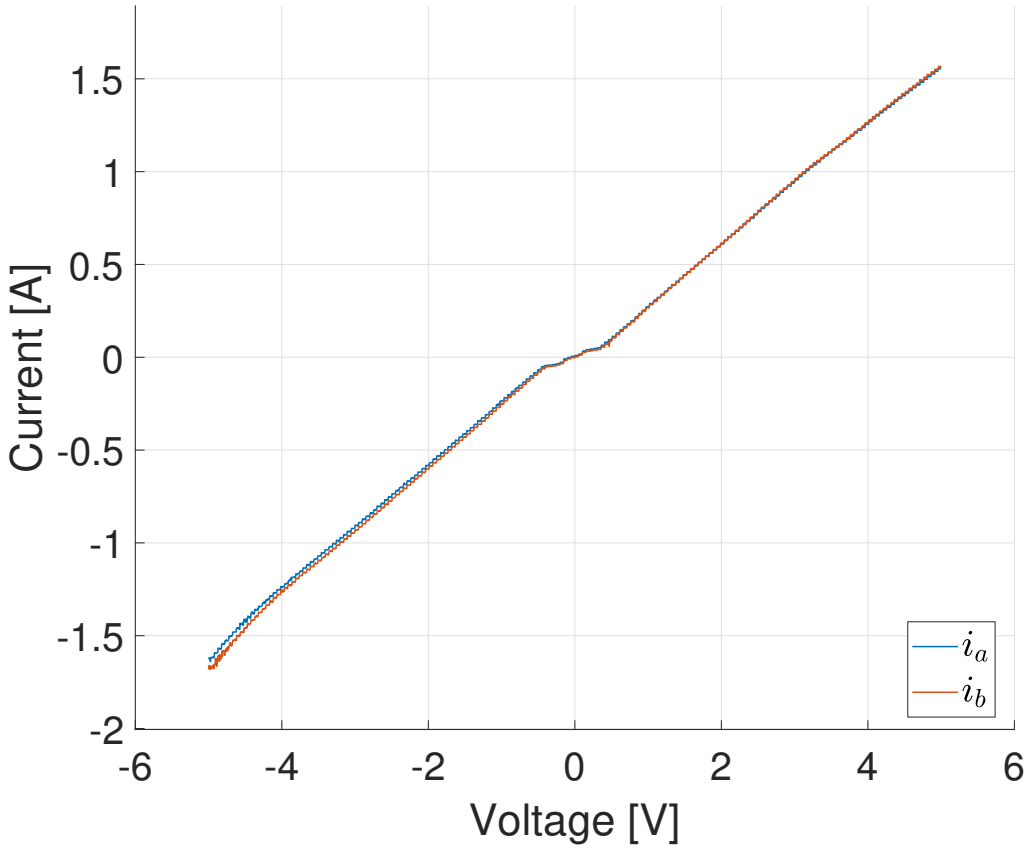
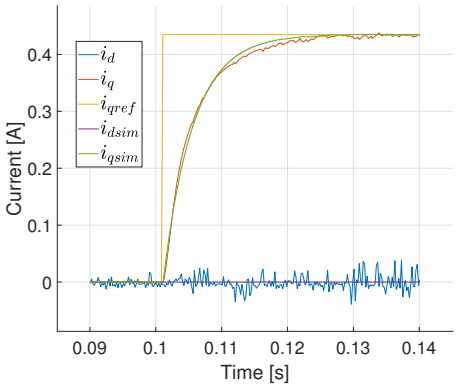


Figure 4.10: Plot showing the commanded voltage versus the measured phase currents

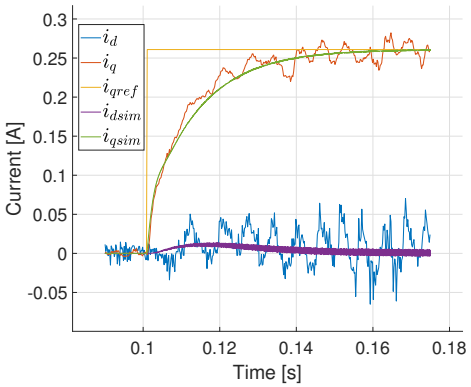
4.3 Motor control results

4.3.1 Current control

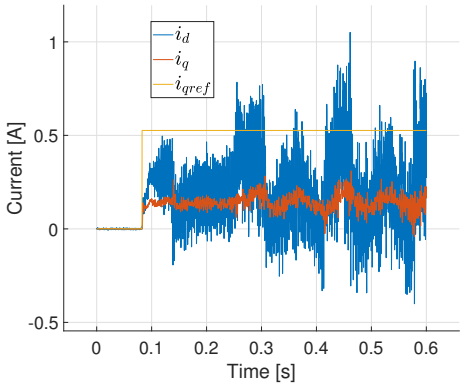
The current controllers in the C implementation performs in a similar way to the simulated controllers. Figure 4.11a shows the step response for the current controller when the rotor was fixed, the same scenario as was simulated in Figure 4.1a is also shown in the same figure. The step response for the motor attached to a dynamic load also shows similar behaviour to its simulation. The step response in hardware was slightly slower than the simulation, which likely was caused by filtering of the current which was done by the use of a lowpass filter. The lowpass filter adds a slight delay to the signal which explains the delay present in the hardware plots in both Figures 4.11c and 4.11b. Furthermore the step response for the freely rotating motor, shown in Figure 4.11c, presents a similar behavior to the simulation of the same case (Figure 4.1b). The controller is initially following the fixed case, but as the motor starts rotating and generating back-emf the current curve starts deviating from the fixed rotor case.



(a) Step response when the rotor is fixed during the step.



(b) Step response when the rotor is subjected to a load proportional to rotation speed.



(c) Step response when the rotor is free to rotate during the step

Figure 4.11: Step response for the current controllers on the hardware.

4.3.2 PID velocity control

The result of the step response for the velocity controller can be seen in Figure 4.12. Here again the graph is slightly delayed compared to the simulations. This is caused by the same reason as before, low pass filtering of the speed estimation. The oscillations that can be seen in the estimated speed could be caused by the manner in which the speed is estimated, numerical derivation. As there is no speed sensor the speed is estimated from the angle sensor. This results in the speed spikes whenever the sensor advances from one angle to the next, while resulting in 0 speed when the sensor has not changed between measurements. On average this gives the correct speed, however it requires filtering. The harder the filter the more accurate the estimation becomes in the long run for a constant speed. The trade off is that this adds a delay to the speed estimation. As can be seen in Figure 4.12, the filtering was chosen in such a way to give a small delay and remove as much oscillations as possible, meaning there is a small delay with some oscillations.

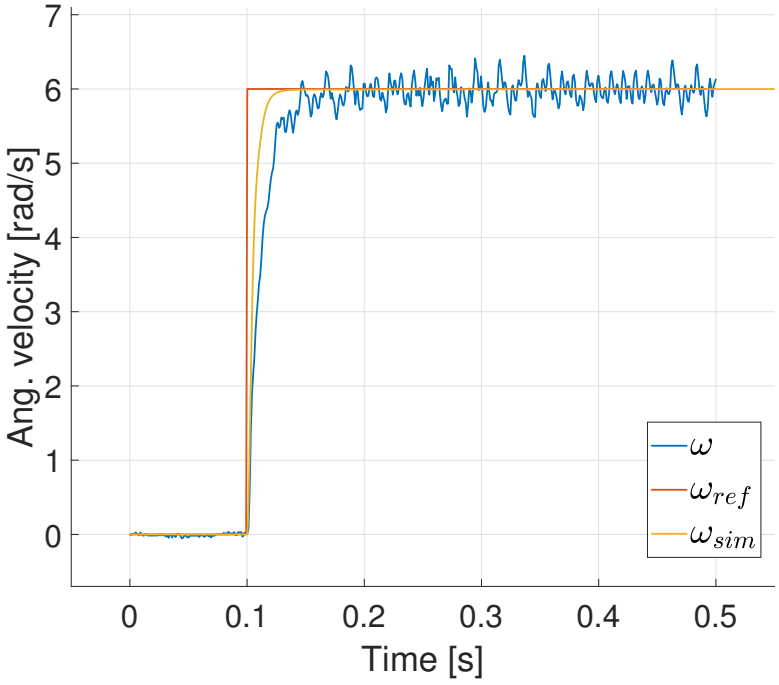


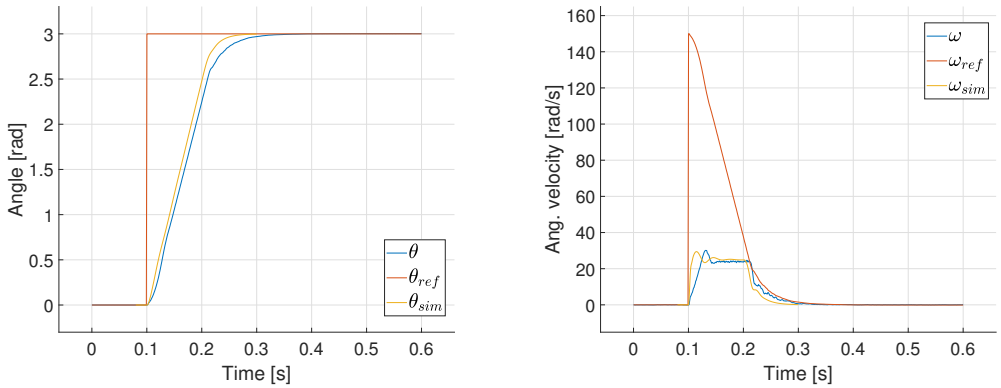
Figure 4.12: Step response for the speed PID controller given a constant step.

4.3.3 Speed limiting

When using a position controller there is no direct control over what speed the motor will run at after the tuning, unless you impose restrictions to the controller. In this case, due to a combination of hardware and software issues the motor was not able to run faster than about $\pm 10\pi$ rad/s. Due to time limitation these issues and their interactions were not fully identified. The hardware issues were probably the noise in both current and angle sensors, while the software issues were likely the speed estimation in combination with the sensor noise.
Exceeding this speed caused the FOC to apply the wrong torque to the motor causing the system to become unstable. Because of this the speed was limited to $\pm 8\pi$ rad/s during the tests to stay on the safe side.

4.3.4 PID position control

The PID position controller is a cascade controller that builds upon the velocity controller from the previous section. The position PID outputs a desired velocity to the velocity controller. The resulting step response for the rotor angle (θ) can be seen in Figure 4.13a and the velocity (ω) is shown in Figure 4.13b. The controller is able to perform a 3 radian step in roughly 0.2 seconds when the speed was limited to $\pm 8\pi$ rad/s. The controller follows the speed reference well with only a minor overshoot at the first step when viewing the ω estimation. When comparing to the simulation (yellow in Figure 4.13b) the low pass delay is apparent again, otherwise the simulation and the hardware are very similar for the velocity. The same can be said for the rotor angle (θ), with the exception that the delay here is not caused by a low pass filter on the angle, but the same low pass filter as before, acting on the velocity.



(a) Rotor angle (θ) **(b)** Angular velocity (ω)

Figure 4.13: Step response for the position PID controller. The controller performs the step in roughly 0.2 seconds while being limited to $\pm 8\pi$ rad/s. The speed reference was followed with only a minor overshoot when viewing the ω estimation.

4.3.5 LQR velocity control

The LQR velocity controller performs close to the simulation, as can be seen in Figure 4.14. The blue line represents the estimated velocity on the hardware test while the yellow line is the speed estimation in simulation. The estimation in hardware is close to the simulation until close to the reference, where some oscillations start. These are the same oscillations that were present for the PID controller mentioned in Section 4.3.2. Additionally the LQR controller was not able to remove the steady state error as fast in hardware as compared to simulations, despite the use of an integrating state. The likely reason for this was that the cost (Q) of the integrating state could not be large enough without the system becoming uncontrollable and causing the *lqi*-command in MATLAB to fail. The initial steady state error which prompted the use of an integrating state was likely caused by model error.

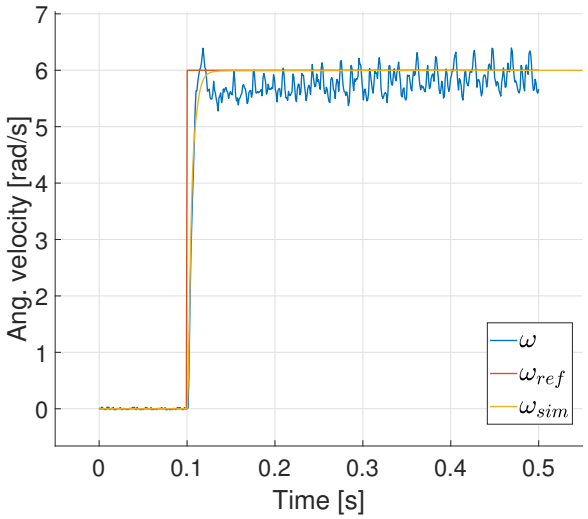
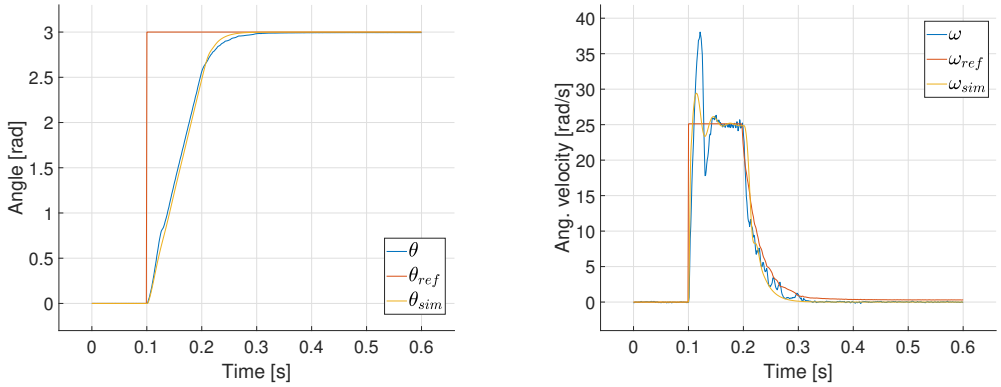


Figure 4.14: Step response for the LQR velocity controller. The simulated speed estimation (ω_{sim}) is very close to the actual estimation (ω) from the hardware test. Some oscillations occur close to the reference after the step has been given.

4.3.6 LQR position control

In Figure 4.15 the results from the LQR position step responses can be seen. The step responses in simulation (yellow) and in hardware (blue) are very close to each other in both velocity and position. The larger overshoot present in the velocity state compared to simulation was likely caused by the filtering of the velocity state in the hardware, which was not modelled in the simulation. This suggests that the model accurately captured the dynamics of the stepper motor. This controller is also affected by the speed limitation discussed earlier, even more so than the PID due to the fast response in velocity.



(a) Rotor angle (θ) **(b)** Angular velocity (ω)

Figure 4.15: Step response for the position LQR controller

4.4 Discussion

4.4.1 Angle sensor

The challenges when developing a FOC for a stepper motor mainly boil down to accurately measuring the rotor angle and the phase currents while outputing the correct voltage on the H-Bridge. The angle is vital as this determines the values and orientation of the phase voltages through the inverse park transform. This is especially prevalent in stepper motors due to their large number of poles. Every pole multiplies the number of electrical revolutions per mechanical revolution, effectively dividing the resolution of the angle sensor proportionally with regards to discrete steps per electrical revolution. This further increases the demands on the angle sensor accuracy. During this project it was found that while the angle sensor was very accurate it was not very linear. It consistently reported the same measured angle for the same orientations, however it did not always present an angle that matched the physical orientation, sometimes differing more than 1° which can be seen in Figure 4.8. Fortunately, due to the repeatable nature of the measurements this was remedied by the use of a lookup table, mapping the measured angles to the actual rotor positions. This lookup table was created by driving the motor using an open-loop control scheme and recording the commanded angles versus the measured angles. The creation of the lookup, the calibration, is only required once per motor/driver pair as this is stored in the flash memory of the ESP32. The use of this calibration improved the linearity of the sensor greatly, as shown by the yellow line in Figure 4.8. To put this in to context, for a 200 steps per revolution motor this error equates to ±5° accuracy on the electrical revolution.

4.4.2 Phase current

The current measurements were another challenge faced during this project. Because the FOC is mainly a current controller the current measurements are another aspect vital to the controllers functionality. The shunt resistance had to be selected with care to make sure that it was small enough to not interfere with the phase current too much, while still being large enough to produce a voltage that could be measured. In the end it was selected as 40 $m\Omega$ which resulted in a voltage drop of 40 mV/A and a current conversion of 0.8 V/A after the amplification of 20. In order to improve the performance of the onboard ADC the amplifier that was used outputs a biased voltage. The ADC in the ESP32 performs best in the region around $V_{cc}/2$, which is where the voltage equivalent to 0 A was placed. Positive currents caused the voltage to increase and negative resulted in a decrease, allowing the ADC to operate in its best performing region most of the time.

4.4.3 H-Bridge voltage

In order to place the correct voltage over the motor phases the voltage used to calculate the duty cycle of the PWM had to be compensated to take into account the losses in the H-Bridge and the shunt resistance. In the H-Bridge there is a voltage

drop for every MOSFET the current passes through, which normally is two. Because the resistances in the H-Bridge and shunt resistor are small the compensated voltage is largely dependant on the motor resistance. For a motor with larger resistance the compensated voltage was very close to the uncompensated, while for motors that have $< 2\ \Omega$ in phase resistance the voltages varied significantly, which is expected when studying (3.16) and (3.17).

4.4.4 Current control

The PI current controllers in the FOC performed very well. The rise time in both simulation and hardware matched the rise time selected during gain calculation (Figure 4.1a and 4.11a), providing the motor was kept still or was loaded. When the rotor was allowed to move unlaoded the rotor accelerated very quickly which in turn caused high back-emf to be generated. The back-emf reached a voltage higher than the source voltage before the controllers could reach their respective setpoints (Figure 4.1b and 4.11c), which caused them to level off at a point lower than the setpoint. This can be seen in both the simulation and hardware. Given ideal inputs (unlimited source voltage, ideal DAC) the setpoint is eventually reached, as was showed in Figure 4.3.

4.4.5 Velocity control

Two different controllers were tested for velocity control, PID and LQR. When comparing the step responses for the controllers in simulation (Figure 4.4a and 4.6a) it is clear that the LQR performed slightly better. The reason for this could insufficient tuning of PID controller. The tuning of the PID followed the step by step process described earlier in Section 3.3.1, which might not be the most ideal way of tuning. The LQR on the other hand did not require much tuning at all to reach a better result than the PID.
In hardware the results show the same tendency as in simulation, the LQR is faster to reach the setpoint (Figure 4.12 vs. 4.14). However the LQR seems to have some issues with fully reaching the setpoint, even with an integrating state (Figure 4.14), an issue which the PID does not seem to have (Figure 4.12) to the same extent.

4.4.6 Position control

Here again PID and LQR were tested, and again LQR is performing the best in the simulations (Figure 4.5a and 4.7a). The PID is slower to reach the setpoint compared to the LQR and furthermore the LQR has the option of providing a reference for the velocity as well. This enables a "soft" speed limit to be set, soft meaning that the controller might violate the speed reference, but will generally keep to it (Figure 4.7b).
In contrast to previous controllers the PID and LQR seems to have similar performance on the hardware. A possible reason for this is likely the speed restriction which had to be imposed in the position cases. This restriction limited the advantage the LQR had by being faster to accelerate, and thus potentially reaching a

higher top speed than PID. Another reason for the similar performance might be due to the selection of velocity reference for the LQR. The velocity reference was selected by means of a proportional gain on the position error, effectively a P controller. This might not have been the ideal reference and more performance could be possible with a proper selection of reference trajectory.

5
Conclusion

This master's thesis investigated model based control (LQR) combined with Field Oriented Control for permanent magnet stepper motors. The FOC, the LQR and a PID controller were simulated and implemented on the target hardware. The FOC controlled the currents in the motor to produce a desired torque, while the two other controllers provided the torque reference based on an external reference (velocity and/or position).

The FOC worked very well in both simulation and hardware. The currents reached their setpoints within the rise time specified during the controller design (10 ms) when the rotor was fixed or put under a dynamic load. The controller was not able to keep the setpoint if the motor was unloaded and allowed to reach a very high velocity, due to back-emf overpowering the source voltage available.

Both the PID and the LQR worked well in both simulation and hardware, though with the LQR controller outperforming the PID in both velocity control and in the position control in simulation as well as hardware. In velocity control the LQR was able to change speed from 0 to 6 rad/s twice as fast as the PID (0.03 s vs 0.06 s). For position control the difference was reduced. The LQR moved the motor from 0 to 3 rad in 0.15 s while the same movement was executed in 0.2 s for the PID. This reduced difference was attributed to speed limitations and reference trajectory selection.

The most suitable controller for stepper motor control depends on the availability of motor parameters. LQR requires a good knowledge of the motor parameters, while PID does not. Given close to correct motor parameters LQR gives the best performance, however PID is not far behind and is most suitable if the motor parameters are not readily available.

6
Future Work

6.1 Speed limitation

Identification and correction of the errors that lead to the speed being limited is the next step for this project. Without the limitation there would likely be a clearer winner between the LQR and PID is the position control test, the the LQR would not be as hampered.

6.2 PI-controller decoupling and back-emf compensation

With the speed limitation lifted the coupling that was disregarded in this project would have to be added, as this is much more prevalent at higher velocities. The same is true for the back-emf which also is highly dependant on the velocity. At low speeds this can be disregarded, but not at higher speeds. The addition of these two compensators would allow the current controller to perform better at higher speeds.

Bibliography

[1] Richard Barry. *FreeRTOS*. Feb. 2019. URL: https://www.freertos.org/index.html.

[2] M. Bendjedia et al. "Position Control of a Sensorless Stepper Motor". In: *IEEE Transactions on Power Electronics* 27.2 (Feb. 2012), pp. 578–587. ISSN: 0885-8993. DOI: 10.1109/TPEL.2011.2161774.

[3] *DMOS Microstepping Driver with Translator And Overcurrent Protection*. A4982. Rev. 5. Allegro Microsystems. May 2014.

[4] *DRV8825 Stepper Motor Controlelr IC*. DRV8825. Rev. F. Texas Instruments. July 2014.

[5] *DRV887x H-Bridge Motor Drivers With Integrated Current Sense and Regulation*. DRV8876. Rev. 1.0. TexasInstruments. Oct. 2018.

[6] *ESP32*. ESP32. Rev. 2.8. Espressif. Jan. 2019.

[7] *ESP32 Technical Reference Manual*. ESP32. Rev. 4.0. Espressif Systems. Dec. 2018.

[8] Espressif. *Espressif IoT Development Framework*. Feb. 2019. URL: https://github.com/espressif/esp-idf.

[9] Lennart Harnefors. *Control of Variable-Speed Drives*. Department of Electronics, Mälardalen University, Sept. 2002.

[10] hauptmech. *What are good strategies for tuning PID loops?* Oct. 2012. URL: https://robotics.stackexchange.com/a/174.

[11] B. Henke et al. "Modeling of hybrid stepper motors for closed loop operation". In: *IFAC Proceedings Volumes* 46.5 (Apr. 2013), pp. 177–183. DOI: 10.3182/20130410-3-CN-2034.00042.

[12] D. Mathew J Paul. "A Novel Vector Control Strategy For Bipolar Stepper Motor". In: *International Journal of Scientific & Engineering Research* 5.11 (Nov. 2014), pp. 1133–1139. ISSN: 2229-5518.

[13] W. Kim, C. Yang, and C. C. Chung. "Design and Implementation of Simple Field-Oriented Control for Permanent Magnet Stepper Motors Without DQ Transformation". In: *IEEE Transactions on Magnetics* 47.10 (Oct. 2011), pp. 4231–4234. ISSN: 0018-9464. DOI: 10.1109/TMAG.2011.2157956.

[14] Kitware. *CMake*. 2001. URL: https://cmake.org/overview/.

[15] J. Kordik. "Comparing Open Loop and StepSERVO Closed Loop Stepper Systems". In: (July 2015).

[16] *LMP860x, LMP860x-Q1 60-V, Bidirectional, Low- or High-Side, Voltage-Output, Current-Sensing Amplifiers*. LMP860x. Rev. G. TexasInstruments. Jan. 2014.

[17] P. Pillay and R. Krishnan. “Modeling of permanent magnet motor drives”. In: *IEEE Transactions on Industrial Electronics* 35.4 (Nov. 1988), pp. 537–541. ISSN: 0278-0046. DOI: 10.1109/41.9176.

[18] C. Rusu, I. Birou, and E. Szoke. “Model based design controller for the stepper motor”. In: *2008 IEEE International Conference on Automation, Quality and Testing, Robotics*. Vol. 2. May 2008, pp. 175–179. DOI: 10.1109/AQTR.2008.4588816.

[19] *TLE5012B*. TLE5012B. Rev. 2.1. Infineon. June 2018.

[20] *TMC2130-LA DATASHEET*. TMC2130. Rev. 1.10. Trinamic Motion Control GmbH & Co. KG. May 2018.

[21] M. Zribi and J. Chiasson. “Position control of a PM stepper motor by exact linearization”. In: *IEEE Transactions on Automatic Control* 36.5 (May 1991), pp. 620–625. ISSN: 0018-9286. DOI: 10.1109/9.76368.