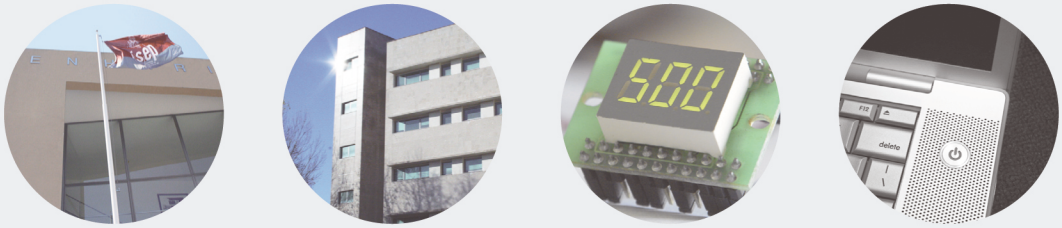
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



# Advanced Three-Phase Grid Synchronization Using Synchronous Reference Frame Phase-Locked Loops

**JOSÉ EDUARDO SANTOS MARAVALHAS SILVA**
outubro de 2019

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

# Advanced Three-Phase Grid Synchronization Using Synchronous Reference Frame Phase-Locked Loops

**José Eduardo Silva**


Master's Degree in Electrical Engineering

Supervisor: Prof. Rui Miguel Monteiro de Brito, PhD

30 September, 2019

© José Eduardo Silva, 2019

*"So long and thanks for all the fish."*

Douglas Adams

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Abstract

Modern power electronics devices require grid synchronization to accurately time the switching of their semiconductor devices. This project steps through the development of such an algorithm for three-phase grids.

The classical synchronous reference frame phase-locked loop is studied in depth, including a detailed analysis of the transforms that give rise to its name. A few improvements are added in order to mitigate some of the well-known pitfalls of this method.

Using the theory of symmetrical sequence components, new equations that describe the behaviour of said components in relation to unbalanced three-phase voltages are derived. These equations are then used to better understand the behaviour of the classical algorithm under unbalanced conditions. From this, an advanced grid synchronization algorithm based on multiple phase-locked loops is developed. This algorithm is then discretized and implemented in a typical microcontroller.

Finally, a custom genetic algorithm is used to fine-tune the parameters of the algorithm to a specific simulated scenario meant to represent harsh grid conditions.

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Acknowledgements

This work is dedicated to my wife, Inês, and my son, Tiago.

I would like to thank my supervisor, Prof. Rui Brito, for his selfless support and for encouraging me to pursue an often overlooked topic.

I wish to express my sincere thanks to my colleagues and friends at the *Laboratório de Sistemas Autónomos* who were always there to lift the spirits in times of failure and frustration and pushed me to finish this work.

I would also like to extend my gratitude to the remaining members of my family for their support during such a critical time.

Last but not least, I would like to thank my friends, Francisco and Matilde, for those fun movie nights which definitely helped to reduce my thesis-induced stress levels.

José Eduardo Silva

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Contents

THIS PAGE
INTENTIONALLY
LEFT BLANK

# List of Figures

# List of Tables

THIS PAGE
INTENTIONALLY
LEFT BLANK

# List of Abbreviations

| | |
|---|---|
| AC | **A**lternating **C**urrent |
| ADC | **A**nalog-to-**D**igital **C**onverter |
| ANS | **A**mplitude **N**ormalization **S**cheme |
| DAC | **D**igital-to-**A**nalog **C**onverter |
| DC | **D**irect **C**urrent |
| DFT | **D**iscrete **F**ourier **T**ransform |
| DQ0 | **D**irect-**Q**uadrature Zero |
| DQ0-DA | **D**irect-**Q**uadrature Zero D Aligned |
| DQ0-QA | **D**irect-**Q**uadrature Zero Q Aligned |
| DSC | **D**elayed **S**ignal **C**ancellation |
| EKF | **E**xtended **K**alman **F**ilter |
| FFF | **F**requency **F**eed-**F**orward |
| FPU | **F**loating-**P**oint **U**nit |
| GA | **G**enetic **A**lgorithm |
| I-SRF-PLL | **I**mproved **S**ynchronous **R**eference **F**rame **P**hase-**L**ocked **L**oop |
| ISEP | ***I**nstituto **S**uperior de **E**ngenharia do **P**orto* |
| LCC | **L**ine-**C**ommutated **C**onverter |
| LED | **L**ight-**E**mitting **D**iode |
| LPF | **L**ow-**P**ass **F**ilter |
| LTI | **L**inear **T**ime **I**nvariant |
| LUT | **L**ook-**U**p **T**able |
| PI | **P**roportional-**I**ntegral |
| PLL | **P**hase-**L**ocked **L**oop |
| PNSC | **P**ositive and **N**egative **S**equence **C**alculation |
| PWM | **P**ulse-**W**idth **M**odulation |
| QSG | **Q**uadrature **S**ignal **G**enerator |
| RAM | **R**andom-**A**ccess **M**emory |
| RMS | **R**oot-**M**ean-**S**quare |

| | |
|---|---|
| SOGI | **S**econd-**O**rder **G**eneralized **I**ntegrator |
| SRF-PLL | **S**ynchronous **R**eference **F**rame **P**hase-**L**ocked **L**oop |
| UKF | **U**nscented **K**alman **F**ilter |
| VSC | **V**oltage **S**ource **C**onverter |

# Chapter 1

# Introduction

## 1.1 Contextualization and Motivation

With the ever-increasing global energy consumption, the need for efficient power electronics solutions has never been greater, with switching converters taking center stage. While there are many types of switching converters, some for direct current (DC), some for alternating current (AC), and others even converting between DC and AC, they all share a key similarity: none of these converters make use of resistive or linear-mode semiconductor devices. This allows for high efficiency, usually at the cost of complex control systems. In contrast, some non-switching converters do not require control systems, relying only on the physical properties of their semiconductor devices to function[1, 2].

The control of switching grid-connected converters is heavily reliant on proper grid synchronization techniques that provide crucial information about the current state of the electrical grid. This information is then used to generate switching signals for the semiconductor devices based on the particular goal of that grid-connected converter[2].

In order to further develop the skills obtained during the Master's Degree in Electrical and Computer Engineering at the *Instituto Superior de Engenharia do Porto* (ISEP), an advanced grid synchronization technique based on the widespread Synchronous Reference Frame Phase-Locked Loop (SRF-PLL) is developed and implemented in a microcontroller-based system.

The goal of this project is to use the popular three-phase SRF-PLL grid synchronization technique as a starting point and combine various improvements to mitigate or even eliminate its downsides.

Hopefully, the self-contained nature of this project also serves a double purpose of introducing future fellow students to the topic at hand, as information tends to be scattered across multiple books - which usually lean towards more generalist overviews and do not present improvements on the classical SRF-PLL - and research papers - which tend to be too focused on a single aspect of the technique.

## 1.2 Considerations About Project Scope

Considering that synchronization algorithms are a vast research topic, it is important to take into account what the actual scope of the project is.

While this project is heavily reliant on simulations, it does not intend to be a detailed guide in how to properly perform tests for safety-critical applications. Simulated conditions are "relaxed" throughout the project, as their sole purpose of existence is to visually demonstrate to the reader the concepts being explained and shouldn't be interpreted as presentations of absolute proof for any given statement. The reader should always refer to the provided references for such proof. As such, simulated conditions are not necessarily consistent across multiple simulations, as different conditions can be used to exacerbate differences in the behaviours being explained.

Unless otherwise specified, this project makes no assumptions about the type of hardware or power electronics converters in which the developed synchronization algorithm could be deployed. This project aims to present an algorithm that is generic enough to be applied to any use-case, even if such use-cases are rare. In doing so, the reader should be able to easily adapt the methodologies presented in this document to their specific requirements.

Finally, this project does not aim to rigorously compare the developed algorithm with other algorithms. It is the author's hope that by heavily focusing on studying and developing a single algorithm, the quality of the information presented to the reader becomes much higher than it would otherwise be.

## 1.3 Objectives

The main goal of the project, as stated before, is to develop and implement an advanced grid synchronization algorithm based on the SRF-PLL. To do this, the following objectives should be achieved:

- Study other grid-synchronization techniques and how they compare to the SRF-PLL
- Study the transforms that give the SRF-PLL its characteristics
- Improve on the classical SRF-PLL
- Demonstrate how grid imperfections impact the accuracy of the SRF-PLL
- Develop an advanced synchronization algorithm using the improved SRF-PLL
- Implement the algorithm in a microcontroller and perform hardware benchmarks
- Optimize tuning parameters

## 1.4 Thesis Structure

Chapter 2 describes the current state-of-the-art. This chapter provides a short overview of grid-connected converters and the need for grid synchronizations algorithms. Then, explanations for different synchronization techniques are presented.

In Chapter 3 two different transforms that simplify three-phase voltages (or currents) are presented and explored in depth.

In Chapter 4 the classical SRF-PLL is deconstructed into its basic components and explained step-by-step. In this chapter the reader is shown how the transforms explained in Chapter 3 play a crucial role in the SRF-PLL.

In Chapter 5 the decomposition of three-phase voltages into their symmetrical sequence components is studied in depth, providing equations that relate the parameters of the components to the parameters of the three-phase voltages. Then, this chapter presents a short analysis of the impact of the symmetrical sequence components on the SRF-PLL, and provides a method for extracting all of the symmetrical sequence components, as well as how to extend the SRF-PLL grid synchronization technique to fully estimate all variables of said components.

Chapter 6 describes the discretization and implementation of every single part of the grid synchronization technique. Hardware benchmarks are performed in order to better understand the feasibility of implementing the technique in a modern microcontroller.

In Chapter 7 a custom genetic algorithm is developed to optimize the tuning parameters to a specific simulated scenario.

Finally, Chapter 8 contains the conclusions and possible ideas for future work.

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Chapter 2

# State-of-the-Art

## 2.1 Introduction

In all controllable converters that connect to the electrical grid it is necessary for the control system to know the exact moment when to switch the semiconductor devices. This chapter provides a few examples of such converters and why proper grid synchronization algorithms are an important part of their control systems.

This chapter guides the reader through some of the existing techniques and presents multiple brief analyses of the inner workings of each method, highlighting their most important advantages and disadvantages.

## 2.2 Grid-connected Converters and the Importance of Grid Synchronization

### 2.2.1 Thyristor Bridge

A thyristor bridge is an improvement on the classical diode bridge rectifier. Substituting the diodes by thyristors creates a converter with one degree of freedom: when to turn on the thyristors. Additionally, a large value of inductance in the DC side guarantees that the DC side current is nearly constant, which results in improved performance as total harmonic distortion is significantly reduced[1].

The voltage on the DC side can be controlled via the firing angle, which also controls the power factor. However, the reactive power is not fully controllable, as the converter always "consumes" reactive power[2]. Converters such as this one are known as line-commutated converters (LCCs)[2] in view of the fact that the semiconductor devices are partially dependent on the grid's voltage waveforms to switch on and off. Specifically for thyristors, even if a turn-on signal is provided, the voltage across the device must be positive in order for it to actually turn on. Then, the turn-off occurs by itself when the current is reduced below the holding current[1, 2].

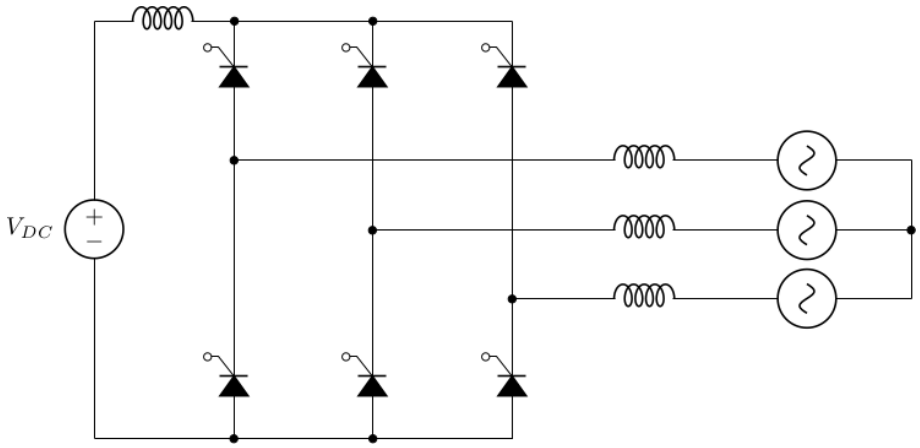



Figure 2.1: Three-phase thyristor bridge

The firing angle can be understood as the "delay" the thyristor takes to switch on in comparison with a normal diode. As such, the control of this converter directly depends on a good synchronization with the grid, as any error in the estimated grid angle will affect the firing angle, which in turn influences the performance of the converter.

As this converter has a quite simple control scheme, the grid synchronization technique can also be simplistic as there is no point in gathering a lot of highly detailed information on the current state of the grid if the converter has no way to act on this information.

### 2.2.2 Voltage Source Converter

A three-phase Voltage Source Converter (VSC) is a type of switching converter capable of bi-directional AC/DC power transmission[2]. The two-level VSC variant is shown in Figure 2.2.

By switching the transistors in each "arm" of the converter according to a reference sine wave, an output waveform is generated. This output waveform will contain more high-amplitude harmonics the less levels the converter has[2]. This is shown in Figure 2.3, where square-wave and pulse-width modulation (PWM) switching techniques are shown for two-level VSCs, and three-level PWM and multilevel switching are shown for three-level VSCs and multilevel VSCs, respectively.

As opposed to the thyristor bridge, the semiconductor devices in VSCs can be switched on and off at will[1]. This implies that VSCs are much more flexible and thus can be used in less conventional ways. For instance, a VSC can be tasked with injecting currents into the grid that cancel out harmonics generated by other devices instead of a more traditional AC to DC - or vice versa - power conversion[2]. Additionally, VSCs also see significant usage in the control of electric machinery[3], in which case the algorithms for grid synchronization are sometimes used to estimate parameters of said electric machinery.

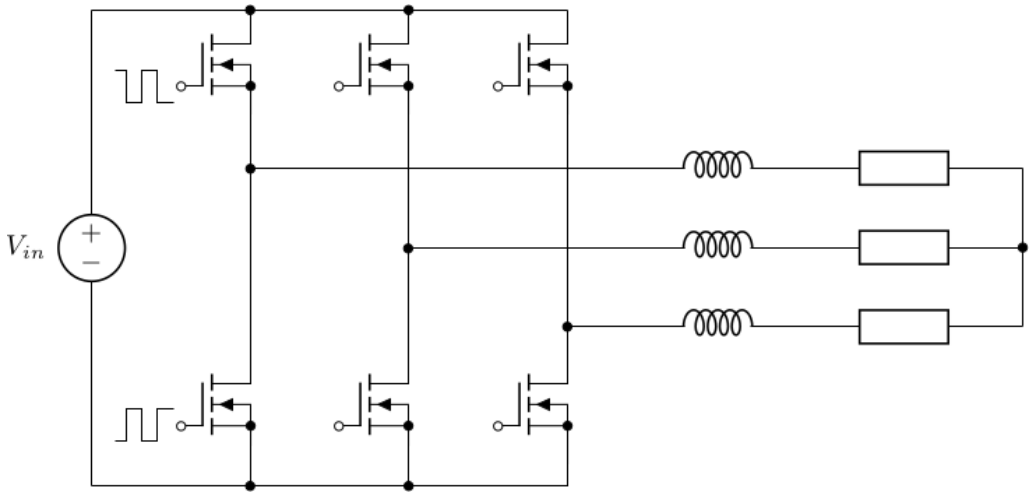



Figure 2.2: Three-phase VSC, two-level variant

In view of these additional capabilities, the grid synchronization algorithms tend to be more advanced, sometimes even including estimation of certain harmonics that are of particular interest[4]. Based on this information, the control system can then selectively generate output voltages to cancel said harmonics. The control of these converters is also much more complex, usually requiring non-trivial amounts of processing[2].

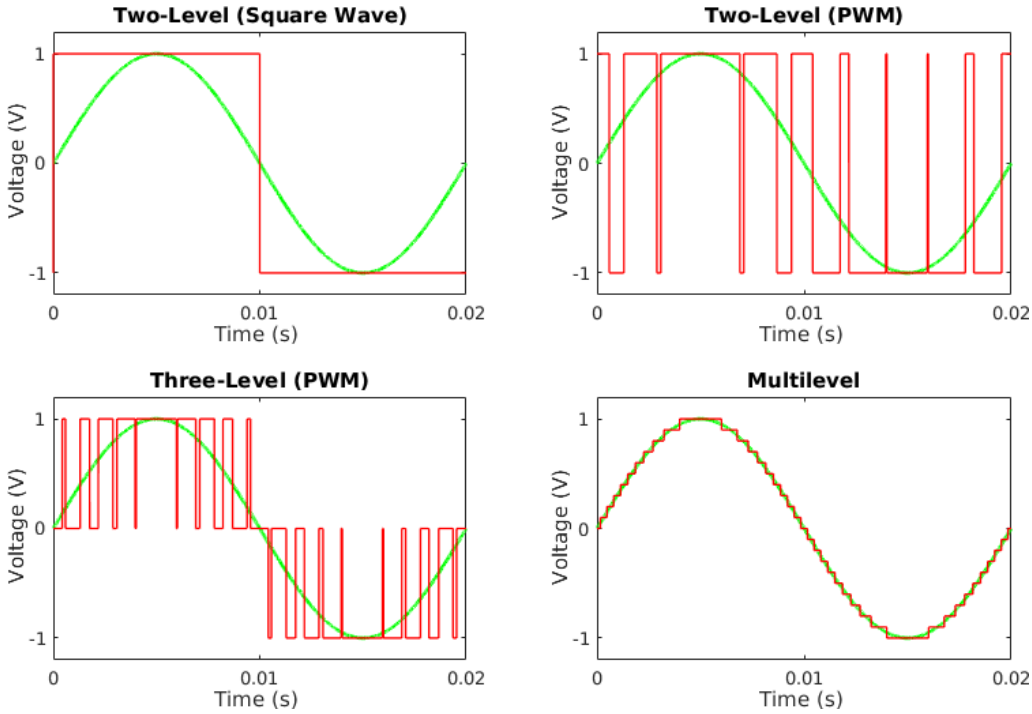



Figure 2.3: Example output waveform of a single phase in different VSCs

## 2.3 Grid Synchronization Techniques

### 2.3.1 Zero-Crossing Detection

One of the simplest synchronization techniques available to power converter designers is the zero-crossing detection method. As the name implies, this open-loop technique is based upon detecting the moments when the sinusoidal wave crosses the zero point, thus allowing a frequency measurement each half-period[5, 6]. Assuming that the system designer implements circuitry which allows the control system to differentiate between positive-to-negative and negative-to-positive zero-crossings, then the grid angle can also be determined each half-period. Additionally, the angle between half-periods can be estimated by dead-reckoning based upon the measured signal frequency and the time that has passed since the last zero-crossing detection.

While extremely simple and easy to implement, the zero-crossing method suffers from a plethora of problems[5, 6]. Noise and harmonic distortion can cause multiple outputs to occur for the same zero-crossing, and delays in the components used for the zero-crossing detection directly translate into phase detection errors. In order to mitigate some of these problems, designers have come up with different improvements[5]:

- Low-Pass Filtering
  - Applying a low-pass filter to the input signal works well for signals with small frequency deviations which have easily separable harmonics. If the frequency is known *a-priori*, the filter's phase shift can be mathematically compensated as long as its characteristics are predictable, which is the case for digital filters. However, the same cannot be said for analog filters constructed from physical components such as resistors and capacitors whose values can vary wildly with temperature, age (i.e. material degradation) and other similarly unpredictable factors.
- Digital signal conditioning and post-processing
  - In order to deal with multiple outputs for the same zero-crossing and erratic outputs due to voltage "notches" caused by commutation of power converters, a microcontroller can be programmed to ignore zero-crossing detections which do not meet the expected timing requirements from previous measurements. This is known as rule-based filtering. In conjunction with other digital processing techniques, this approach can increase the robustness of the zero-crossing detection method with minimal computational resources. However, expecting the next zero-crossing detection to be close to the next half-period implies that the system is assuming that the frequency is highly stable, which limits the usability of the technique.

- Improved zero-crossing detection hardware

  - A variety of different hardware designs can be used to improve the detection. For instance, instead of a simple comparator which informs a microcontroller-based system when the sine wave is positive or negative, two comparators can be used: one which outputs a detection signal before the actual zero-crossing, and another which outputs a detection signal after. Then, the microcontroller can compare the outputs of both comparators and linearly interpolate between them to estimate when the zero crossing actually occurred. This type of hardware is shown to improve accuracy, but it is still not completely immune to phase error. An example of such hardware is shown in Figure 2.4, where the voltage drops of the optocouplers' light-emitting diodes (LEDs) and the external diodes cause the outputs to switch state before and after the zero crossing. In this example circuit, a major source of error would be manufacturing differences between the multiple diodes, resistors and optocouplers used.

While there are many possible improvements, some of them not mentioned in the previous list, truly eliminating phase errors is not just hard to achieve, it is near impossible[5]. Even if the delays caused by hardware imperfections (such as diode forward voltage in detector circuits[5]) and microcontroller response times can somehow be compensated for, this technique still relies on only two data points per wave period, thus not being able to capture any frequency fluctuations which happen in between the data points[6].

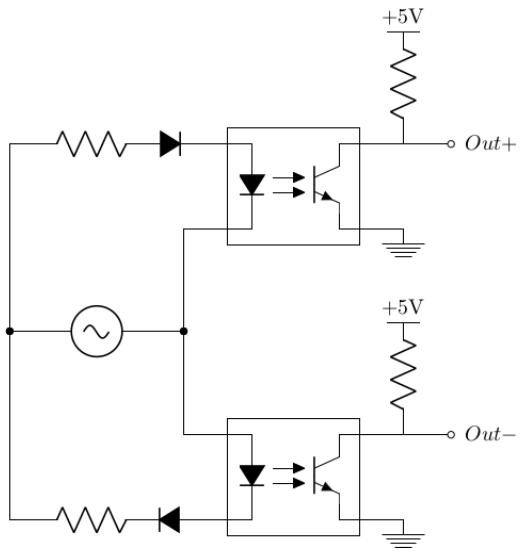



Figure 2.4: Example of a circuit capable of distinguishing between positive-to-negative and negative-to-positive zero-crossings

### 2.3.2 Fourier Transform

Fourier transform based techniques make use of the discrete Fourier transform (DFT) - which is normally applied to the calculation of a signal's spectrum - to estimate the phase angle. This technique offers significant advantages, such as being able to perform accurate angle estimation even when presented with highly distorted, noisy and unbalanced three-phase voltages without sacrificing response time[6, 7].

One particularly important advantage to DFT-based techniques is that it is an open-loop method, thus no parameter tuning is required. This makes it more predictable than other techniques whose tuning parameters are affected by the conditions of the grid [6].

Unfortunately, the downsides of this approach are quite significant as well. One such downside is that most methods use variable sampling rate techniques to cope with frequency variations[8]. This immensely complicates control systems, as they directly depend on the grid-synchronization algorithm[7]. To solve this, some methods use a fixed sampling rate and instead rely on a dynamically adjusted observation window to cope with distinct frequencies, but since the size of the window must be an integer number of samples, a high sampling rate must be used to achieve good frequency resolution[7]. Additionally, the problems that spectral leakage cause are usually not addressed.

Fourier transform based techniques are also extremely resource intensive, and even variations that attempt to reduce the computational complexity still require up to hundreds of additions and multiplications[6, 7]. Paired with the requirement for high-sampling frequencies, not only to achieve a good resolution of the observation window (for fixed sample-rate methods) but also because the control systems of power converters also require it[7], DFT-based techniques can be somewhat unsuitable for most modern microcontrollers, even high-performance ones. Some attempts to reduce the computational burden include interleaving specific steps of the algorithm between samples and linearly extrapolating what the outputs should be whenever said steps are skipped[7].

### 2.3.3 Kalman Filter

A Kalman filter is an algorithm capable of providing optimal estimation of system's state variables and their uncertainties[9, 10]. The first step of the Kalman filter is the predict step, where the state variables and their uncertainties are predicted using a model of the system. Then, the update step compares the results of the measurements with the predicted state, and the state is updated accordingly. An overview of the algorithm is shown in Figure 2.5.

The Kalman filter and its many variations have become a staple in a variety of applications, such as robotic control systems, navigation, target tracking, multisensor data fusion and computer vision[10].

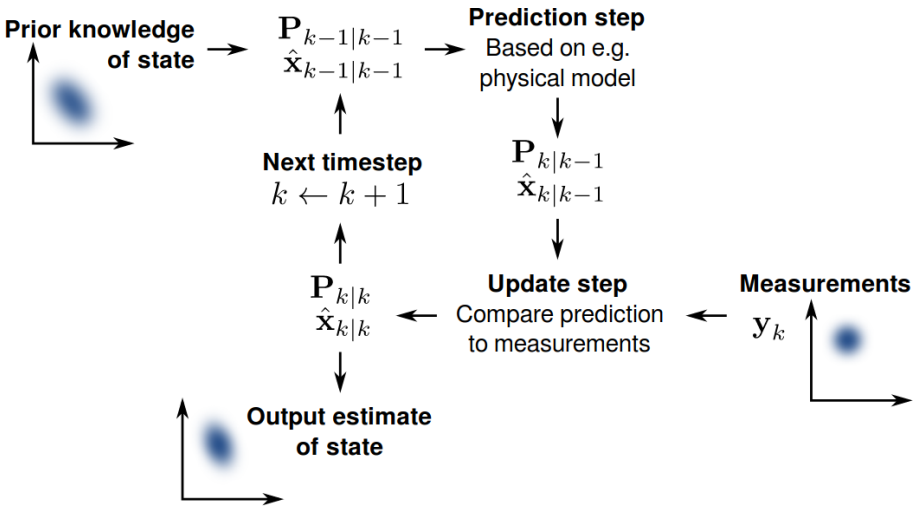



Figure 2.5: Overview of the Kalman filter algorithm (image published under public domain on Wikimedia Commons)

Grid-synchronization using Kalman filters has been performed successfully[11, 12]. Since grid-synchronization is a non-linear problem, variations of the Kalman filter that are more suited towards non-linear problems such as the extended Kalman filter (EKF) and the unscented Kalman filter (UKF) are typically employed [13, 14].

While exceptionally versatile, the Kalman filter has not seen much use in grid synchronization applications. This may be due to the computational complexity required, or simply just due to the lack of research for this specific application[6]. Nonetheless, given its widespread use in other areas, the appearance of more Kalman filter based grid-synchronization techniques is to be expected.

### 2.3.4 Neural Networks

Neural networks are sets of brain-inspired algorithms that loosely model how real neurons work[15]. These algorithms have been used in a huge variety of applications, such as image and video processing, image classification, object detection, speech recognition, machine translation, medical imaging, visual navigation, control systems, weather forecasting and much more [15]. Their massive flexibility stems from the huge interconnectedness of the network and the fact that each neuron employs some non-linear function to the sum of its inputs. This allows the neural network to effectively approximate extremely complex, hyper-dimensional non-linear functions.

Figure 2.6 presents a trivial example of a neural network that is two-layers deep[1]. The left-most dots represent input neurons, while the middle-most and right-most circles represent neurons. The neurons are connected by synapses, represented by the arrows,

[1]Some networks can be hundreds of layers deep

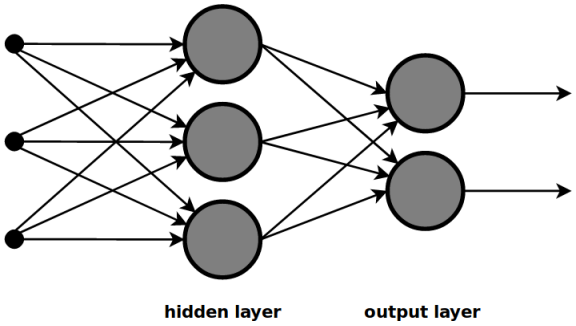



Figure 2.6: Example of a two-layer neural network (image published under public domain on Wikimedia Commons)

which are individually weighted. Each neuron sums the weighted inputs, applies a non-linear function - such as the sigmoid function - to it and passes its output on to the next layer assuming it exists, or its output is simply the output of the network if said neuron belongs to the output layer. It should be noted that not all neurons need to use the same non-linear function.

A neural network learns by adjusting the weights of the synapses. Many learning algorithms exist, but they all fundamentally focus on getting the neural network to behave properly for a set of training data, usually by attempting to maximize some "score" based on the performance of the network[15]. Broadly speaking, the bigger and more varied the data set is, the better.

The use of neural networks in power electronics is nothing new[16], and even outdated research by today's standards shows that neural networks can fiercely compete with alternative algorithms[6].

The main disadvantage of neural networks is the immense processing power required. Neural networks - much like human brains - are massively parallel, while typical processing units execute instructions in a sequential fashion and thus are not well suited for the task[15].

### 2.3.5 Phase Locked Loop

Among the different possible techniques for angle estimation, the phase-locked loop (PLL) is one of the most popular methods for synchronizing power electronics converters and has been used in a vast range of applications since its first appearance in the 1930s for usage in the reception of radio signals[6, 17, 18].

The PLL is a nonlinear negative-feedback system whose output has the same frequency and is in phase with the input. It consists of three main components: a phase detector, a loop filter and a controllable oscillator.

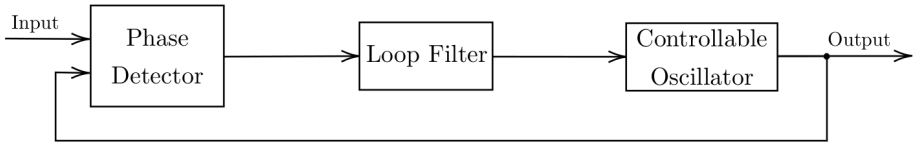



Figure 2.7: Standard architecture of a PLL

- Phase Detector
  - Compares the phase of the PLL's output and input signals, and produces an error signal which is proportional to the phase difference.
- Loop filter
  - Filters the output of the phase detector, as the phase detector might introduce unwanted components into the error signal. The filter output is fed to the controllable oscillator.
- Controllable oscillator
  - Produces a waveform of a given type (e.g. sine, triangle, sawtooth etc.) whose frequency depends on the oscillator's input. This waveform is considered to be the output of the PLL, and is then fed back into the phase detector, thus completing the loop.

In order to understand how a PLL works, consider a PLL operating in steady state, where the outputs of the phase detector and loop filter are constant, and the output of the controllable oscillator is some periodic waveform with a given frequency. If a small disturbance occurs such that the output's phase (in comparison with the input) is now lower, the output of the phase detector increases. This leads to an increase in the controllable oscillator's input, which in turn increases the frequency of the PLL's output signal, thus allowing it to "catch up" with the input signal. Conversely, if the output's phase increases, the phase detector reduces its output, thus "slowing down" the oscillator and allowing the input signal to "catch up" with the output signal[6].

While far from being perfect, the PLL technique allows for accurate grid angle tracking in comparison to the zero-crossing technique, while suffering from almost none of its flaws. In comparison with the more advanced techniques, its implementation is simple, straightforward and also has a relatively low computational burden allowing for implementation in low-cost microcontrollers, but the more advanced methods tend to be more robust and/or have faster system dynamics[6]. Hence, the PLL method strikes a good balance between the advantages and disadvantages of the available methods, and this may be the reason why it has seen widespread use.

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Chapter 3

# $\alpha\beta$ and DQ0 Transforms

## 3.1 Introduction

The $\alpha\beta$ and the direct-quadrature-zero (DQ0) transforms, alternatively known as the Clarke and Park transforms, are useful analysis tools that can be used to simplify three-phase voltages (and/or currents) into two-phase and DC signals, respectively. These transforms are usually employed in three-phase grid synchronization and converter control algorithms, thus a proper understanding of them is absolutely crucial.

Most literature tends to focus heavily on mathematical deduction to explain these transforms. This chapter takes a different approach by leveraging the reader's geometrical intuition in order to provide a clearer explanation of the inner workings of these transforms, and only relying on pure mathematical proofs where they are either trivial or where the alternatives would be more difficult to understand.

## 3.2 $\alpha\beta$ Transform

In an ideal and perfectly balanced three-phase system, the voltages $v_a(t)$, $v_b(t)$ and $v_c(t)$ are given by:

$$v_a(t) = V \cdot sin(\theta(t));\ v_b(t) = V \cdot sin\left(\theta(t) - \frac{2\pi}{3}\right);\ v_c(t) = V \cdot sin\left(\theta(t) + \frac{2\pi}{3}\right) \quad (3.1)$$

Where $V$ is the amplitude of the sine waves and $\theta(t)$ is the angle of the grid.

While most three-phase systems are usually close to a balanced system, this is only so because electrical equipment and/or regulations require it. Technically, the voltages need not be strictly interdependent. Therefore, one possible way to represent the voltages involves the usage of a three-dimensional, orthogonal coordinate system. This way, each voltage $v_a(t)$, $v_b(t)$ and $v_c(t)$ is plotted on its corresponding axis, which will be named A, B and C. In Figure 3.1 such a coordinate system is shown, with a cube of side length 2 centered at the origin to better portray depth.

Figure 3.1 makes use of what is known as a perspective projection, which functions the same way as a camera or a human eye[19]. This type of projection introduces distortion into the objects being viewed. Any objects that are closer to the "camera" are enlarged, while objects away from it are shrunk. This provides the illusion of depth, even though the image itself is just a two-dimensional entity[19].

A projection can be understood as if light rays, known here as projection lines, were to hit the object being viewed and then intersect a plane known as the image plane. As the name implies, this plane is where the image of the object is formed from the intersection with the projection lines. In a perspective projection, the projections lines all meet at a single point behind the image plane.

An alternative to the perspective projection is a parallel projection, which is actually a group of many different types of projections. In any parallel projection, the projection lines never meet, leading to parallel lines in the original object always remaining parallel in the resulting image and thus no distortion due to distance occurs[19].

Figure 3.2 depicts identical circles being projected onto different image planes, using the perspective and one type of parallel projections. Only a few projection lines are shown for clarity. Note how the circle suffers no distortion due to distance in the parallel projection.

The isometric projection is a type of parallel projection in which the image plane is placed in such a way that all three axes are equally distorted, resulting in them being equally spaced apart and having equal units[19]. Figure 3.3 shows the three-dimensional coordinate system from Figure 3.1 being projected on one possible isometric image plane. The result of the projection is shown in Figure 3.4.

Please note that it does not matter how far away from the origin the image plane is placed since no distortion due to distance occurs. In fact, the image plane from Figure 3.1 could even be placed at the origin as long as the plane is in the correct orientation.

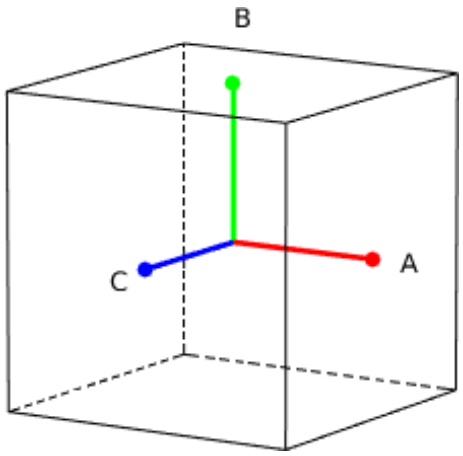



Figure 3.1: 3D coordinate system

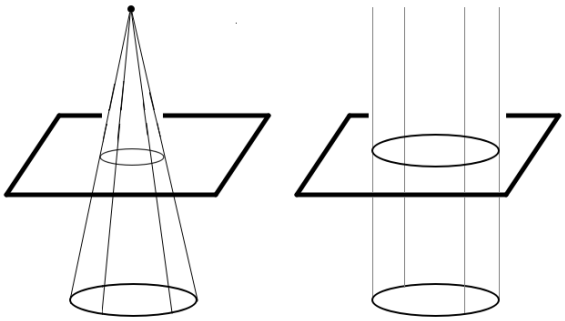


Figure 3.2: Perspective (left) and parallel (right) projections

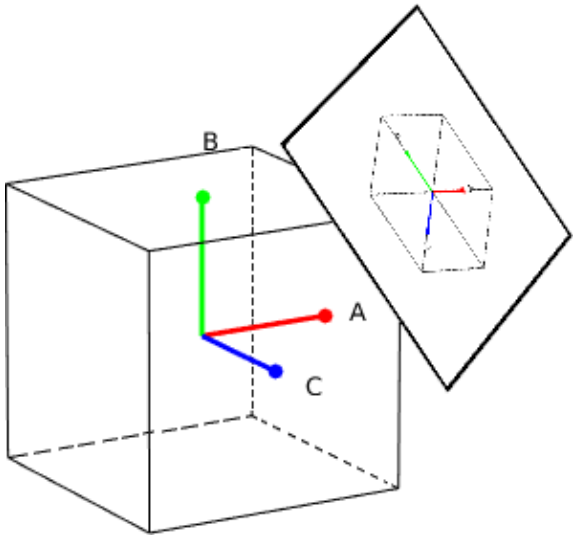



Figure 3.3: 3D coordinate system with isometric image plane represented

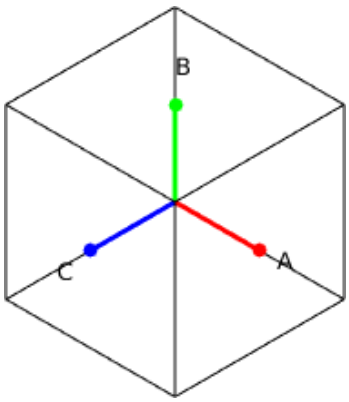



Figure 3.4: Isometric projection

If each voltage at a given time is considered to be a vector aligned with its respective axis, then the vectorial sum of $\vec{v_a}$, $\vec{v_b}$ and $\vec{v_c}$ results in what is known as a space vector $\vec{S} = (v_a(t), v_b(t), v_c(t))$[20], as shown in Figure 3.5.

In an ideal three-phase system, where the voltages are given exactly by equations 3.1, the following condition is true, as demonstrated in Appendix A:

$$v_a(t) + v_b(t) + v_c(t) = 0 \tag{3.2}$$

The general equation of a plane in any given XYZ coordinate system is given by:

$$A \cdot x + B \cdot y + C \cdot z + D = 0 \tag{3.3}$$

Where $\vec{n} = (A, B, C)$ is the normal vector of the plane and $D$ is the shortest distance from the plane to the origin[21].

Notice how similar equations 3.2 and 3.3 are. In fact, they are equal for $A = B = C = 1$ and $D = 0$. This means that in a balanced three-phase system, the space vector $\vec{S}$ will always be contained inside a plane, known as the zero plane, which is perpendicular to the vector $\vec{n} = (1, 1, 1)$ and also goes through the origin.

Interestingly, the isometric projection image plane is also perpendicular to vector $\vec{n} = (1, 1, 1)$, which makes it parallel to the zero plane, shown in Figure 3.6. This is due to the image plane being placed at an azimuth of 45 ° and at an elevation of $\arcsin(1/\sqrt{3}) \approx 35.26$ °[19], which when transformed from spherical coordinates to Cartesian coordinates equals exactly $(1, 1, 1)$ for unit distance from the origin. Due to this parallelism between the image and zero planes, the magnitude of the space vector $\vec{S}$ is preserved in the isometric projection during balanced grid conditions.

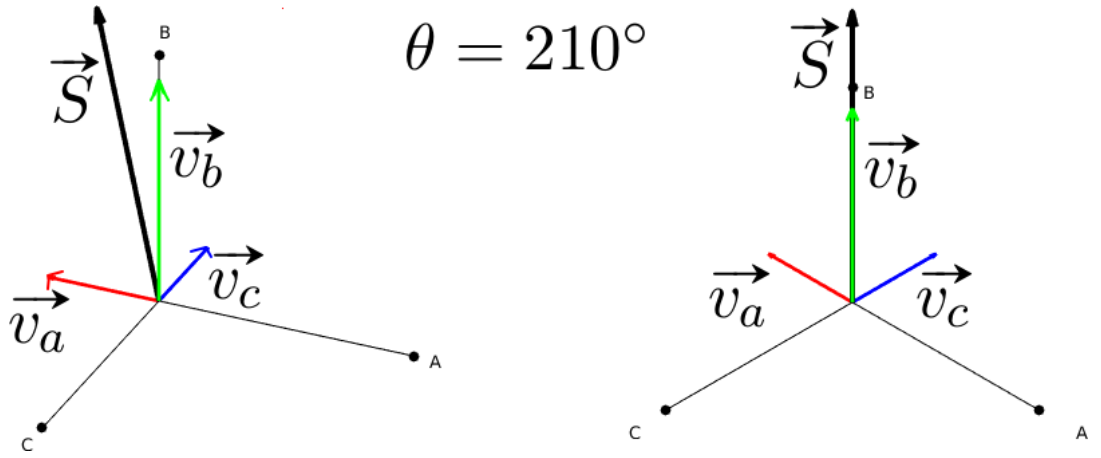



Figure 3.5: Space vector resulting from the vectorial sum: perspective (left) and isometric (right) projections

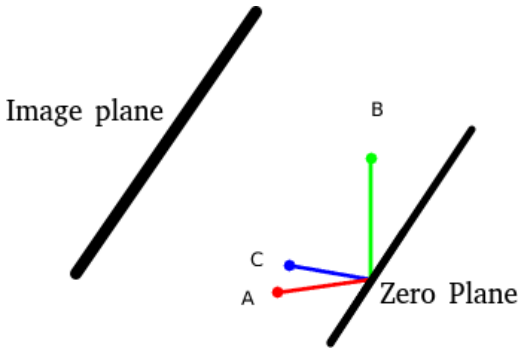



Figure 3.6: Parallelism between image plane and zero plane

The next step is to add a new reference frame on top of the image plane, called the $\alpha\beta$ reference frame, such that the $\alpha$ axis is coincident with the A axis, as shown in Figure 3.7. The goal of this is to find two vectors $\vec{v_\alpha}$ and $\vec{v_\beta}$, thus converting the three-phase system into a two-phase system.

Notice how the $\alpha$ axis is longer than the $A$ axis. This is because an isometric projection introduces a shrink factor of $\sqrt{2/3}$ on each axis[19]. This however poses no problem to our analysis, since we can simply scale the magnitudes of vectors $\vec{v_a}$, $\vec{v_b}$ and $\vec{v_c}$ accordingly.

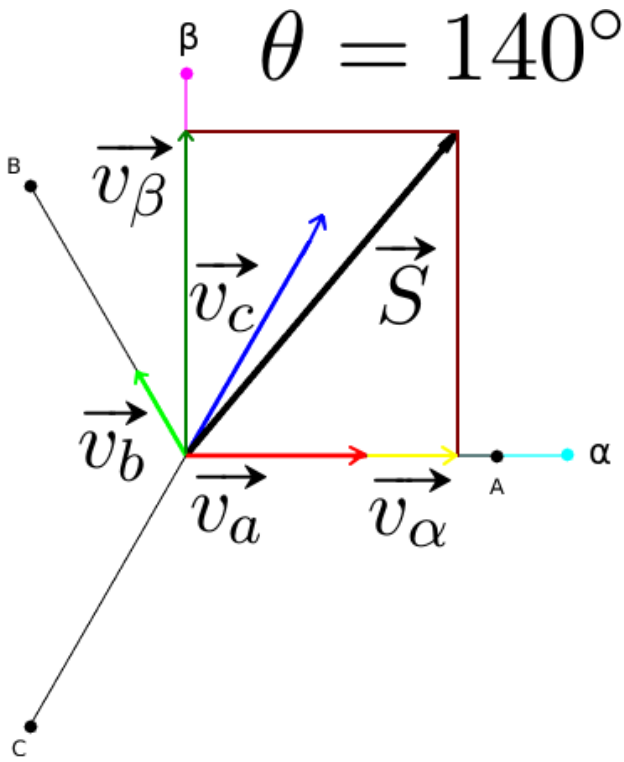



Figure 3.7: $\alpha\beta$ reference frame

It follows from basic geometry that $\overrightarrow{v_\alpha}$ and $\overrightarrow{v_\beta}$ are simply the sum of the projection of vectors $\overrightarrow{v_a}$, $\overrightarrow{v_b}$ and $\overrightarrow{v_c}$ onto the the $\alpha$ and $\beta$ axes, respectively. Knowing that the magnitudes of each voltage appear scaled by $\sqrt{2/3}$ on the isometric projection, one can easily extract $\overrightarrow{v_\alpha}$ and $\overrightarrow{v_\beta}$ using basic trigonometry:

$$\begin{cases} v_\alpha(t) &= \sqrt{\frac{2}{3}} \cdot (\cos(0\ ^\circ) \cdot v_a(t) + \cos(120\ ^\circ) \cdot v_b(t) + \cos(240\ ^\circ) \cdot v_c(t)) \\ v_\beta(t) &= \sqrt{\frac{2}{3}} \cdot (\sin(0\ ^\circ) \cdot v_a(t) + \sin(120\ ^\circ) \cdot v_b(t) + \sin(240\ ^\circ) \cdot v_c(t)) \end{cases} \tag{3.4}$$

Simplifying the above equations into a single matrix equation yields:

$$\begin{bmatrix} v_\alpha(t) \\ v_\beta(t) \end{bmatrix} = \sqrt{\frac{2}{3}} \cdot \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} v_a(t) \\ v_b(t) \\ v_c(t) \end{bmatrix} \tag{3.5}$$

Equation 3.5 is known as the power-invariant version of the $\alpha\beta$ transform, which allows for the correct calculation of active and reactive power using $v_\alpha(t)$ and $v_\beta(t)$.

In unbalanced systems, the space vector $\vec{S}$ is not contained within the zero plane since equation 3.2 is no longer valid, which leads to a distortion in its amplitude when projected to the image plane. This leads some authors to include a third component in the $\alpha\beta$ transform, known as $v_0(t)$[22]. Since this component represents how unbalanced the system is, it follows that $v_0(t)$ must be the sum of $v_a(t)$, $v_b(t)$ and $v_c(t)$, but this sum must be scaled correctly in order to preserve power. The full transform is given by[22]:

$$\begin{bmatrix} v_\alpha(t) \\ v_\beta(t) \\ v_0(t) \end{bmatrix} = \sqrt{\frac{2}{3}} \cdot \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} v_a(t) \\ v_b(t) \\ v_c(t) \end{bmatrix} \tag{3.6}$$

Let $i_\alpha(t)$ and $i_\beta(t)$ be the $\alpha\beta$ transformations of the three-phase currents in a given system. The equations for power are given by[22]:

$$\begin{cases} p_0(t) &= v_0(t) \cdot i_0(t) \\ p(t) &= v_\alpha(t) \cdot i_\alpha(t) + v_\beta(t) \cdot i_\beta(t) \\ q(t) &= v_\alpha(t) \cdot i_\beta(t) - v_\beta(t) \cdot i_\alpha(t) \end{cases} \tag{3.7}$$

Where $p(t)$ is instantaneous real power, $q(t)$ is instantaneous imaginary power, and $p_0(t)$ is instantaneous zero-sequence power.

While $p(t)$ matches exactly the definition of active power, the same cannot be said for $q(t)$ which has a completely different physical meaning from reactive power. However, under balanced conditions, $q(t)$ happens to take the exact same value as the reactive power[22]. The value of $p_0(t)$ is usually ignored due to being small (ideally zero) or considered to be a part of $p(t)$ since it also represents active power.

Sometimes it is beneficial to scale the $\alpha\beta$ transform such that the amplitude $V$ of $v_a(t)$, $v_b(t)$ and $v_c(t)$ matches the amplitudes of $v_\alpha(t)$ and $v_\beta(t)$. Looking at Figure 3.7, it is easy to see that in order for the amplitude of $v_\alpha(t)$ to match $V$, the space vector $\vec{S}$ must be scaled such that its magnitude (i.e. its norm) is $V$. This way, when $\vec{S}$ is aligned with the $\alpha$ axis, $v_\alpha(t)$ will have amplitude $V$. Since $\vec{S}$ has a constant magnitude[1] of $\sqrt{3/2} \cdot V$, the voltages $v_a(t)$, $v_b(t)$ and $v_c(t)$ must be scaled by $\sqrt{2/3}$ to force the space vector $\vec{S}$ to have a magnitude of $V$.

Applying this scaling factor to equation 3.6 results in:

$$\begin{bmatrix} v'_\alpha(t) \\ v'_\beta(t) \\ v'_0(t) \end{bmatrix} = \sqrt{\frac{2}{3}} \cdot \sqrt{\frac{2}{3}} \cdot \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} v_a(t) \\ v_b(t) \\ v_c(t) \end{bmatrix} \tag{3.8}$$

Where $v'_\alpha(t)$, $v'_\beta(t)$ and $v'_0(t)$ are the power-variant, amplitude-invariant versions of $v_\alpha(t)$, $v_\beta(t)$ and $v_0(t)$.

Simplifying equation 3.8 yields:

$$\begin{bmatrix} v'_\alpha(t) \\ v'_\beta(t) \\ v'_0(t) \end{bmatrix} = \frac{2}{3} \cdot \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} v_a(t) \\ v_b(t) \\ v_c(t) \end{bmatrix} \tag{3.9}$$

Notice in equation 3.8 how scaling voltages $v_a(t)$, $v_b(t)$ and $v_c(t)$ by $\sqrt{2/3}$ is actually equivalent to scaling $v_\alpha(t)$, $v_\beta(t)$ and $v_0(t)$ by $\sqrt{3/2}$. This means that power can still be easily calculated using the power-variant version of the transform by taking this scaling into account:

$$\begin{cases} p_0(t) &= \sqrt{\frac{3}{2}} \cdot v'_0(t) \cdot \sqrt{\frac{3}{2}} \cdot i'_0(t) \\ p(t) &= \sqrt{\frac{3}{2}} \cdot v'_\alpha(t) \cdot \sqrt{\frac{3}{2}} \cdot i'_\alpha(t) + \sqrt{\frac{3}{2}} \cdot v'_\beta(t) \cdot \sqrt{\frac{3}{2}} \cdot i'_\beta(t) \\ q(t) &= \sqrt{\frac{3}{2}} \cdot v'_\alpha(t) \cdot \sqrt{\frac{3}{2}} \cdot i'_\beta(t) - \sqrt{\frac{3}{2}} \cdot v'_\beta(t) \cdot \sqrt{\frac{3}{2}} \cdot i'_\alpha(t) \end{cases} \tag{3.10}$$

Simplifying the above equations:

$$\begin{cases} p_0(t) &= \frac{3}{2} \cdot v'_0(t) \cdot i'_0(t) \\ p(t) &= \frac{3}{2} \cdot \Big( v'_\alpha(t) \cdot i'_\alpha(t) + v'_\beta(t) \cdot i'_\beta(t) \Big) \\ q(t) &= \frac{3}{2} \cdot \Big( v'_\alpha(t) \cdot i'_\beta(t) - v'_\beta(t) \cdot i'_\alpha(t) \Big) \end{cases} \tag{3.11}$$

[1] Proof provided in Appendix B

## 3.3 DQ0 Transform

The DQ0 transform is very similar to the $\alpha\beta$ transform, but instead it simply rotates the whole $\alpha\beta$ coordinate system by a given angle $\theta_r(t)$[20]. In Figure 3.8, the resulting $v_d$ and $v_q$ components of the DQ0 transform are shown for a $\theta_r$ of 10 °, with the A, B and C axis omitted for clarity.

Since the DQ0 transform is just a simple rotation of the $\alpha\beta$ transform, equations 3.4 can be easily adapted to fit the DQ0 transform. Consider the following example: since the D axis is just a rotation of the $\alpha$ axis then a rotation by 10 ° means that the B axis will now be closer by 10 ° to the D axis when compared to the $\alpha$ axis. So, instead of using $\cos(120\ ^\circ)$ to compute the contribution of $v_b(t)$ to $v_\alpha(t)$, one should use $\cos(120\ ^\circ - 10\ ^\circ)$ to get the contribution of $v_b(t)$ to $v_d(t)$. Generalizing the logic for any angle $\theta_r$, for all A B and C axis and for both $v_d(t)$ and $v_q(t)$ components yields:

$$\begin{cases} v_d(t) = \sqrt{\frac{2}{3}} \cdot (\cos(-\theta_r(t)) \cdot v_a(t) + \cos(120\ ^\circ - \theta_r(t)) \cdot v_b(t) \\ \qquad\qquad + \cos(240\ ^\circ - \theta_r(t)) \cdot v_c(t)) \\ v_q(t) = \sqrt{\frac{2}{3}} \cdot (\sin(-\theta_r(t)) \cdot v_a(t) + \sin(120\ ^\circ - \theta_r(t)) \cdot v_b(t) \\ \qquad\qquad + \sin(240\ ^\circ - \theta_r(t)) \cdot v_c(t)) \end{cases} \tag{3.12}$$

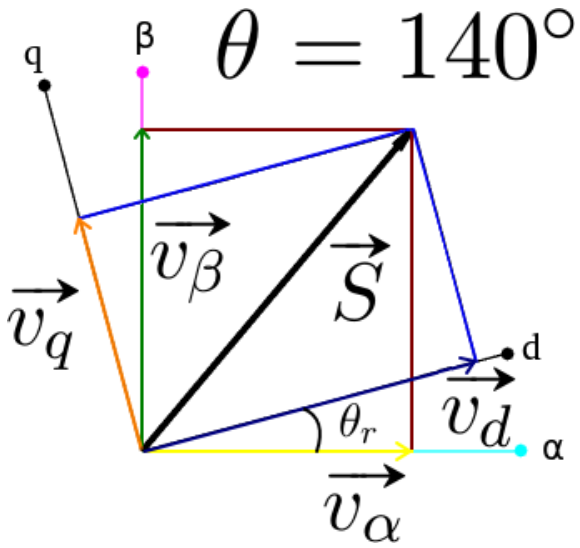



Figure 3.8: DQ0 reference frame at a $\theta_r$ of 10°

The zero component of the DQ0 transform follows the same logic as the $\alpha\beta$ transform. Hence, the power-invariant version of the DQ0 transform can also be written in matrix form:

$$\begin{bmatrix} v_d(t) \\ v_q(t) \\ v_0(t) \end{bmatrix} = \sqrt{\frac{2}{3}} \cdot \begin{bmatrix} \cos(\theta_r(t)) & \cos(\theta_r(t) - 120°) & \cos(\theta_r(t) + 120°) \\ -\sin(\theta_r(t)) & -\sin(\theta_r(t) - 120°) & -\sin(\theta_r(t) + 120°) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} v_a(t) \\ v_b(t) \\ v_c(t) \end{bmatrix} \quad (3.13)$$

The above equation 3.13 differs slightly from equations 3.12 due to the application of cosine and sine symmetries. In literature, the above equation may take other different, albeit equivalent forms due to the same reason.

For some specific applications it can be useful to perform the $\alpha\beta$ transform first, and then perform a counter-clockwise rotation to the reference system[23] to obtain $v_d(t)$ and $v_q(t)$:

$$\begin{bmatrix} v_d(t) \\ v_q(t) \end{bmatrix} = \begin{bmatrix} \cos(\theta_r(t)) & \sin(\theta_r(t)) \\ -\sin(\theta_r(t)) & \cos(\theta_r(t)) \end{bmatrix} \cdot \begin{bmatrix} v_\alpha(t) \\ v_\beta(t) \end{bmatrix} \quad (3.14)$$

Figure 3.8 represents precisely what equation 3.14 does, where the D axis is a simple rotation of the $\alpha$ axis.

The arguments made for the preservation of amplitudes also hold for the DQ0 transform, meaning that the power-variant, amplitude preserving versions can also be obtained by applying a scaling factor of $\sqrt{2/3}$ to equations 3.13 and 3.14.

If the angle $\theta_r(t)$ has the same angular frequency as the grid angle $\theta(t)$, the D and Q axes will "follow" the space vector as it rotates, which results in constant $v_d(t)$ and $v_q(t)$ values. If $\theta_r(t)$ not only has the same angular frequency but is also exactly equal to $\theta(t)$ (i.e. there is no phase difference), then the D axis will always be 90 ° ahead of $\vec{S}$, and $\vec{S}$ will be coincident with the negative semi-axis Q. This results in $v_q(t)$ taking the negative value of the magnitude of $\vec{S}$, and in $v_d(t)$ becoming zero.

Another variant of the DQ0 transform involves starting the D axis 90 ° behind the $\alpha$ axis (i.e. the Q axis becomes aligned with the $\alpha$ for $\theta_r(t) = 0$). This has the clear advantage that if $\theta_r(t) = \theta(t)$, then $v_d(t)$ takes the positive value of the magnitude of $\vec{S}$, while $v_q(t)$ becomes zero. To obtain this variant of the DQ0 transform, simply subtract 90 ° to $\theta_r(t)$ in equation 3.13. Doing this and simplifying results in:

$$\begin{bmatrix} v_d(t) \\ v_q(t) \\ v_0(t) \end{bmatrix} = \sqrt{\frac{2}{3}} \cdot \begin{bmatrix} \sin(\theta_r(t)) & \sin(\theta_r(t) - 120\ °) & \sin(\theta_r(t) + 120\ °) \\ \cos(\theta_r(t)) & \cos(\theta_r(t) - 120\ °) & \cos(\theta_r(t) + 120\ °) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} v_a(t) \\ v_b(t) \\ v_c(t) \end{bmatrix} \quad (3.15)$$

This variant of the DQ0 transform can also be obtained directly from the $\alpha\beta$ transform by applying the same 90 ° subtraction to equation 3.14 and further simplifying:

$$\begin{bmatrix} v_d(t) \\ v_q(t) \end{bmatrix} = \begin{bmatrix} \sin(\theta_r(t)) & -\cos(\theta_r(t)) \\ \cos(\theta_r(t)) & \sin(\theta_r(t)) \end{bmatrix} \cdot \begin{bmatrix} v_\alpha(t) \\ v_\beta(t) \end{bmatrix} \quad (3.16)$$

To reduce ambiguity when referring to the DQ0 transform, from this point on the transforms that convert directly from three-phase voltages to the DQ0 reference frame will be referenced as "ABC to DQ0" transforms, whereas the others will be referenced as "$\alpha\beta$ to DQ0" transforms. Additionally, transforms that align the D axis with the $\alpha$ axis at $\theta_r(t) = 0$ will be known as the DQ0-DA transforms, whereas transforms that align the Q axis will be known as DQ0-QA.

Using this new nomenclature, one can see that there are 8 possible versions of the DQ0 transform:

- ABC to DQ0-DA - Power invariant (eq. 3.13)
- ABC to DQ0-DA - Amplitude invariant (eq. 3.13 with $\sqrt{2/3}$ scaling factor)
- ABC to DQ0-QA - Power invariant (eq. 3.15)
- ABC to DQ0-QA - Amplitude invariant (eq. 3.15 with $\sqrt{2/3}$ scaling factor)
- $\alpha\beta$ to DQ0-DA - Power invariant (eq. 3.14 applied to eq. 3.6)
- $\alpha\beta$ to DQ0-DA - Amplitude invariant (eq. 3.14 applied to eq. 3.9)
- $\alpha\beta$ to DQ0-QA - Power invariant (eq. 3.16 applied to eq. 3.6)
- $\alpha\beta$ to DQ0-QA - Amplitude invariant (eq. 3.16 applied to eq. 3.9)

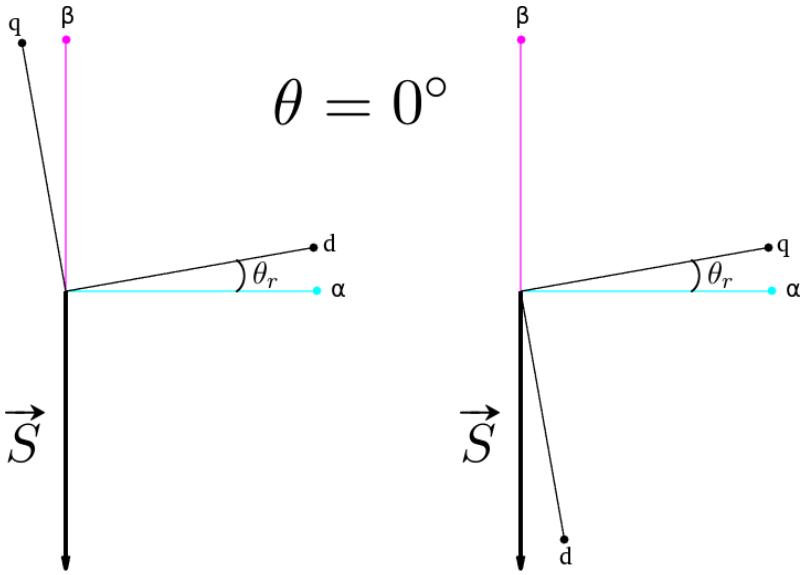



Figure 3.9: DQ0-DA (left) and DQ0-QA (right)

# Chapter 4

# Synchronous Reference Frame PLL

## 4.1 Introduction

The synchronous reference frame phase-locked loop (SRF-PLL) is a well established three-phase grid synchronization technique[4, 6, 18]. This chapter presents its fundamental building blocks and improves on the classical SRF-PLL. To limit the scope of this project, the simulations presented are purely demonstrative and should not be taken as strict proof of the concepts being demonstrated.

## 4.2 Components of the SRF-PLL

Let $v_a(t)$, $v_b(t)$ and $v_c(t)$ be ideal three-phase voltages as defined in equations 3.1. Let $\theta(t)$ be the angle of the grid and $\omega_g(t)$ its angular frequency[1], such that:

$$\theta(t) = \int \omega_g(t) \cdot dt \tag{4.1}$$

The goal of the SRF-PLL is to reconstruct the signal $\theta(t)$ from the three-phase voltages $v_a(t)$, $v_b(t)$ and $v_c(t)$. The reconstructed signal $\hat{\theta}(t)$ is the output of the PLL.

The error between the real grid angle $\theta(t)$ and the reconstructed angle $\hat{\theta}(t)$ is given by:

$$\phi(t) = \theta(t) - \hat{\theta}(t) \tag{4.2}$$

In an ideal PLL, the phase detector block would directly output $\phi(t)$. Since ideal phase detectors do not exist, the DQ0 transform can be used to produce a signal proportional to $\phi(t)$. Before trying to understand how, it is easier to first assume an ideal phase detector and work backwards from the output of the PLL, the controllable oscillator, which will be the block generating the signal $\hat{\theta}(t)$.

[1] Ideally $\omega_g(t)$ would not change over time, however it may vary slightly in practice

### 4.2.1 Controllable Oscillator

The controllable oscillator can be implemented as an integrator block which, when fed some frequency $\omega_i(t)$ equal to the grid's angular frequency $\omega_g(t)$, generates a signal equal to $\theta(t)$ with some unknown phase difference. Due to the cyclic nature of the sinusoidal function, the integrator can be reset every $2\pi$ radians to avoid unbounded variables, as $\hat{\theta}(t)$ will tend towards infinity. This results in a controllable oscillator which outputs a sawtooth wave at the specified input frequency.

### 4.2.2 Loop Filter

The loop filter must generate some input frequency $\omega_i(t)$ to feed the controllable oscillator based on the phase detector's output $\phi(t)$. Notice the importance of the sign of $\phi(t)$ to the loop filter: if $\phi(t)$ is positive, this means that $\hat{\theta}(t) < \theta(t)$. As was previously mentioned, to correct this error $\omega_i(t)$ must increase to allow $\hat{\theta}(t)$ to increase faster than $\theta(t)$, such that $\hat{\theta}(t)$ catches up with $\theta(t)$. If $\phi(t)$ is negative the reverse must happen. The properties of the loop filter now become clearer:

- Increase $\omega_i(t)$ if $\phi(t)$ is positive
- Decrease $\omega_i(t)$ if $\phi(t)$ is negative
- Hold the current value of $\omega_i(t)$ if $\phi(t)$ is zero

Technically, a simple integrator would suffice, but system dynamics may not be satisfactory. A more appropriate choice for a loop filter is a proportional-integral (PI) controller, which also incorporates the required integral action. With a non-ideal phase comparator, it might also be beneficial to add low-pass filtering before the PI controller to reduce unwanted frequency components that the phase comparator might produce[4, 18].

On a side note, under steady-state conditions (i.e. $\phi(t)$ is zero) the frequency $\omega_i(t)$ will match $\omega_g(t)$, meaning that the PLL also extracts the grid's frequency. This can be useful for advanced control strategies which rely on online parameter tuning to improve PLL performance[18].

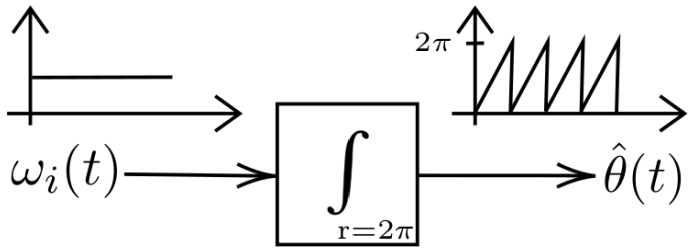



Figure 4.1: Controllable oscillator with integrated reset

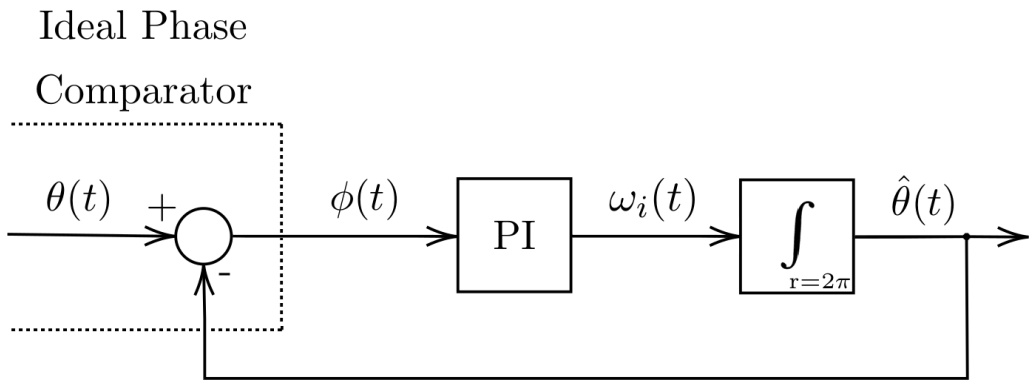



Figure 4.2: Idealized model of a PLL

Figure 4.2 represents an idealized model of the PLL, where the phase detector block outputs precisely $\phi(t)$. This model, with a few modifications to include noise and disturbance sources, could indeed be used to tune PI parameters[4]. Unfortunately, modeling the non-linear behaviour of a real phase detector as a disturbance source can be quite difficult. However, this simplistic model is quite useful to see exactly how a PLL can be understood as being a classical negative-feedback loop, where the loop filter (i.e. the PI controller) serves as a control system to the oscillator.

### 4.2.3 DQ0 Transform as a Phase Detector

The phase detector can be easily implemented using the ABC to DQ0-QA transform described in Chapter 3. Visually, it is easy to understand how the DQ0-QA transform can be used to detect phase: since the D axis starts aligned with $\vec{S}$ at $\theta(t) = \hat{\theta}(t) = 0$, then any slight phase difference between the real and estimated angles will cause a non-zero $v_q(t)$ signal, as demonstrated in Figure 4.3. From this, it is fairly obvious that $v_q(t)$ is proportional to $\phi(t)$ as long as the D axis is relatively close to $\vec{S}$. Additionally, $v_d(t)$ will also take the amplitude of the space vector when the D axis is aligned with it.

For this particular application the amplitude-invariant version is more convenient than the power-invariant version since using it allows for estimation of the grid's amplitude, which will prove useful later. As per Chapter 3, this transform is simply equation 3.15 with an additional scaling factor of $\sqrt{2/3}$:

$$\begin{bmatrix} v_d(t) \\ v_q(t) \end{bmatrix} = \sqrt{\frac{2}{3}} \cdot \sqrt{\frac{2}{3}} \cdot \begin{bmatrix} \sin(\theta_r(t)) & \sin(\theta_r(t) - 120\ ^\circ) & \sin(\theta_r(t) + 120\ ^\circ) \\ \cos(\theta_r(t)) & \cos(\theta_r(t) - 120\ ^\circ) & \cos(\theta_r(t) + 120\ ^\circ) \end{bmatrix} \cdot \begin{bmatrix} v_a(t) \\ v_b(t) \\ v_c(t) \end{bmatrix} \quad (4.3)$$

The zero component is omitted from the above equation since it is of no use to the SRF-PLL.

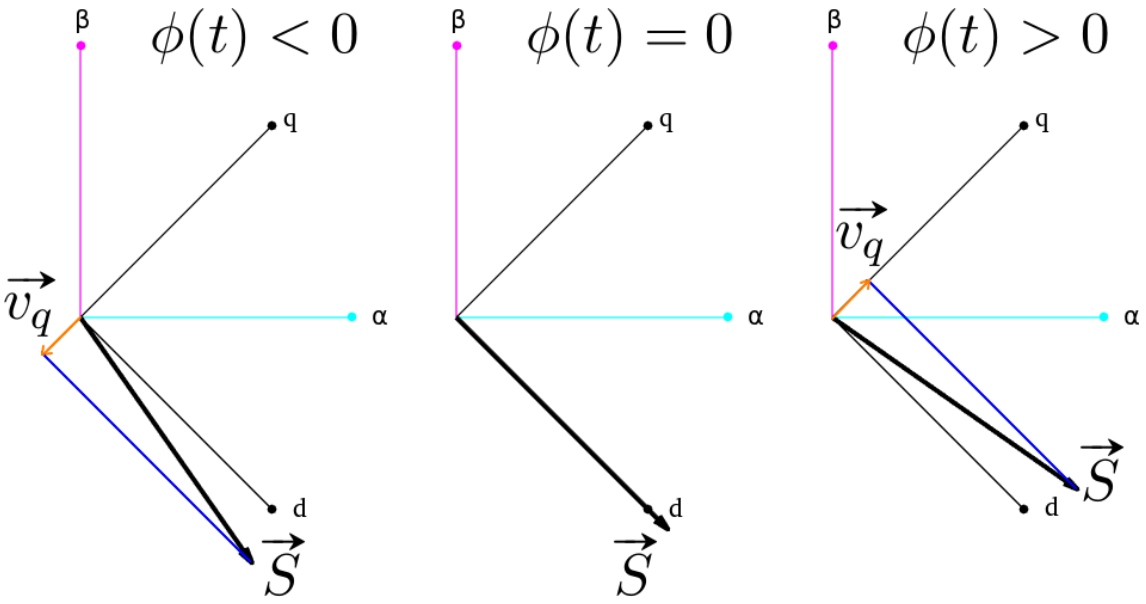



Figure 4.3: DQ0-QA transform as a phase detector

Now consider what happens when $\hat{\theta}(t)$ is used as the rotation angle $\theta_r(t)$. From equation 4.2 it follows that:

$$\hat{\theta}(t) = \theta(t) - \phi(t) \tag{4.4}$$

Applying the substitution 4.4 and the substitution 3.1 to equation 4.3 and only extracting the equation for $v_q(t)$ yields:

$$\begin{aligned} v_q(t) = \frac{2}{3}(&\cos(\theta(t) - \phi(t)) \cdot V \cdot \sin(\theta(t)) \\ &+ \cos(\theta(t) - \phi(t) - 120\ ^\circ) \cdot V \cdot \sin(\theta(t) - 120\ ^\circ) \\ &+ \cos(\theta(t) - \phi(t) + 120\ ^\circ) \cdot V \cdot \sin(\theta(t) + 120\ ^\circ)) \end{aligned} \tag{4.5}$$

Considering the following relation[2]: $\cos(a - b) \cdot \sin(a) = \frac{1}{2} \cdot (\sin(2 \cdot a - b) + \sin(b))$, then:

$$\begin{aligned} v_q(t) = \frac{2}{3} \cdot \frac{1}{2} \cdot V \cdot &(\sin(2 \cdot \theta(t) - \phi(t)) + \sin(\phi(t)) \\ &+ \sin(2 \cdot (\theta(t) - 120\ ^\circ) - \phi(t)) + \sin(\phi(t)) \\ &+ \sin(2 \cdot (\theta(t) + 120\ ^\circ) - \phi(t)) + \sin(\phi(t))) \end{aligned} \tag{4.6}$$

Further simplification leads to:

$$\begin{aligned} v_q(t) = V \cdot \sin(\phi(t)) + \frac{V}{3} \cdot &(\sin(2 \cdot \theta(t) - \phi(t)) + \sin(2 \cdot (\theta(t) - 120\ ^\circ) - \phi(t)) \\ &+ \sin(2 \cdot (\theta(t) + 120\ ^\circ) - \phi(t))) \end{aligned} \tag{4.7}$$

[2]Proof provided in Appendix C

Note how in equation 4.7, $v_q(t)$ takes the value of $V \cdot \sin(\phi(t))$ plus some additional double-frequency components.

Since the double-frequency components in equation 4.7 all have the same amplitude and are 120 ° apart from each other, then they will perfectly cancel out and result in the following equation:

$$v_q(t) = V \cdot \sin(\phi(t)) \tag{4.8}$$

Solving for $\phi(t)$:

$$\phi(t) = \arcsin\left(\frac{v_q(t)}{V}\right) \tag{4.9}$$

However, if $\phi(t)$ is sufficiently small, then it may be approximated as:

$$\phi(t) \approx \frac{v_q(t)}{V} \tag{4.10}$$

Hence, $v_q(t)$ can be used as the phase detector's output as shown in Figure 4.4.

The more astute reader may have noticed that equation 4.9 can also be derived by directly analyzing Figure 4.3 and realizing that $\vec{S}$, the D axis and $\vec{v}_q$ form a right triangle if $\vec{v}_q$ is shifted along the D axis to point at the tip of $\vec{S}$. But doing so skips over equation 4.7, which provides insight into the fundamental condition for equation 4.9 to be valid: the grid must be balanced. Otherwise, the double-frequency components do not perfectly cancel each other, leading to oscillations in $v_q(t)$.

Another interesting fact is that $v_q(t)$ is also zero when $\phi(t) = \pm 180$ °. This means that under ideal grid conditions and ideal starting conditions, the SRF-PLL would not be able to correct the phase difference. However, even the tiniest deviation from $\pm 180$ ° would cause the SRF-PLL to correct itself, since this point is effectively an unstable equilibrium point. For example, if $\phi(t)$ was 179 ° this would result in positive $v_q(t)$, thus the PI would increase the frequency. The DQ0 axis would then rotate anti-clockwise faster than the space vector such that $\phi(t)$ would tend to zero over time.

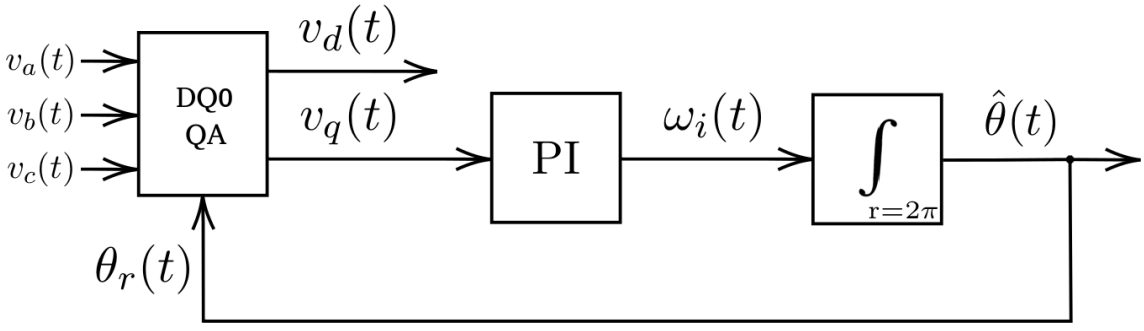



Figure 4.4: SRF-PLL

### 4.2.4 Performance of the SRF-PLL

The performance of the SRF-PLL is dependent on PI parameters, which is a trade-off between response time and disturbance rejection[4]. Figure 4.5 shows a Matlab/Simulink simulation of the performance of the SRF-PLL under ideal conditions with aggressive PI tuning parameters (i.e. high bandwidth) obtained by trial and error. As expected, the SRF-PLL starts by increasing the oscillator frequency beyond the actual grid frequency (50 Hz) to allow $\hat{\theta}(t)$ to "catch-up" with $\theta(t)$. The performance of the SRF-PLL is good under ideal conditions, however it degrades rapidly in the presence of noise, grid unbalance, harmonics and DC offset[6].

Figure 4.6 presents the results for an unbalanced grid and non-zero DC offsets. Even without noise and harmonics, there are high amplitude steady-state oscillations. Lowering the bandwidth mitigates this problem at the expense of slower system dynamics. From Figure 4.6 one can deduce that the signal $v_q(t)$ has multiple harmonics present, as the steady-state oscillations are clearly not pure sine waves. In the previous analysis of the DQ0 transform as a phase detector, equation 4.7 points to the non-perfect cancellation of the double-frequency components under an unbalanced grid, but never to the existence of components with other frequencies. Since an analysis of the effects of DC offset has not been carried up to this point, the simulation results point to the existence of harmonics generated by the DC offsets at a frequency other than double the grid's frequency. If this was not the case, the steady-state oscillations should have been close to pure sine waves.

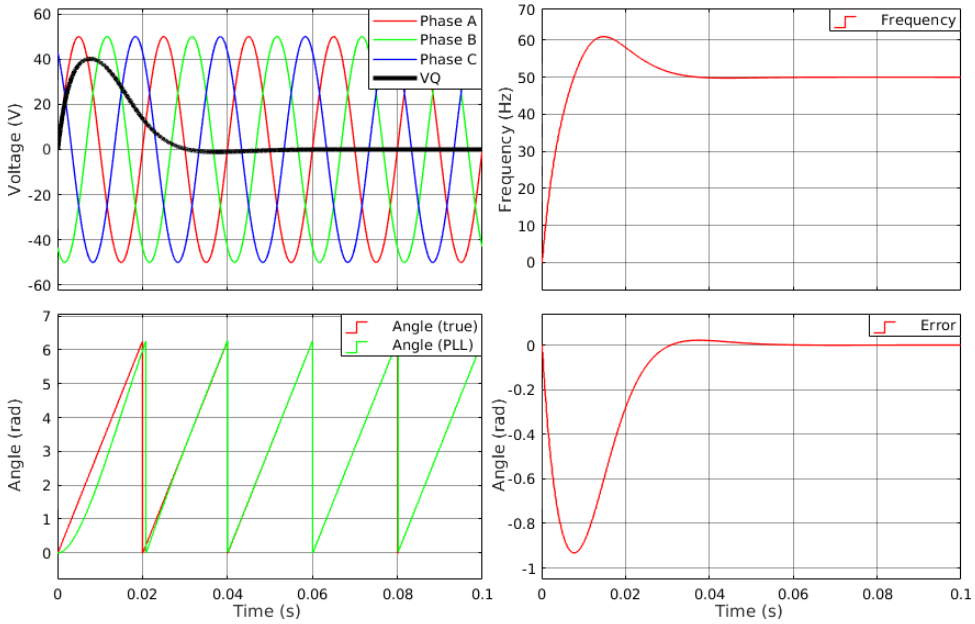



Figure 4.5: SRF-PLL under ideal conditions: amplitude = 50 V, proportional gain = 50, integral gain = 500

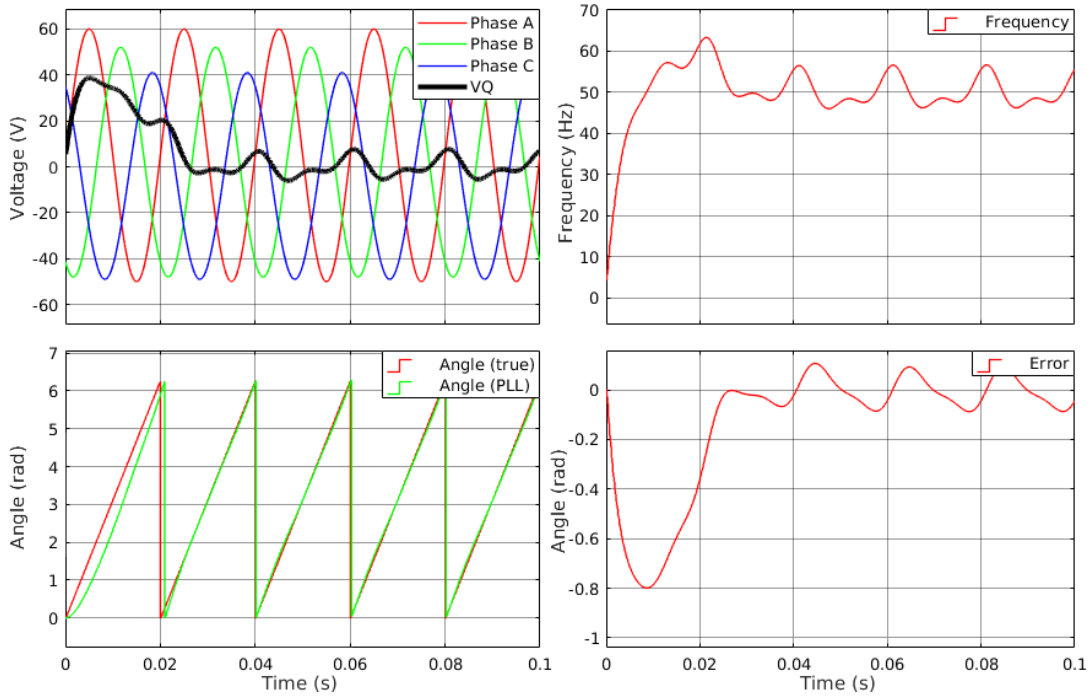



Figure 4.6: SRF-PLL with unbalanced grid and DC offset: amplitudes = [55 50 45] V and DC offsets = [5 2 -4] V for phases A, B and C, respectively. Maximum steady-state error is approximately 5.7 °

## 4.3 Simple Improvements to the SRF-PLL

Many variations of the SRF-PLL have been studied in order to improve its performance under non-ideal conditions[4, 6, 18]. A lot of improvements are relatively simple to add. However, while each improvement requires relatively low computational effort, the combination of many different improvements can easily become a significant burden. Hence, improving the SRF-PLL is also a trade-off between performance and computational effort. In this section, only the simplest improvements are studied.

### 4.3.1 Static Center Frequency

One of the easiest improvements to add to the SRF-PLL is to simply sum the expected grid frequency to the PI output. This reduces the control effort of the PI controller since it only needs to compensate for small deviations instead of actually estimating the frequency, thus mitigating the long start up required.

In the best case scenario, the real grid frequency matches the expected frequency and the PLL starts exactly when the grid angle is zero, thus requiring zero control effort from the PI controller.

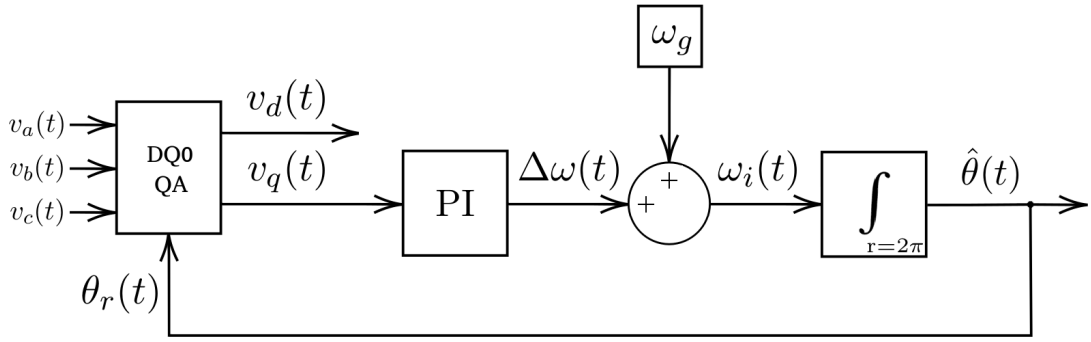



Figure 4.7: SRF-PLL with static center frequency

In a scenario where the PLL is enabled before the converter is connected to the grid, there are two possible cases. If the measured three-phase voltages are exactly zero, then $v_q(t)$ will also be zero, hence the PI controller does not change the input frequency to the oscillator. However, if the measured voltages are not zero (e.g. due to noise or sensor offset), then $v_q(t)$ will not be zero, and the PI controller will change the frequency. Assuming that the input frequency $\omega_i(t)$ is bounded, then the worst-case scenario would be the frequency drifting to the maximum or minimum value allowed while the converter is not connected, effectively nullifying the advantages of adding the initial center frequency.

### 4.3.2 Amplitude Normalization Scheme

As can be seen in equations 4.9 and 4.10, the amplitude $V$ appears as a gain in the forward path of the SRF-PLL, which means that it changes the dynamic characteristics of the SRF-PLL. An amplitude normalization scheme (ANS) can be used to eliminate this gain, and in doing so the performance of the PLL becomes independent of amplitude. Studies have shown that this also improves the filtering capabilities of the SRF-PLL[18].

The signal $v_d(t)$ takes the amplitude $V$ when the PLL is locked, hence it can be used for an ANS. However, this has the clear disadvantage that for big transients, the gain $V$ cannot be fully eliminated since the D axis will not be coincident with $\vec{S}$, thus $v_d(t) < V$. This indirectly increases the gains of the PI controller, therefore degrading noise immunity for the duration of the transient. This may increase the settling time or, in the worst-case scenario, lead to instability.

A better way to extract $V$ is to compute the norm of $\vec{S}$, either in the DQ0 reference frame or in the $\alpha\beta$ reference frame. This method is invariant to the current locking condition of the PLL, but it introduces costly square and square-root operations[3].

[3]Refer to equation B.1 in Appendix B

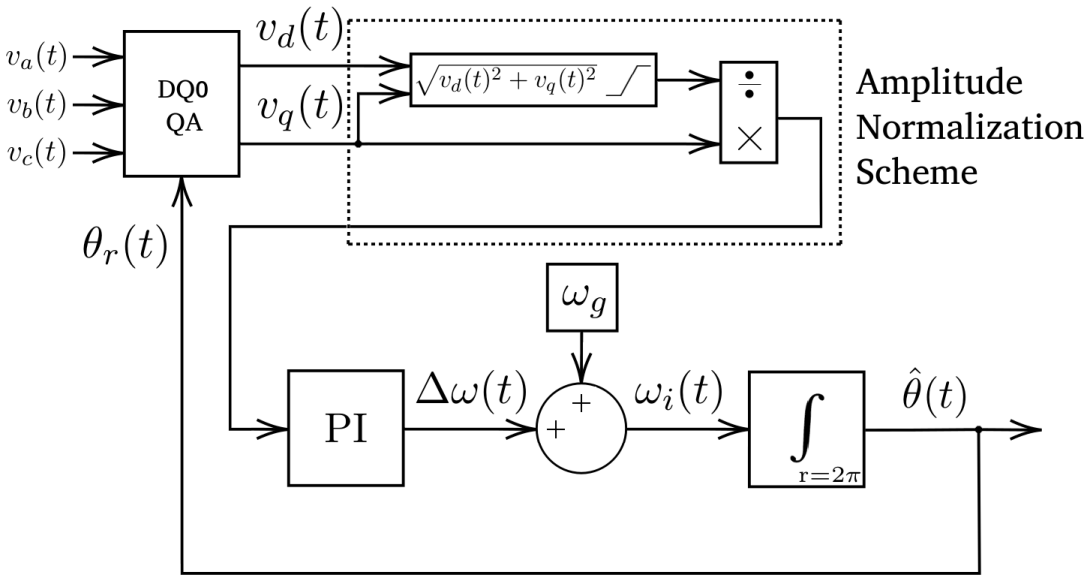



Figure 4.8: SRF-PLL with an ANS and static center frequency

In Figure 4.8 an amplitude normalization scheme using the output of the DQ0 transform is presented. Notice how the norm calculation includes saturation. Technically, only an inferior limit is necessary to avoid a division by zero.

In Figure 4.9, the performance of SRF-PLLs with and without an ANS are shown. The PI gains of the PLL with an ANS were multiplied by 50 to match the gains of the PLL without an ANS at $V = 50$ V, causing both to have the same response in this condition. A grid angle step of 45 ° is introduced at $t = 0.02$ s for the tests with $V = 50$ V and at $t = 0.021$ s for the other tests for better clarity. As can be seen, the ANS results in a PLL indifferent to amplitude. This is highly advantageous since differences between simulated and real conditions are likely, thus facilitating the selection of robust tuning parameters.

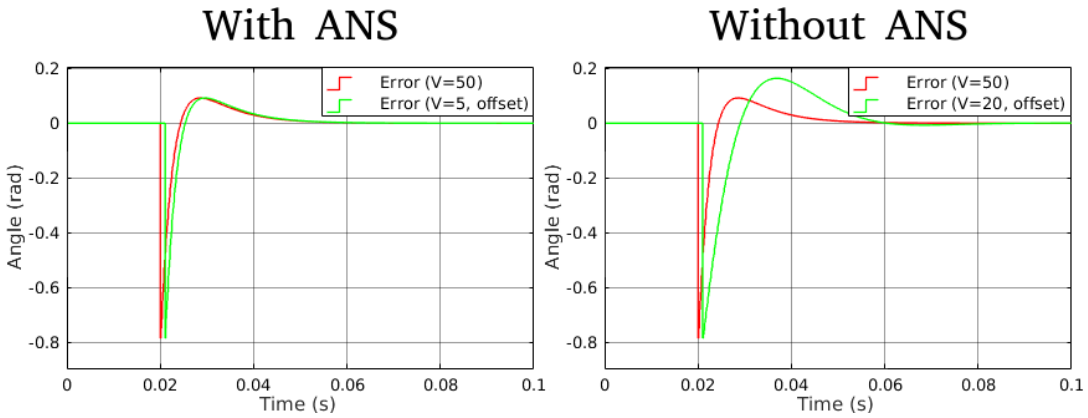



Figure 4.9: Comparison of performance with and without ANS for different amplitudes

### 4.3.3 Low-Pass Filtering

Introducing low-pass filters (LPFs) in different sections of the PLL can be useful to improve disturbance rejection. However, not unlike the PI controller gains, increasing the filtering capabilities of the PLL impacts negatively on its transient response[18, 24].

Placing LPFs directly on the three-phase voltages is not a good option since it would introduce phase shift to the signals, resulting in error in the estimated angle. Hence, in the simplest SRF-PLL architecture, it is common to filter $v_q(t)$[18]. However, if an ANS is used, there are a few different options:

1. Filter only the ANS output
2. Filter both $v_d(t)$ and $v_q(t)$, and use the filtered signals as the input to the ANS.
3. Compute the norm using $v_d(t)$ and $v_q(t)$ and filter the output of the norm, and then:
   (a) filter nothing else
   (b) filter the result of the division.
   (c) filter $v_q(t)$ before the division.

Option 1 has the lowest computational burden, but is also the less flexible option.

Option 2 is similar to option 1, but using separate filters for $v_d(t)$ and $v_q(t)$ allows for different cutoff frequencies. This may prove somewhat problematic for the norm calculation, since different cutoff frequencies imply different phase shifts.

Option 3(a) is a good choice for grid-connected systems, since the amplitude of the grid does not change significantly. In these conditions, by aggressively filtering the norm computation almost all noise and harmonics can be eliminated from the amplitude estimation, and since the amplitude of the grid is expected to be fairly constant, then any phase delay incurred by the filter is effectively insignificant. Thus, the disturbance rejection of the SRF-PLL with an ANS can be improved without negatively impacting performance since $v_q(t)$ is unfiltered, thus containing no phase delay.

Options 3(b) and 3(c) are very similar and may indeed be equivalent under certain conditions. For instance, in linear time-invariant (LTI) systems, the output $y$ of a system whose input is $x$ scales according to all scaling factors applied to $x$[25]. In other words, if a scaling factor of $k$ is applied to $x$ resulting in a input of $k \cdot x$, then the output becomes $k \cdot y$. It follows from this that if the norm calculation is aggressively filtered to remove all noise and harmonics to the point where it is practically constant and equal to $V$, then filtering either $v_q(t)$ or $v_q(t)/V$ is equivalent.

Option 3(c), shown in Figure 4.10, may be slightly faster if the hardware is capable of processing the norm and the filtering of $v_q(t)$ in parallel, and given that it is slightly more intuitive to tune since one filter does not affect the other, it is the preferred method in this project.

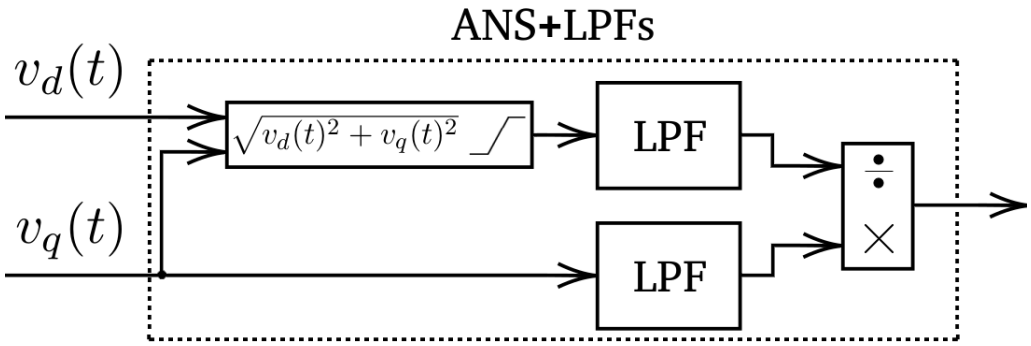



Figure 4.10: Chosen variant of the ANS with low-pass filtering

### 4.3.4 Frequency Feed-Forward

Instead of adding a static center frequency to the output of the PI controller as shown in Chapter 4.3.1, it is possible to dynamically estimate the current grid frequency[26]. This is done by computing the angle of the space vector $\vec{S}$ in the $\alpha\beta$ reference frame[4]:

$$\theta_{ff}(t) = \text{atan2}\left(v_\beta(t), v_\alpha(t)\right) \tag{4.11}$$

$$\omega_{ff}(t) = \frac{d\theta(t)_{ff}}{dt} \tag{4.12}$$

Under ideal conditions, $\theta_{ff}(t)$ is equal to $\theta(t)$ and $\omega_{ff}(t)$ is equal to $\omega_g(t)$. Hence, $\omega_{ff}(t)$ can be used as the center frequency for the SRF-PLL, thus resulting in a frequency feed-forward (FFF) mechanism as shown in Figure 4.12.

The most obvious advantage of this technique is that if the $\omega_{ff}(t)$ is always accurate and equal to $\omega_g(t)$, then the PLL becomes indifferent to frequency changes[26]. The second implication is that the SRF-PLL becomes more sensitive to phase jumps since the derivative will increase very significantly whenever a jump occurs.

The problem with this technique is that if harmonics are present on the grid, then the space vector $\vec{S}$ will no longer trace a perfectly circular trajectory in the $\alpha\beta$ plane[20], causing $\theta_{ff}(t)$ to not be equal to $\theta(t)$. This is shown in Figure 4.11 where a high amplitude and high frequency harmonic is present. This can be mitigated by a LPF.

A LPF additionally helps in reducing derivative spiking resulting from noise in the $v_\alpha(t)$ and $v_\beta(t)$ signals, but introduces some phase delay leading to a non-perfect cancellation of the effects of frequency changes. Therefore, the selection of the cutoff frequency of the filter is a trade-off between noise immunity and cancellation of the error introduced by frequency variations.

For simplicity, $\omega_{ff}(t)$ is now redefined as the output of the entire FFF mechanism.

[4]Equation 4.11 can be used directly to estimate the grid angle instead of relying on the SRF-PLL, but without any feedback loop it would need to rely heavily on low-pass filtering, thus yielding poor results due to phase shift

A comparison between a PLL with and without a FFF mechanism is shown in Figure 4.13, where a frequency step of 10 Hz is introduced at $t = 0.04$ s. Depending on the cutoff frequency chosen, the maximum error and the settling time both decrease, as is the case with the 50 Hz cutoff scenario. If the cutoff frequency is low, then the settling time may actually increase, as seen in the 10 Hz cutoff scenario. Even so, in both cases with the FFF present the maximum error is still reduced.

In the 10 Hz cutoff frequency scenario, it is clear from observing the controllable oscillator's input frequency and the FFF output after $t = 0.07$ s that the PI controller and the FFF mechanism are opposing each other, leading to a long settling time. This can possibly be improved by adjusting the PI gains, since they were kept constant in the scenarios shown in Figure 4.13 to provide a good comparison.

It should be noted that applying directly the output of the atan2 to the derivative will result in huge derivative spiking due to angle wrap-around. This problem has been solved for the simulations in Figure 4.13.

Since the SRF-PLL shown in Figure 4.12 contains all the improvements previously described, this improved version will be designated I-SRF-PLL from this point onward to avoid confusion with the classical SRF-PLL.

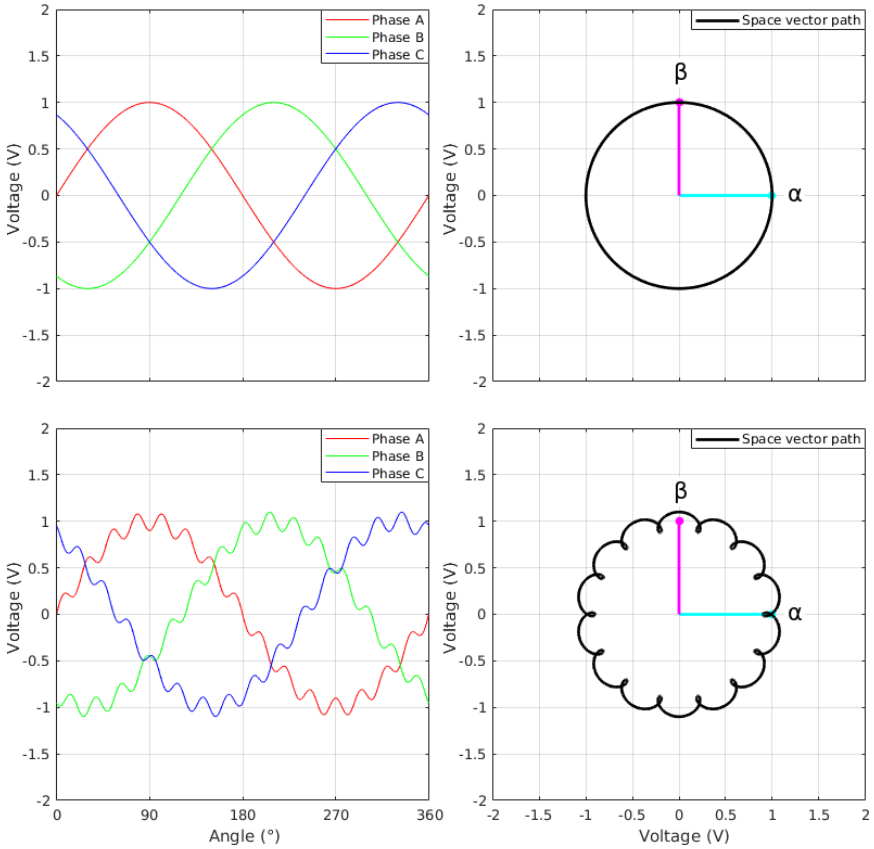



Figure 4.11: Space vector path in the presence of harmonics

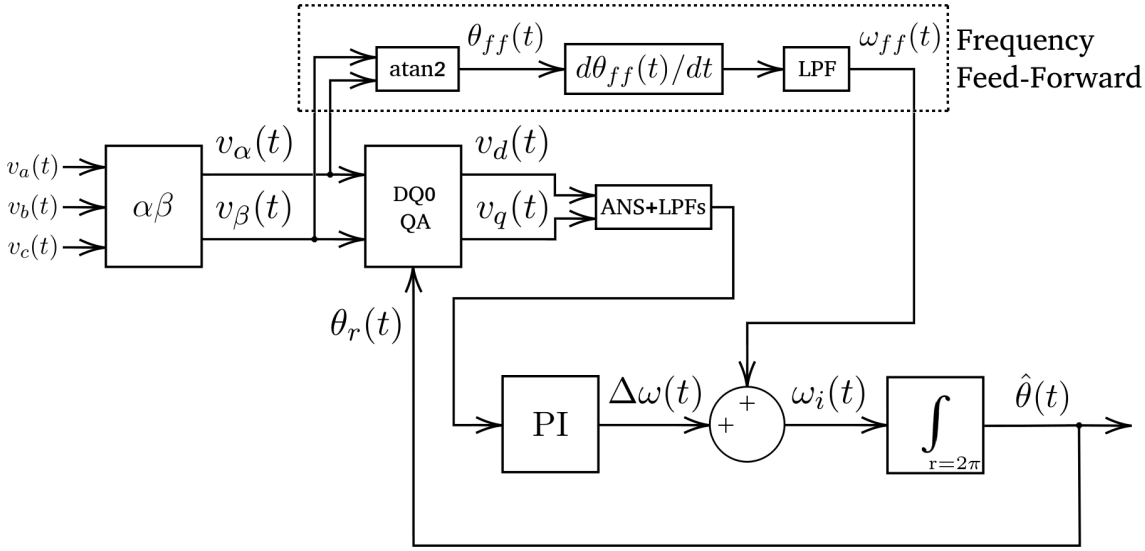



Figure 4.12: I-SRF-PLL

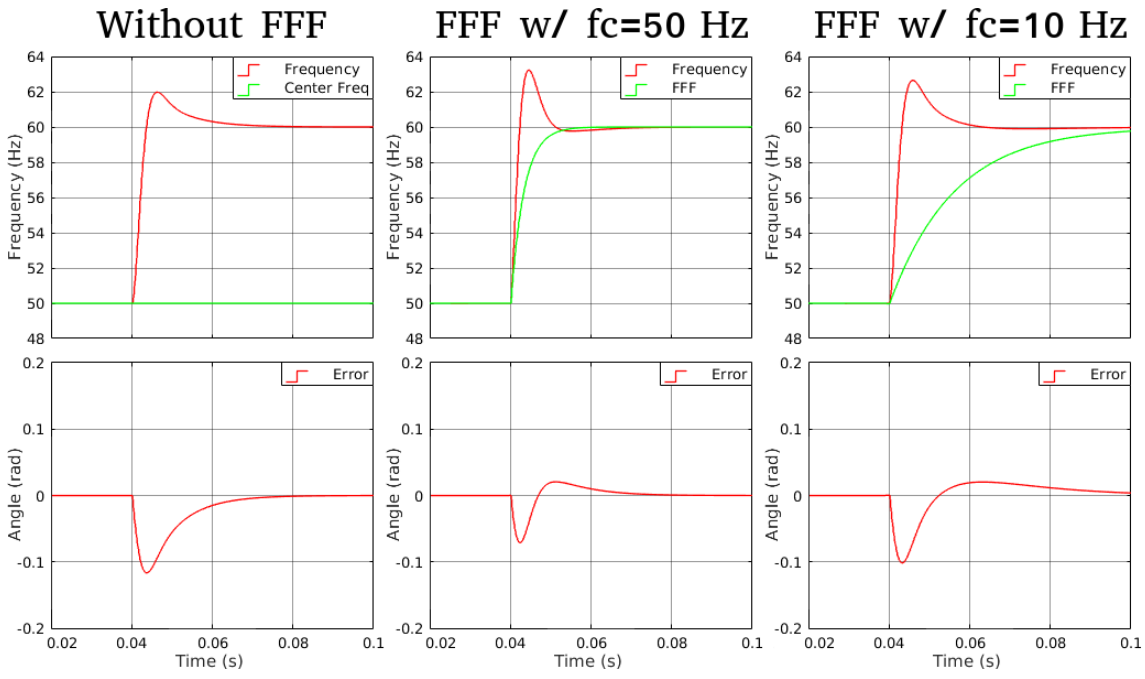



Figure 4.13: Comparison of a 10 Hz frequency step response without FFF (left) and with FFF whose output is filtered at cutoff frequencies (fc) of 50 Hz (middle) and 10 Hz (right)

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Chapter 5

# Extraction of Symmetrical Sequence Components

## 5.1 Introduction

One particular problem with the SRF-PLL (and with the I-SRF-PLL to a lesser extent) is that an unbalanced grid causes high-amplitude oscillations to appear in the outputs under steady-state conditions as per what was shown in Chapter 4, where the difference in performance under balanced and unbalanced conditions was demonstrated.

In order to understand how to further improve synchronization with the grid, it is easier to first study how unbalanced three-phase voltages can be mathematically transformed into positive, negative and zero sequence components, collectively known as symmetrical sequence components[27], and later on study how each one of these components affect the performance of the SRF-PLL.

## 5.2 Symmetrical Sequence Components

A positive sequence component is simply the set of voltages $[v_a^+(t)\ v_b^+(t)\ v_c^+(t)]^T$ which share the same amplitude $V^+$ and frequency, according to the following definition:

$$\begin{bmatrix} v_a^+(t) \\ v_b^+(t) \\ v_c^+(t) \end{bmatrix} = V^+ \cdot \begin{bmatrix} \sin\left(\theta(t) + \phi^+\right) \\ \sin\left(\theta(t) - \frac{2\pi}{3} + \phi^+\right) \\ \sin\left(\theta(t) + \frac{2\pi}{3} + \phi^+\right) \end{bmatrix} \tag{5.1}$$

The negative sequence component $[v_a^-(t)\ v_b^-(t)\ v_c^-(t)]^T$ is defined as:

$$\begin{bmatrix} v_a^-(t) \\ v_b^-(t) \\ v_c^-(t) \end{bmatrix} = V^- \cdot \begin{bmatrix} \sin\left(\theta(t) + \phi^-\right) \\ \sin\left(\theta(t) + \frac{2\pi}{3} + \phi^-\right) \\ \sin\left(\theta(t) - \frac{2\pi}{3} + \phi^-\right) \end{bmatrix} \tag{5.2}$$

Similarly, a zero sequence component $[v_a^0(t)\ v_b^0(t)\ v_c^0(t)]$ is defined as:

$$\begin{bmatrix} v_a^0(t) \\ v_b^0(t) \\ v_c^0(t) \end{bmatrix} = V^0 \cdot \begin{bmatrix} \sin\left(\theta(t) + \phi^0\right) \\ \sin\left(\theta(t) + \phi^0\right) \\ \sin\left(\theta(t) + \phi^0\right) \end{bmatrix} \tag{5.3}$$

Then, the three-phase voltages of a system can be expressed as a sum of all components:

$$\begin{bmatrix} v_a(t) \\ v_b(t) \\ v_c(t) \end{bmatrix} = \begin{bmatrix} v_a^+(t) + v_a^-(t) + v_a^0(t) \\ v_b^+(t) + v_b^-(t) + v_b^0(t) \\ v_c^+(t) + v_c^-(t) + v_c^0(t) \end{bmatrix} \tag{5.4}$$

In an unbalanced system, every single voltage may have different amplitudes and phase shifts. Assume said voltages can be represented as a sum of a positive, negative and zero sequence components. For the following deductions, it is easier to represent all sinusoidal voltages as the imaginary parts of the complex exponentials $[U_a(t)\ U_b(t)\ U_c(t)]$ as per Euler's formula[28],:

$$\begin{bmatrix} v_a(t) \\ v_b(t) \\ v_c(t) \end{bmatrix} = \begin{bmatrix} V_a \cdot \sin\left(\theta(t) + \phi_a\right) \\ V_b \cdot \sin\left(\theta(t) + \phi_b\right) \\ V_c \cdot \sin\left(\theta(t) + \phi_c\right) \end{bmatrix} = \mathrm{Im}\left( \begin{bmatrix} V_a \cdot e^{\phi_a j} e^{\theta(t) j} \\ V_b \cdot e^{\phi_b j} \cdot e^{\theta(t) j} \\ V_c \cdot e^{\phi_c j} \cdot e^{\theta(t) j} \end{bmatrix} \right) = \mathrm{Im}\left( \begin{bmatrix} U_a(t) \\ U_b(t) \\ U_c(t) \end{bmatrix} \right) \tag{5.5}$$

Let $\alpha = e^{\frac{2\pi}{3} j}$, which when multiplied by a complex exponential is equivalent to adding a phase shift of 120°. Therefore, $\alpha^2$ adds 240°, which is equivalent to subtracting 120°. Using the complex exponential representation of equation 5.4:

$$\begin{aligned} \mathrm{Im}\left( \begin{bmatrix} U_a(t) \\ U_b(t) \\ U_c(t) \end{bmatrix} \right) &= \mathrm{Im}\left( \begin{bmatrix} U_a^+(t) + U_a^-(t) + U_a^0(t) \\ U_b^+(t) + U_b^-(t) + U_b^0(t) \\ U_c^+(t) + U_c^-(t) + U_c^0(t) \end{bmatrix} \right) \\ &= \mathrm{Im}\left( \begin{bmatrix} U_a^+(t) \\ \alpha^2 \cdot U_a^+(t) \\ \alpha \cdot U_a^+(t) \end{bmatrix} + \begin{bmatrix} U_a^-(t) \\ \alpha \cdot U_a^-(t) \\ \alpha^2 \cdot U_a^-(t) \end{bmatrix} + \begin{bmatrix} U_a^0(t) \\ U_a^0(t) \\ U_a^0(t) \end{bmatrix} \right) \end{aligned} \tag{5.6}$$

Simplifying the above equation:

$$\mathrm{Im}\left( \begin{bmatrix} U_a(t) \\ U_b(t) \\ U_c(t) \end{bmatrix} \right) = \mathrm{Im}\left( \begin{bmatrix} 1 & 1 & 1 \\ 1 & \alpha^2 & \alpha \\ 1 & \alpha & \alpha^2 \end{bmatrix} \cdot \begin{bmatrix} U_a^0(t) \\ U_a^+(t) \\ U_a^-(t) \end{bmatrix} \right) \tag{5.7}$$

Taking the above equation and solving it for $[U_a^0(t)\ U_a^+(t)\ U_a^-(t)]$ yields:

$$\mathrm{Im}\left( \begin{bmatrix} U_a^0(t) \\ U_a^+(t) \\ U_a^-(t) \end{bmatrix} \right) = \mathrm{Im}\left( \frac{1}{3} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \\ 1 & \alpha^2 & \alpha \end{bmatrix} \cdot \begin{bmatrix} U_a(t) \\ U_b(t) \\ U_c(t) \end{bmatrix} \right) \tag{5.8}$$

### 5.2.1 Amplitudes and Phase Shifts of Symmetrical Components

Equation 5.8 can be usually found in literature[27]. However, little to no literature[1] has taken this equation and extract the amplitudes and phases of the positive, negative and zero sequence components, and demonstrate how $V^+$, $V^-$, $V^0$, $\phi^+$, $\phi^-$ and $\phi^0$ relate to the unbalanced grid voltages $V_a$, $V_b$ and $V_c$ and the corresponding phase shifts $\phi_a$, $\phi_b$ and $\phi_c$.

From equation 5.8, the equations for $U_a^0(t)$, $U_a^+$ and $U_a^-$ can be extracted and further simplified. Please note that the imaginary operator is omitted to improve readability:

$$\begin{cases} U_a^0(t) & = \frac{1}{3} \cdot (U_a(t) + U_b(t) + U_c(t)) \\ U_a^+(t) & = \frac{1}{3} \cdot \big(U_a(t) + \alpha \cdot U_b(t) + \alpha^2 \cdot U_c(t)\big) \\ U_a^-(t) & = \frac{1}{3} \cdot \big(U_a(t) + \alpha^2 \cdot U_b(t) + \alpha \cdot U_c(t)\big) \end{cases} \tag{5.9}$$

Expanding each voltage phasor and applying the $\alpha$ coefficients:

$$\begin{cases} U_a^0(t) & = \frac{1}{3} \cdot \big(V_a \cdot e^{\phi_a j} \cdot e^{\theta(t)j} + V_b \cdot e^{\phi_b j} \cdot e^{\theta(t)j} + V_c \cdot e^{\phi_c j} \cdot e^{\theta(t)j}\big) \\ U_a^+(t) & = \frac{1}{3} \cdot \Big(V_a \cdot e^{\phi_a j} \cdot e^{\theta(t)j} + V_b \cdot e^{\left(\phi_b + \frac{2\pi}{3}\right)j} \cdot e^{\theta(t)j} + V_c \cdot e^{\left(\phi_c - \frac{2\pi}{3}\right)j} \cdot e^{\theta(t)j}\Big) \\ U_a^-(t) & = \frac{1}{3} \cdot \Big(V_a \cdot e^{\phi_a j} \cdot e^{\theta(t)j} + V_b \cdot e^{\left(\phi_b - \frac{2\pi}{3}\right)j} \cdot e^{\theta(t)j} + V_c \cdot e^{\left(\phi_c + \frac{2\pi}{3}\right)j} \cdot e^{\theta(t)j}\Big) \end{cases} \tag{5.10}$$

Notice how the term $e^{\theta(t)j}$ is common to all bracketed terms in all equations, therefore it can be brought out:

$$\begin{cases} U_a^0(t) & = \frac{1}{3} \cdot \big(V_a \cdot e^{\phi_a j} + V_b \cdot e^{\phi_b j} + V_c \cdot e^{\phi_c j}\big) \cdot e^{\theta(t)j} \\ U_a^+(t) & = \frac{1}{3} \cdot \Big(V_a \cdot e^{\phi_a j} + V_b \cdot e^{\left(\phi_b + \frac{2\pi}{3}\right)j} + V_c \cdot e^{\left(\phi_c - \frac{2\pi}{3}\right)j}\Big) \cdot e^{\theta(t)j} \\ U_a^-(t) & = \frac{1}{3} \cdot \Big(V_a \cdot e^{\phi_a j} + V_b \cdot e^{\left(\phi_b - \frac{2\pi}{3}\right)j} + V_c \cdot e^{\left(\phi_c + \frac{2\pi}{3}\right)j}\Big) \cdot e^{\theta(t)j} \end{cases} \tag{5.11}$$

Let $N^0$, $N^+$ and $N^-$ be complex numbers inside the bracketed expressions in the above equations, such that:

$$\begin{cases} N^0 & = V_a \cdot e^{\phi_a j} + V_b \cdot e^{\phi_b j} + V_c \cdot e^{\phi_c j} \\ N^+ & = V_a \cdot e^{\phi_a j} + V_b \cdot e^{\left(\phi_b + \frac{2\pi}{3}\right)j} + V_c \cdot e^{\left(\phi_c - \frac{2\pi}{3}\right)j} \\ N^- & = V_a \cdot e^{\phi_a j} + V_b \cdot e^{\left(\phi_b - \frac{2\pi}{3}\right)j} + V_c \cdot e^{\left(\phi_c + \frac{2\pi}{3}\right)j} \end{cases} \tag{5.12}$$

[1] To the best of this project's author's knowledge, there is no literature that formally presents such equations, but there exists an online resource that visually demonstrates these relationships[29], thus it is possible that such literature already exists

Using Euler's formula[30], the complex exponentials in the complex numbers $N^0$, $N^+$ and $N^-$ can be separated into their real and imaginary parts. For compactness and improved readability, the notation $\mathrm{c}(x)$ and $\mathrm{s}(x)$ is temporarily used to represent $\cos(x)$ and $\sin(x)$:

$$\begin{cases} N^0 = V_a \cdot \mathrm{c}(\phi_a) + V_a \cdot \mathrm{s}(\phi_a) \cdot j + V_b \cdot \mathrm{c}(\phi_b) + V_b \cdot \mathrm{s}(\phi_b) \cdot j \\ \qquad\qquad + V_c \cdot \mathrm{c}(\phi_c) + V_c \cdot \mathrm{s}(\phi_c) \cdot j \\ N^+ = V_a \cdot \mathrm{c}(\phi_a) + V_a \cdot \mathrm{s}(\phi_a) \cdot j + V_b \cdot \mathrm{c}(\phi_b + \frac{2\pi}{3}) + V_b \cdot \mathrm{s}(\phi_b + \frac{2\pi}{3}) \cdot j \\ \qquad\qquad + V_c \cdot \mathrm{c}(\phi_c - \frac{2\pi}{3}) + V_c \cdot \mathrm{s}(\phi_c - \frac{2\pi}{3}) \cdot j \\ N^- = V_a \cdot \mathrm{c}(\phi_a) + V_a \cdot \mathrm{s}(\phi_a) \cdot j + V_b \cdot \mathrm{c}(\phi_b - \frac{2\pi}{3}) + V_b \cdot \mathrm{s}(\phi_b - \frac{2\pi}{3}) \cdot j \\ \qquad\qquad + V_c \cdot \mathrm{c}(\phi_c + \frac{2\pi}{3}) + V_c \cdot \mathrm{s}(\phi_c + \frac{2\pi}{3}) \cdot j \end{cases} \tag{5.13}$$

These complex numbers can now be written as:

$$\begin{cases} N^0 &= a^0 + b^0 \cdot j = \sqrt{(a^0)^2 + (b^0)^2} \cdot e^{\mathrm{atan}(b^0/a^0)j} \\ N^+ &= a^+ + b^+ \cdot j = \sqrt{(a^+)^2 + (b^+)^2} \cdot e^{\mathrm{atan}(b^+/a^+)j} \\ N^- &= a^- + b^- \cdot j = \sqrt{(a^-)^2 + (b^-)^2} \cdot e^{\mathrm{atan}(b^-/a^-)j} \end{cases} \tag{5.14}$$

Where:

$$\begin{cases} a^0 &= V_a \cdot \mathrm{c}(\phi_a) + V_b \cdot \mathrm{c}(\phi_b) + V_c \cdot \mathrm{c}(\phi_c) \\ b^0 &= V_a \cdot \mathrm{s}(\phi_a) + V_b \cdot \mathrm{s}(\phi_b) + V_c \cdot \mathrm{s}(\phi_c) \\ a^+ &= V_a \cdot \mathrm{c}(\phi_a) + V_b \cdot \mathrm{c}\left(\phi_b + \frac{2\pi}{3}\right) + V_c \cdot \mathrm{c}\left(\phi_c - \frac{2\pi}{3}\right) \\ b^+ &= V_a \cdot \mathrm{s}(\phi_a) + V_b \cdot \mathrm{s}\left(\phi_b + \frac{2\pi}{3}\right) + V_c \cdot \mathrm{s}\left(\phi_c - \frac{2\pi}{3}\right) \\ a^- &= V_a \cdot \mathrm{c}(\phi_a) + V_b \cdot \mathrm{c}\left(\phi_b - \frac{2\pi}{3}\right) + V_c \cdot \mathrm{c}\left(\phi_c + \frac{2\pi}{3}\right) \\ b^- &= V_a \cdot \mathrm{s}(\phi_a) + V_b \cdot \mathrm{s}\left(\phi_b - \frac{2\pi}{3}\right) + V_c \cdot \mathrm{s}\left(\phi_c + \frac{2\pi}{3}\right) \end{cases} \tag{5.15}$$

Substituting the complex numbers $N^0$, $N^+$ and $N^-$ from equation 5.14 in equation 5.11 leads to:

$$\begin{cases} U_a^0(t) &= \frac{\sqrt{(a^0)^2+(b^0)^2}}{3} \cdot e^{(\theta(t)+\mathrm{atan}(b^0/a^0))j} \\ U_a^+(t) &= \frac{\sqrt{(a^+)^2+(b^+)^2}}{3} \cdot e^{(\theta(t)+\mathrm{atan}(b^+/a^+))j} \\ U_a^-(t) &= \frac{\sqrt{(a^-)^2+(b^-)^2}}{3} \cdot e^{(\theta(t)+\mathrm{atan}(b^-/a^-))j} \end{cases} \tag{5.16}$$

Taking the imaginary parts of the previous equations:

$$\begin{cases} v_a^0(t) & = \frac{1}{3} \cdot \sqrt{(a^0)^2 + (b^0)^2} \cdot \sin\left(\theta(t) + \operatorname{atan}\left(\frac{b^0}{a^0}\right)\right) \\ v_a^+(t) & = \frac{1}{3} \cdot \sqrt{(a^+)^2 + (b^+)^2} \cdot \sin\left(\theta(t) + \operatorname{atan}\left(\frac{b^+}{a^+}\right)\right) \\ v_a^-(t) & = \frac{1}{3} \cdot \sqrt{(a^-)^2 + (b^-)^2} \cdot \sin\left(\theta(t) + \operatorname{atan}\left(\frac{b^-}{a^-}\right)\right) \end{cases} \tag{5.17}$$

By comparing the above equations with the original definitions of the zero, positive and negative sequence components, it becomes clear that:

$$\begin{cases} V^0 & = \frac{\sqrt{(a^0)^2+(b^0)^2}}{3} \\ \phi^0 & = \operatorname{atan}\left(\frac{b^0}{a^0}\right) \end{cases} \quad \begin{cases} V^+ & = \frac{\sqrt{(a^+)^2+(b^+)^2}}{3} \\ \phi^+ & = \operatorname{atan}\left(\frac{b^+}{a^+}\right) \end{cases} \quad \begin{cases} V^- & = \frac{\sqrt{(a^-)^2+(b^-)^2}}{3} \\ \phi^- & = \operatorname{atan}\left(\frac{b^-}{a^-}\right) \end{cases} \tag{5.18}$$

These equations can be useful either as models for usage in estimation algorithms such as the Kalman filter, or in simulated environments to confirm that the grid-synchronization algorithm is correctly estimating all parameters.

### 5.2.2 Special Case for Voltage Unbalance and Phase Balance

In the special case of an unbalanced grid where $\phi_a = 0$, $\phi_b = -120°$ and $\phi_c = 120°$, then by substitution on equations 5.15 and 5.18 the symmetrical components take the following values:

$$\begin{cases} V^0 & = \frac{1}{3} \cdot \sqrt{V_a^2 + V_b^2 + V_c^2 - (V_a \cdot V_b + V_a \cdot V_c + V_b \cdot V_c)} \\ \phi^0 & = -\operatorname{atan}\left(\frac{\sqrt{3} \cdot (V_b - V_c)}{2 \cdot V_a - V_b - V_c}\right) \\ V^+ & = \frac{1}{3} \cdot V_a + V_b + V_c \\ \phi^+ & = 0 \\ V^- & = \frac{1}{3} \cdot \sqrt{V_a^2 + V_b^2 + V_c^2 - (V_a \cdot V_b + V_a \cdot V_c + V_b \cdot V_c)} \\ \phi^- & = \operatorname{atan}\left(\frac{\sqrt{3} \cdot (V_b - V_c)}{2 \cdot V_a - V_b - V_c}\right) \end{cases} \tag{5.19}$$

Note how the zero and negative sequence components' amplitudes ($V^0$ and $V^-$) are equal and how their phase shifts ($\phi^0$ and $\phi^-$) only differ in sign. Additionally, the phase shift of the positive sequence component is zero, which implies that the voltages of the positive sequence component are always in phase with their corresponding grid voltages.

These novel equations could be useful to implement a power electronics converter that, by extracting information on the positive and negative sequence components, can actively cancel both the negative and zero sequence components.

Finally, note how in a balanced grid where $V_a = V_b = V_c$, only the positive sequence component remains, as $V^-$ and $V^0$ become zero. As expected, the negative and zero sequence components can only exist under unbalanced conditions.

## 5.3 Effects of Symmetrical Sequence Components and DC Offsets on SRF-PLL Performance

In order to comprehend exactly how grid unbalance impacts the performance of the classical SRF-PLL and the I-SRF-PLL, one can use the knowledge on the decomposition of three-phase voltages into their symmetrical sequence components to compute the result of the transforms and observe what type of oscillations emerge.

In this chapter, DC offsets were not considered to be a part of the three-phase voltages since they did not add any useful insight up to this point. While DC offsets technically should not occur in three-phase systems, signal acquisition hardware may introduce some unexpected bias. As such, studying the effects of DC offsets is crucial to understanding how any practical SRF-PLL might perform.

Consider the amplitude-invariant $\alpha\beta$ transform (equation 3.9). Using the symmetrical components plus some additional DC offsets as the input to this transform yields:

$$\begin{bmatrix} v_\alpha(t) \\ v_\beta(t) \\ v_0(t) \end{bmatrix} = \frac{2}{3} \cdot \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} v_a^+(t) + v_a^-(t) + v_a^0(t) + v_a^{DC} \\ v_b^+(t) + v_b^-(t) + v_b^0(t) + v_b^{DC} \\ v_c^+(t) + v_c^-(t) + v_c^0(t) + v_c^{DC} \end{bmatrix} \tag{5.20}$$

Extracting the equations for $v_\alpha(t)$ and $v_\beta(t)$:

$$\begin{cases} v_\alpha(t) = v_\alpha^+(t) + v_\alpha^-(t) + v_\alpha^0(t) + v_\alpha^{DC} \\ v_\beta(t) = v_\beta^+(t) + v_\beta^-(t) + v_\beta^0(t) + v_\beta^{DC} \end{cases} \tag{5.21}$$

Where:

$$\begin{cases} v_\alpha^+(t) &= \frac{2}{3} \cdot \left( v_a^+(t) - \frac{v_b^+(t)}{2} - \frac{v_c^+(t)}{2} \right) \\ v_\alpha^-(t) &= \frac{2}{3} \cdot \left( v_a^-(t) - \frac{v_b^-(t)}{2} - \frac{v_c^-(t)}{2} \right) \\ v_\alpha^0(t) &= \frac{2}{3} \cdot \left( v_a^0(t) - \frac{v_b^0(t)}{2} - \frac{v_c^0(t)}{2} \right) = \frac{2}{3} \cdot \left( v_a^0(t) - \frac{v_a^0(t)}{2} - \frac{v_a^0(t)}{2} \right) = 0 \\ v_\alpha^{DC} &= \frac{2}{3} \cdot \left( v_a^{DC} - \frac{v_b^{DC}}{2} - \frac{v_c^{DC}}{2} \right) \\ v_\beta^+(t) &= \frac{2}{3} \cdot \left( \frac{\sqrt{3}}{2} \cdot v_b^+(t) - \frac{\sqrt{3}}{2} \cdot v_c^+(t) \right) \\ v_\beta^-(t) &= \frac{2}{3} \cdot \left( \frac{\sqrt{3}}{2} \cdot v_b^-(t) - \frac{\sqrt{3}}{2} \cdot v_c^-(t) \right) \\ v_\beta^0(t) &= \frac{2}{3} \cdot \left( \frac{\sqrt{3}}{2} \cdot v_b^0(t) - \frac{\sqrt{3}}{2} \cdot v_c^0(t) \right) = \frac{2}{3} \cdot \left( \frac{\sqrt{3}}{2} \cdot v_b^0(t) - \frac{\sqrt{3}}{2} \cdot v_b^0(t) \right) = 0 \\ v_\beta^{DC} &= \frac{2}{3} \cdot \left( \frac{\sqrt{3}}{2} \cdot v_b^{DC} - \frac{\sqrt{3}}{2} \cdot v_c^{DC} \right) \end{cases} \tag{5.22}$$

It is clear from the previous equations that the $\alpha\beta$ transform happens to cancel the zero sequence components by itself, leaving only the positive and negative sequence components.

Further simplifications can be made to the previous equations, but since these are somewhat extensive and do not provide useful insight, they were deferred to Appendix D. The results are shown in equations 5.23.

$$\begin{cases} v_\alpha^+(t) &= V^+ \cdot \sin\left(\theta(t) + \phi^+\right) \\ v_\alpha^-(t) &= V^- \cdot \sin\left(\theta(t) + \phi^-\right) \\ v_\alpha^0(t) &= 0 \\ v_\alpha^{DC} &= \frac{2}{3} \cdot v_a^{DC} - \frac{v_b^{DC}}{3} - \frac{v_c^{DC}}{3} \end{cases} \qquad \begin{cases} v_\beta^+(t) &= V^+ \cdot \sin\left(\theta(t) + \phi^+ - \frac{\pi}{2}\right) \\ v_\beta^-(t) &= -V^- \cdot \sin\left(\theta(t) + \phi^- - \frac{\pi}{2}\right) \\ v_\beta^0(t) &= 0 \\ v_\beta^{DC} &= \frac{\sqrt{3}}{3} \cdot \left(v_b^{DC} - v_c^{DC}\right) \end{cases} \tag{5.23}$$

Assume a situation where some SRF-PLL is estimating an angle $\hat{\theta}(t)$ which is equal to the grid angle $\theta(t)$ plus some additional phase shift $\phi^R$. By applying the DQ0-QA transform (eq. 3.16) to the output of the $\alpha\beta$ transform:

$$\begin{cases} v_d(t) &= \sin\left(\theta(t) + \phi^R\right) \cdot \left(v_\alpha^+(t) + v_\alpha^-(t) + v_\alpha^{DC}\right) - \cos\left(\theta(t) + \phi^R\right) \cdot \left(v_\beta^+(t) + v_\beta^-(t) + v_\beta^{DC}\right) \\ v_q(t) &= \cos\left(\theta(t) + \phi^R\right) \cdot \left(v_\alpha^+(t) + v_\alpha^-(t) + v_\alpha^{DC}\right) + \sin\left(\theta(t) + \phi^R\right) \cdot \left(v_\beta^+(t) + v_\beta^-(t) + v_\beta^{DC}\right) \end{cases} \tag{5.24}$$

When simplified, the above equations result in[2]:

$$\begin{cases} v_d(t) = V^+ \cdot \cos\left(\phi^+ - \phi^R\right) - V^- \cdot \cos\left(2 \cdot \theta(t) + \phi^R + \phi^-\right) + v_\alpha^{DC} \cdot \sin\left(\theta(t) + \phi^R\right) \\ \qquad\qquad - v_\beta^{DC} \cdot \cos\left(\theta(t) + \phi^R\right) \\ v_q(t) = V^+ \cdot \sin\left(\phi^+ - \phi^R\right) + V^- \cdot \sin\left(2 \cdot \theta(t) + \phi^R + \phi^-\right) + v_\alpha^{DC} \cdot \cos\left(\theta(t) + \phi^R\right) \\ \qquad\qquad + v_\beta^{DC} \cdot \sin\left(\theta(t) + \phi^R\right) \end{cases} \tag{5.25}$$

It is extremely clear from the previous expressions that the existence of a negative sequence component causes double-frequency oscillations to appear, while DC offsets cause oscillations at the grid's frequency to appear. This fully explains the high amplitude steady-state oscillations previously seen in Figure 4.6, which appeared to be composed of harmonics of varying frequencies.

Analyzing carefully the previous equation for $v_q(t)$, one can also see that the positive sequence component results in an offset of $V^+ \cdot \sin\left(\phi^+ - \phi^R\right)$. As previously studied in Chapter 4.2.3, under steady-state conditions, $v_q(t)$ will contain no DC offset due to the PLL's integral action in the PI controller.

[2]Proof provided in Appendix E

In other words, $\phi^R$ becomes equal to $\phi^+$ to satisfy the condition $V^+ \cdot \sin(\phi^+ - \phi^R) = 0$, provided the oscillations from the negative sequence component and DC offsets are filtered out. This has the following implications:

- Under phase balance, where $\phi^+ = 0$, the PLL will estimate the grid angle $\theta(t)$ since $\phi^R = \phi^+ = 0$, provided sufficient filtering exists to remove unwanted oscillations.

- Under phase unbalance, where $\phi^+ \neq 0$, the PLL will output the angle of the positive sequence component $\theta(t) + \phi^+$ since $\phi^R = \phi^+$, provided sufficient filtering exists to remove unwanted oscillations.

## 5.4 Component Extraction from Filtered Signals

In the previous analysis of symmetrical components, the $\alpha\beta$ transform gave rise to some components and DC offsets which directly depend on the positive sequence component, negative sequence component and DC offsets of the three-phase voltages. Directly applying the DQ0-QA transform results in undesirable steady-state oscillations. However, it is possible to exploit the relationships between $v_\alpha^+(t)$, $v_\alpha^-(t)$, $v_\beta^+(t)$ and $v_\beta^-(t)$ to isolate these components.

Assume that some kind of filter removes the DC offsets $v_\alpha^{DC}$ and $v_\beta^{DC}$ from $v_\alpha(t)$ and $v_\beta(t)$. These filtered versions are now designated $v_{\alpha f}(t)$ and $v_{\beta f}(t)$. Then, assume that some other filter generates a phase shifted version designated $dv_{\alpha f}(t)$ and $dv_{\beta f}(t)$. The phase shift is selected to be exactly $-90°$, such that:

$$\begin{cases} v_{\alpha f}(t) &= v_\alpha^+(t) + v_\alpha^-(t) = V^+ \cdot \sin(\theta(t) + \phi^+) + V^- \cdot \sin(\theta(t) + \phi^-) \\ dv_{\alpha f}(t) &= dv_\alpha^+(t) + dv_\alpha^-(t) = V^+ \cdot \sin(\theta(t) + \phi^+ - \frac{\pi}{2}) + V^- \cdot \sin(\theta(t) + \phi^- - \frac{\pi}{2}) \\ v_{\beta f}(t) &= v_\beta^+(t) + v_\beta^-(t) = V^+ \cdot \sin\left(\theta(t) + \phi^+ - \frac{\pi}{2}\right) - V^- \cdot \sin\left(\theta(t) + \phi^- - \frac{\pi}{2}\right) \\ dv_{\beta f}(t) &= dv_\beta^+(t) + dv_\beta^-(t) = V^+ \cdot \sin\left(\theta(t) + \phi^+ - \frac{\pi}{2} - \frac{\pi}{2}\right) - V^- \cdot \sin\left(\theta(t) + \phi^- - \frac{\pi}{2} - \frac{\pi}{2}\right) \end{cases} \tag{5.26}$$

In order to obtain the positive and negative symmetrical sequence components, consider the outcomes of summing or subtracting $v_{\alpha f}(t)$ and $dv_{\beta f}(t)$, as well as the outcomes of summing or subtracting $v_{\beta f}(t)$ and $dv_{\alpha f}(t)$, as shown in equations 5.27.

$$\begin{cases} v_{\alpha f}(t) \pm dv_{\beta f}(t) = V^+ \cdot \sin\left(\theta(t) + \phi^+\right) + V^- \cdot \sin\left(\theta(t) + \phi^-\right) \\ \qquad\qquad\qquad \pm \left(V^+ \cdot \sin\left(\theta(t) + \phi^+ - \pi\right) - V^- \cdot \sin\left(\theta(t) + \phi^- - \pi\right)\right) \\ v_{\beta f}(t) \pm dv_{\alpha f}(t) = V^+ \cdot \sin\left(\theta(t) + \phi^+ - \dfrac{\pi}{2}\right) - V^- \cdot \sin\left(\theta(t) + \phi^- - \dfrac{\pi}{2}\right) \\ \qquad\qquad\qquad \pm \left(V^+ \cdot \sin\left(\theta(t) + \phi^+ - \dfrac{\pi}{2}\right) + V^- \cdot \sin\left(\theta(t) + \phi^- - \dfrac{\pi}{2}\right)\right) \end{cases} \tag{5.27}$$

By applying sine angle symmetry to the $v_{\alpha f}(t) \pm dv_{\beta f}(t)$ equation:

$$\begin{cases} v_{\alpha f}(t) \pm dv_{\beta f}(t) = V^+ \cdot \sin\left(\theta(t) + \phi^+\right) + V^- \cdot \sin\left(\theta(t) + \phi^-\right) \\ \qquad\qquad\qquad \pm \left(-V^+ \cdot \sin\left(\theta(t) + \phi^+\right) + V^- \cdot \sin\left(\theta(t) + \phi^-\right)\right) \\ v_{\beta f}(t) \pm dv_{\alpha f}(t) = V^+ \cdot \sin\left(\theta(t) + \phi^+ - \dfrac{\pi}{2}\right) - V^- \cdot \sin\left(\theta(t) + \phi^- - \dfrac{\pi}{2}\right) \\ \qquad\qquad\qquad \pm \left(V^+ \cdot \sin\left(\theta(t) + \phi^+ - \dfrac{\pi}{2}\right) + V^- \cdot \sin\left(\theta(t) + \phi^- - \dfrac{\pi}{2}\right)\right) \end{cases} \tag{5.28}$$

Which when separated yields:

$$\begin{cases} v_{\alpha f}(t) + dv_{\beta f}(t) &= 2 \cdot V^- \cdot \sin\left(\theta(t) + \phi^-\right) = 2 \cdot v_\alpha^-(t) \\ v_{\alpha f}(t) - dv_{\beta f}(t) &= 2 \cdot V^+ \cdot \sin\left(\theta(t) + \phi^+\right) = 2 \cdot v_\alpha^+(t) \\ v_{\beta f}(t) + dv_{\alpha f}(t) &= 2 \cdot V^+ \sin\left(\theta(t) + \phi^+ - \frac{\pi}{2}\right) = 2 \cdot v_\beta^+(t) \\ v_{\beta f}(t) - dv_{\alpha f}(t) &= -2 \cdot V^- \cdot \sin\left(\theta(t) + \phi^- - \frac{\pi}{2}\right) = 2 \cdot v_\beta^-(t) \end{cases} \tag{5.29}$$

Solving for the positive and negative sequence components:

$$\begin{cases} v_\alpha^+(t) &= \frac{1}{2} \cdot \left(v_{\alpha f}(t) - dv_{\beta f}(t)\right) \\ v_\alpha^-(t) &= \frac{1}{2} \cdot \left(v_{\alpha f}(t) + dv_{\beta f}(t)\right) \\ v_\beta^+(t) &= \frac{1}{2} \cdot \left(v_{\beta f}(t) + dv_{\alpha f}(t)\right) \\ v_\beta^-(t) &= \frac{1}{2} \cdot \left(v_{\beta f}(t) - dv_{\alpha f}(t)\right) \end{cases} \tag{5.30}$$

From these equations it is clear that isolating the positive and negative sequence components using the $v_{\alpha f}(t)$, $dv_{\alpha f}(t)$, $v_{\beta f}(t)$ and $dv_{\beta f}(t)$ is possible. However, the question of how to obtain these filtered signals remains.

## 5.5 Obtaining Filtered Signals For Component Extraction

Perhaps the most simple technique for extracting the symmetrical sequence components is via delayed signal cancellation (DSC)[6, 18, 31], a purely discrete method where the $-90°$ phase shifted signals are obtained by directly delaying the sampled signals $v_\alpha[n]$ and $v_\beta[n]$ (where $n$ denotes the current sample of the given signal). The amount of delay varies based the frequency estimated by the SRF-PLL. While the implementation of DSC is trivial, it does not provide any filtering, hence DC offsets, harmonics and noise will still be present.

A significantly better alternative to the DSC technique is the usage of a quadrature signal generator (QSG) based on the second-order generalized integrator (SOGI) to generate the filtered and phase shifted versions $v_\alpha(t)$ and $v_\beta(t)$ [32]. A QSG-SOGI is shown in Figure 5.1. As can be seen, the QSG-SOGI has a single tuning parameter, $k$, and two inputs, the input signal and the frequency. The transfer functions are given by:

$$\begin{aligned} D(s) &= \frac{X'(s)}{X(s)} = \frac{k \cdot \omega \cdot s}{s^2 + k \cdot \omega \cdot s + \omega^2} \\ Q(s) &= \frac{dX'(s)}{X(s)} = \frac{k \cdot \omega^2}{s^2 + k \cdot \omega \cdot s + \omega^2} \end{aligned} \tag{5.31}$$

The bode plots for various values of $k$ are shown in Figure 5.3. The smaller the value of $k$, the better the filtering capabilities of the QSG-SOGI are, but decreasing this value too much makes the QSG-SOGI more sensitive to signal deviations from the input frequency. Despite this, the output $dx'(t)$ always presents a $-90°$ phase shift from $x'(t)$ and the amplitudes are preserved at the resonant frequency, which is precisely what is required for isolating the positive and negative sequence components. Unfortunately, while D(s) behaves as a band-pass filter, Q(s) behaves as a low-pass filter, thus not removing the DC offsets.

While there are different ways to completely solve the problem of DC offset [33], a simple and highly effective way to deal with it is to cascade two QSG-SOGI blocks, where the transfer function $D(s)$ is used to pre-filter the input signal before a QSG-SOGI. This architecture is shown in 5.2, and the transfer functions are given by:

$$\begin{aligned} D_f(s) &= \frac{X'(s)}{X(s)} \cdot \frac{X''(s)}{X'(s)} = D(s) \cdot D(s) = \frac{k \cdot \omega \cdot s}{s^2 + k \cdot \omega \cdot s + \omega^2} \cdot \frac{k \cdot \omega \cdot s}{s^2 + k \cdot \omega \cdot s + \omega^2} \\ Q_f(s) &= \frac{X'(s)}{X(s)} \cdot \frac{dX''(s)}{X'(s)} = D(s) \cdot Q(s) = \frac{k \cdot \omega \cdot s}{s^2 + k \cdot \omega \cdot s + \omega^2} \cdot \frac{k \cdot \omega^2}{s^2 + k \cdot \omega \cdot s + \omega^2} \end{aligned} \tag{5.32}$$

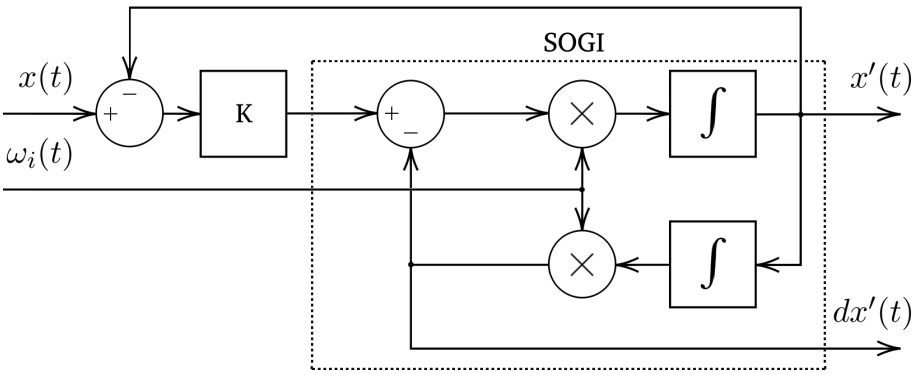



Figure 5.1: Structure of the QSG-SOGI

The bode plots for the cascaded QSG-SOGI for various values of $k$ are shown in Figure 5.4. It is clear from comparing Figures 5.3 and 5.4 that for the same values of $k$, the cascaded QSG-SOGI presents much better filtering capabilities.

Additionally, DC offset can be estimated by calculating $x(t) - x''(t)$[33], but noise and harmonics in the original $x(t)$ will affect the DC offset estimation, which could be filtered by an additional LPF with a very low cutoff frequency. Alternatively, computing $dx'(t) - dx''(t)$ and compensating for the gain at DC could result in a less noisy offset estimation, as $dx'(t)$ has its harmonics and high-frequency noise filtered. This is more computationally expensive since $dx'(t)$ would have to be computed for this specific feature.

The reason to choose the cascaded QSG-SOGI over the alternative method of adding another integrator to the QSG-SOGI for DC offset estimation is that this alternative method transforms the QSG-SOGI into a third-order system, which is not only harder to discretize later on, it also performs slightly worse since the cascaded QSG-SOGI has better filtering capabilities[33].

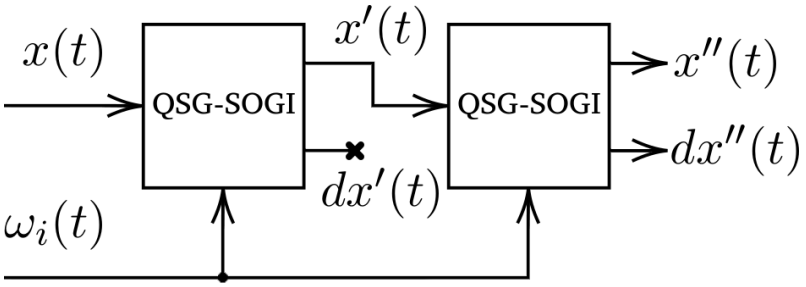



Figure 5.2: Cascaded QSG-SOGI

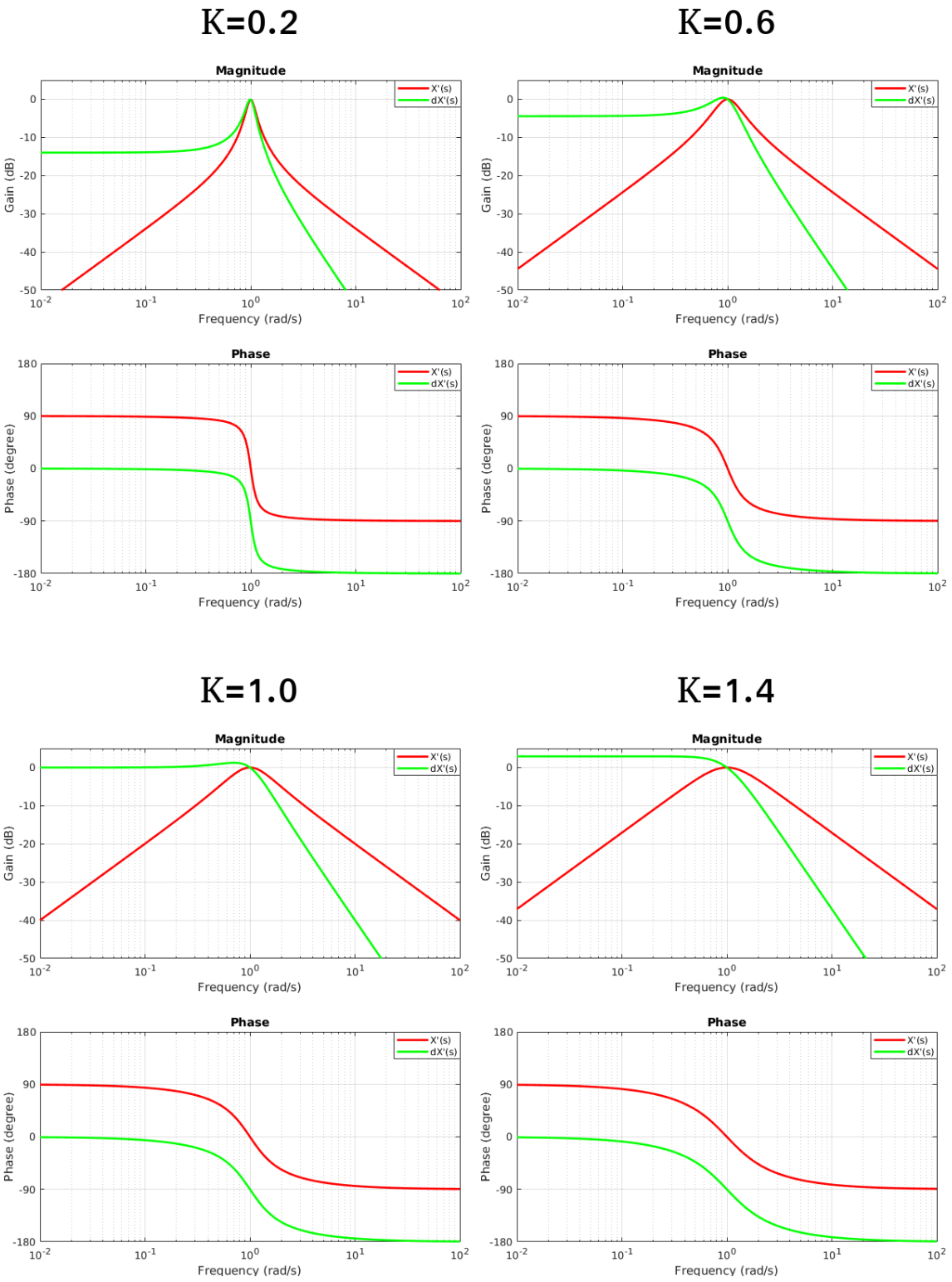



Figure 5.3: Bode plots for the QSG-SOGI

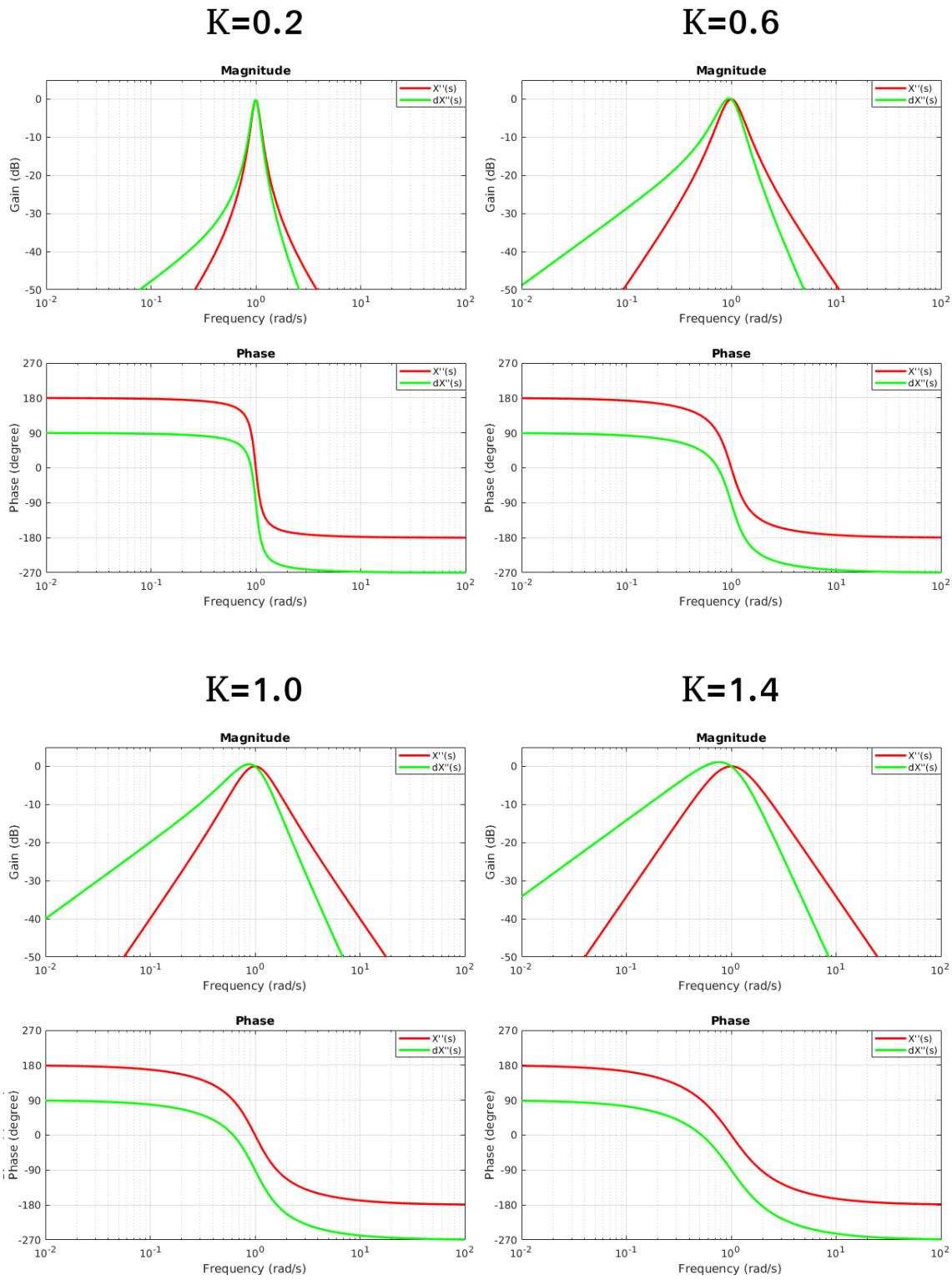



Figure 5.4: Bode plots for the cascaded QSG-SOGI

## 5.6 I-SRF-PLL with Extraction of Symmetrical Sequence Components

Using the cascaded QSG-SOGI, it is possible to obtain the filtered and phase shifted versions of $v_\alpha(t)$ and $v_\beta(t)$. A positive and negative sequence calculation (PNSC) block can then be implemented based on equations 5.30, as shown in Figure 5.5.

If the grid is close to ideal, the extraction of the positive sequence component provides a good estimation of the grid's angle and amplitude. The frequency estimated by the I-SRF-PLL is fed-back to the cascaded QSG-SOGI blocks for dynamic frequency adjustment.

If estimation of the negative sequence component is required (e.g. for employing corrective control techniques) then a second I-SRF-PLL whose inputs are $v_\alpha^-(t)$ and $v_\beta^-(t)$ can be used. Since $v_\beta^-(t)$ has an amplitude of $-V^-$, the negative component of the space vector rotates clockwise instead of anti-clockwise, thus leading to the I-SRF-PLL incorrectly estimating the angle of the negative sequence component as being $-(\theta(t) + \phi^-)$. Solving this requires either inverting the sign of $v_\beta^-(t)$ or modifying the PNSC block. To save computational resources, this I-SRF-PLL does not need to implement a FFF mechanism, since the output of the FFF mechanism of the positive sequence I-SRF-PLL can be used.

As a side note, under phase balance two I-SRF-PLLs are enough to obtain all parameters, including the zero sequence's amplitude and angle. As previously seen in Chapter 5.2.2, $\phi^+$ is zero, hence the positive I-SRF-PLL will estimate the grid angle $\theta(t)$. This can be subtracted to the negative I-SRF-PLL's angle to obtain $\phi^-$. From here, $\phi^0$ is simply $-\phi^-$ and $V^0$ is $V^-$.

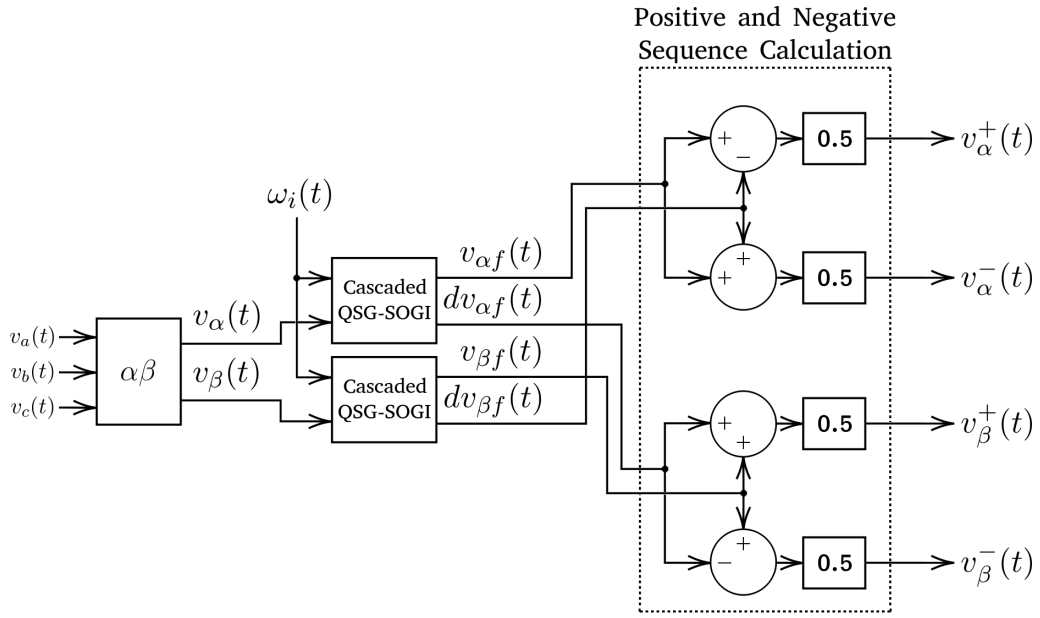



Figure 5.5: Positive and negative sequence calculation block

Obtaining the zero sequence component is not as straightforward as the other components. While the full amplitude-invariant $\alpha\beta$ transform features a zero sequence component, the transform is only amplitude-invariant with respect to $v_\alpha(t)$ and $v_\beta(t)$. To solve this problem, consider the average of the three-phase voltages:

$$\begin{aligned}
&\frac{1}{3}\cdot(v_a(t)+v_b(t)+v_c(t)+v_{DC})\\
=&\frac{1}{3}\cdot(v_a^+(t)+v_a^-(t)+v_a^0(t)+v_b^+(t)+v_b^-(t)+v_b^0(t)+v_c^+(t)+v_c^-(t)+v_c^0(t)+v_{DC})\\
=&\frac{1}{3}\cdot\left(v_a^0(t)+v_b^0(t)+v_c^0(t)\right)+\frac{1}{3}\cdot v_{DC}\\
=&\frac{1}{3}\cdot 3\cdot v_a^0(t)+\frac{1}{3}\cdot v_{DC}=V^0\cdot\sin\left(\theta(t)+\phi^0\right)+\frac{1}{3}\cdot v_{DC}
\end{aligned} \tag{5.33}$$

In the above equations, $v_{DC}$ is used as an all-encompassing term for the sum of the DC offsets.

Note how the average is equal to the zero sequence component plus some additional DC offset. To simulate the existence of $\alpha$ and $\beta$ components the cascaded QSG-SOGI can once again be used, resulting in a technique which resembles typical single-phase synchronization mechanisms[34]. However, the I-SRF-PLL for the zero sequence component is aided by the FFF mechanism of the positive sequence I-SRF-PLL, as well as its frequency estimation for the cascaded QSG-SOGI block. The signals generated by the cascaded QSG-SOGI are designated $v_{0f}(t)$ and $dv_{0f}(t)$.

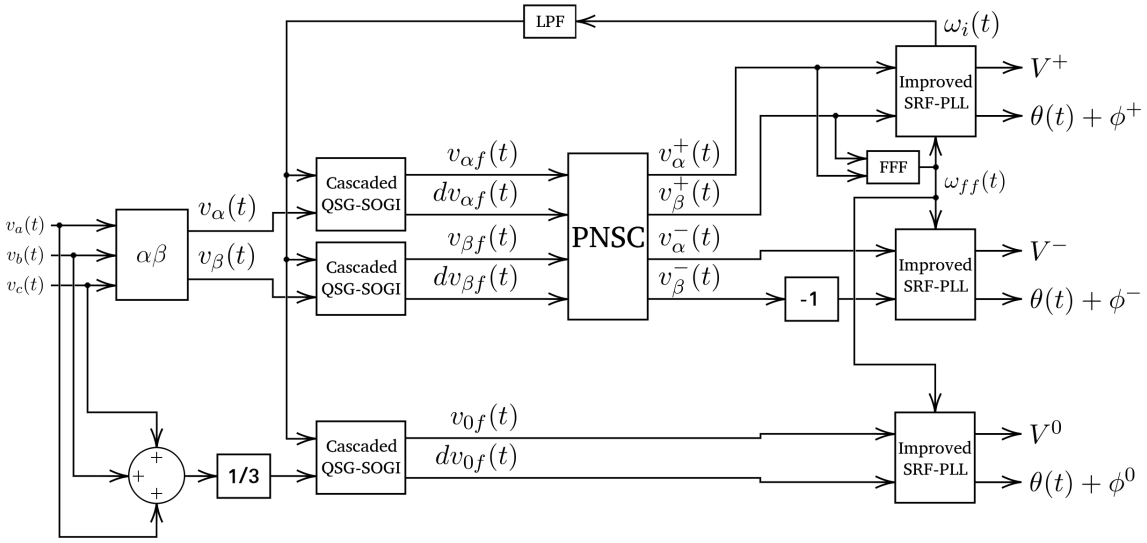



Figure 5.6: Full sequence extraction

The reason to include a low-pass filter on the frequency feed-back to the QSG-SOGIs is to avoid oscillatory behaviour. As can be seen in Figure 5.4, the magnitude of both transfer functions is only equal at the resonant frequency, therefore any deviations from it will cause the filtered signals to have different amplitudes. This distorts the circular path of the space vector, resulting in an ellipse, which in turn distorts the estimation of the angle via the atan2 function. This not only directly impacts the FFF mechanism, it also impacts the ANS since an ellipse path is equivalent to having a circular path with a varying space vector magnitude. Specifically, oscillations at the grid's frequency occur.

If these oscillations are not eliminated via filtering, they cause the QSG-SOGI to constantly "miss" the resonant frequency while the I-SRF-PLL is constantly receiving a distorted input. Figure 5.8 shows the results of the positive sequence component I-SRF-PLL in the same conditions as Figure 4.6 without the low-pass filter. Due to the oscillations previously mentioned, the I-SRF-PLL is not able to lock in to the angle of the positive sequence component. In Figure 5.9, the LPF is added with a cutoff frequency of 10 Hz, which makes the I-SRF-PLL stable again. For the cascaded QSG-SOGIs, a value of $k = 3$ was chosen.

Since a lower bound to the input frequency is required - otherwise the QSG-SOGIs would always output zero and the I-SRF-PLLs would be locked in their initial state - a range between 40 Hz and 70 Hz was chosen to speed up the locking process.

Please note that the angle errors shown in Figures 5.8 and 5.9 are between the estimated angles and the real angle of the positive sequence component, which was computed using equations 5.15 and 5.18.

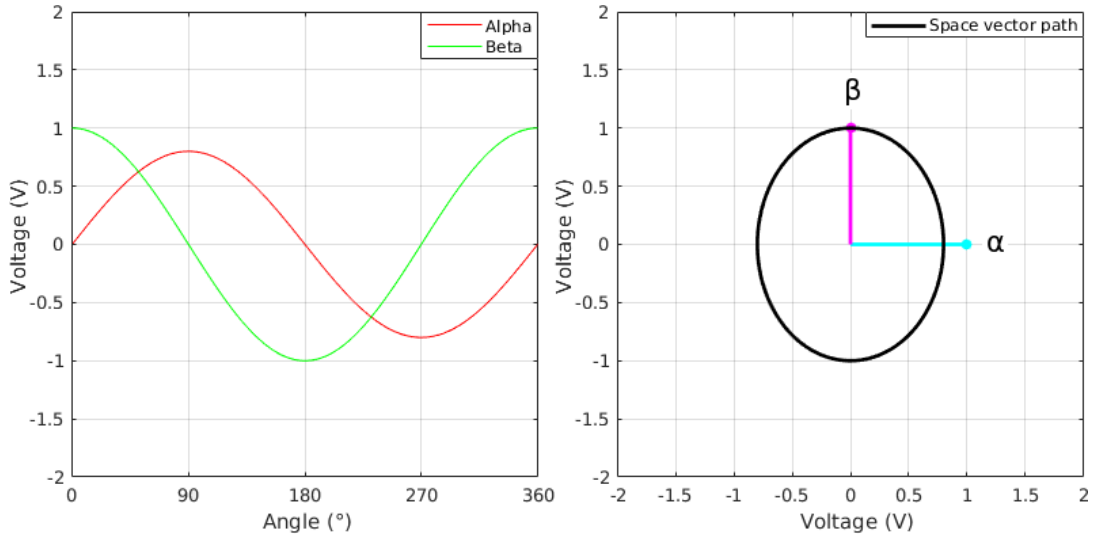



Figure 5.7: Space vector's path with alpha and beta components of different amplitudes

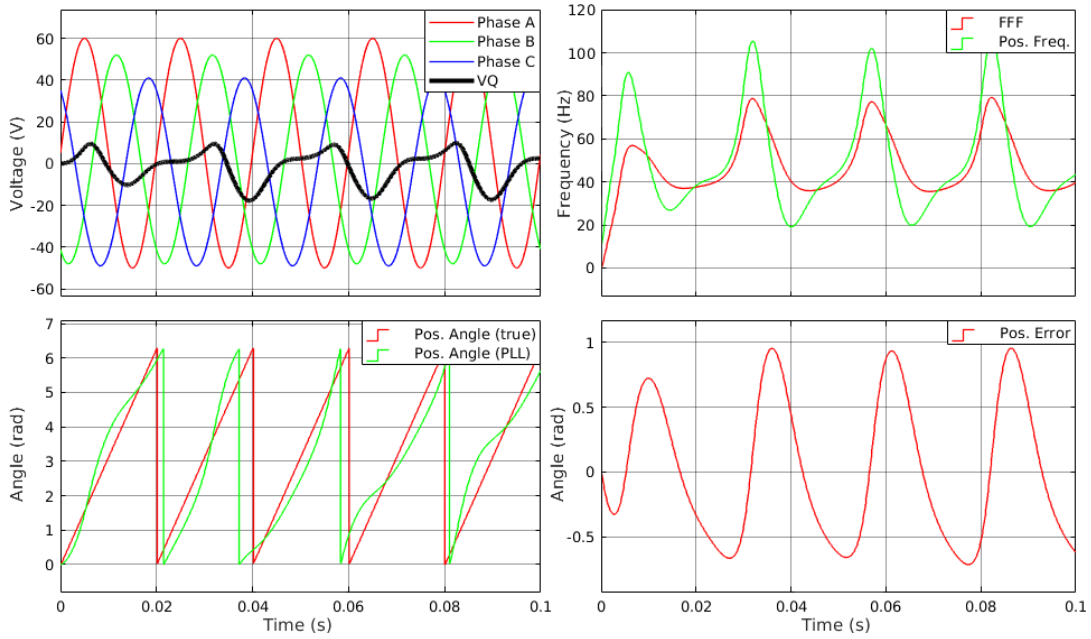



Figure 5.8: Performance of the positive sequence I-SRF-PLL without QSG-SOGI input frequency LPF in the same conditions as Figure 4.6

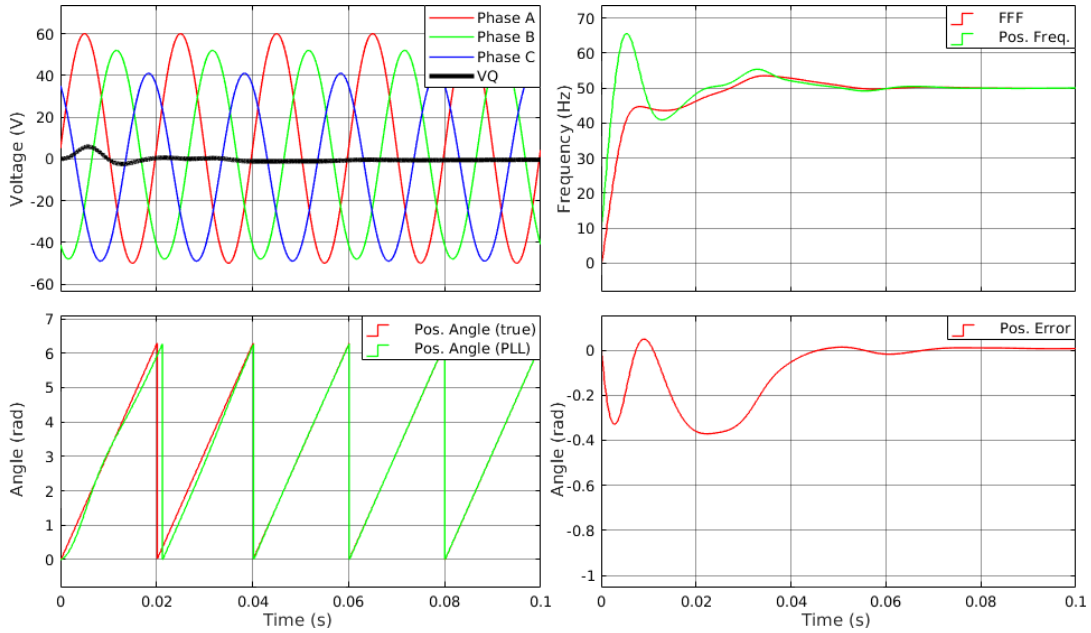



Figure 5.9: Performance of the positive sequence I-SRF-PLL with QSG-SOGI input frequency LPF in the same conditions as Figure 4.6

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Chapter 6

# Discretization and Implementation

## 6.1 Introduction

This chapter focuses on the discretization and implementation of the different components of the I-SRF-PLL, as well as the other filtering blocks discussed in previous chapters.

The amount of time taken to compute each part of the algorithm is measured. While these measurements are specific to the microcontroller used, the relative times still serve as a useful guideline.

## 6.2 Cascaded QSG-SOGI

### 6.2.1 Discretization

In order to implement the cascaded QSG-SOGI in a digital device, a discretization method known as the bilinear transform - also known as Tustin's method - is considered to be the most adequate technique[35]. This is due to the non-linear distortion of the frequency axis, which strongly distorts the frequency and phase response near the Nyquist frequency, but is negligible for low frequencies[36, 37].

A generic second-order transfer function is given by:

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_2 \cdot s^2 + b_1 \cdot s + b_0}{a_2 \cdot s^2 + a_1 \cdot s + a_0} \tag{6.1}$$

The discrete version of H(s) approximated by bilinear transform with a sampling period of $T_s$ is given by[1]:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{C_0^b + C_1^b \cdot z^{-1} + C_2^b \cdot z^{-2}}{C_0^a + C_1^a \cdot z^{-1} + C_2^a \cdot z^{-2}} \tag{6.2}$$

[1]Proof provided in Appendix F

Where:

$$\begin{cases} C_0^b = \frac{b_2 \cdot \left(\frac{2}{T_s}\right)^2 + b_1 \cdot \frac{2}{T_s} + b_0}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \\ C_1^b = \frac{-2 \cdot b_2 \cdot \left(\frac{2}{T_s}\right)^2 + 2 \cdot b_0}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \\ C_2^b = \frac{b_2 \cdot \left(\frac{2}{T_s}\right)^2 - b_1 \cdot \frac{2}{T_s} + b_0}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \end{cases} \qquad \begin{cases} C_0^a = 1 \\ C_1^a = \frac{-2 \cdot a_2 \cdot \left(\frac{2}{T_s}\right)^2 + 2 \cdot a_0}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \\ C_2^a = \frac{a_2 \cdot \left(\frac{2}{T_s}\right)^2 - a_1 \cdot \frac{2}{T_s} + a_0}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \end{cases} \tag{6.3}$$

For the transfer function $D(s)$, the equivalent discrete time coefficients are:

$$\begin{cases} DC_0^b = \frac{k \cdot \omega \cdot \frac{2}{T_s}}{\left(\frac{2}{T_s}\right)^2 + k \cdot \omega \cdot \frac{2}{T_s} + \omega^2} \\ DC_1^b = 0 \\ DC_2^b = -\frac{k \cdot \omega \cdot \frac{2}{T_s}}{\left(\frac{2}{T_s}\right)^2 + k \cdot \omega \cdot \frac{2}{T_s} + \omega^2} \end{cases} \qquad \begin{cases} DC_0^a = 1 \\ DC_1^a = \frac{-2 \cdot \left(\frac{2}{T_s}\right)^2 + 2 \cdot \omega^2}{\left(\frac{2}{T_s}\right)^2 + k \cdot \omega \cdot \frac{2}{T_s} + \omega^2} \\ DC_2^a = \frac{\left(\frac{2}{T_s}\right)^2 - k \cdot \omega \cdot \frac{2}{T_s} + \omega^2}{\left(\frac{2}{T_s}\right)^2 + k \cdot \omega \cdot \frac{2}{T_s} + \omega^2} \end{cases} \tag{6.4}$$

For the transfer function $Q(s)$, the equivalent discrete time coefficients are:

$$\begin{cases} QC_0^b = \frac{k \cdot \omega^2}{\left(\frac{2}{T_s}\right)^2 + k \cdot \omega \cdot \frac{2}{T_s} + \omega^2} \\ QC_1^b = \frac{2 \cdot k \cdot \omega^2}{\left(\frac{2}{T_s}\right)^2 + k \cdot \omega \cdot \frac{2}{T_s} + \omega^2} \\ QC_2^b = \frac{k \cdot \omega^2}{\left(\frac{2}{T_s}\right)^2 + k \cdot \omega \cdot \frac{2}{T_s} + \omega^2} \end{cases} \qquad \begin{cases} QC_0^a = 1 \\ QC_1^a = \frac{-2 \cdot \left(\frac{2}{T_s}\right)^2 + 2 \cdot \omega^2}{\left(\frac{2}{T_s}\right)^2 + k \cdot \omega \cdot \frac{2}{T_s} + \omega^2} \\ QC_2^a = \frac{\left(\frac{2}{T_s}\right)^2 - k \cdot \omega \cdot \frac{2}{T_s} + \omega^2}{\left(\frac{2}{T_s}\right)^2 + k \cdot \omega \cdot \frac{2}{T_s} + \omega^2} \end{cases} \tag{6.5}$$

### 6.2.2 Implementation

The key to an efficient implementation is a fast computation of the coefficients by reducing the number of redundant calculations. In equations 6.4 and 6.5, notice that:

$$\begin{cases} QC_1^a &= DC_1^a \\ QC_2^a &= DC_2^a \\ DC_2^b &= -DC_0^b \\ QC_2^b &= QC_0^b \\ QC_1^b &= 2 \cdot QC_0^b \end{cases} \tag{6.6}$$

Making use of these relationships should significantly reduce computational effort. Additionally, since $\omega$ is the only variable then multiple constants can be pre-calculated, including the denominator common to all non-trivial coefficients. Taking these points into consideration, the most efficient method for computing the coefficients is given in Algorithm 1. It requires 4 sums, 10 multiplications and 1 division. Fortunately, the coefficients are the same for all cascaded SOGI-QSGs, hence they only need to be computed once per time-step.

The reason for computing the inverse of the denominator is that subsequent divisions become multiplications, which are much more efficiently implemented in modern hardware. As an example, in the ARM Cortex-M4 processor both floating point and integer addition and multiplication instructions can be executed in a single cycle, whereas division can take anywhere between 2 and 12 cycles for integers, and is guaranteed to take 14 cycles for floating point variables[38].

**Algorithm 1** Cascaded QSG-SOGI Coefficient Calculation

**Require:**

$const\_1 = k \cdot \frac{2}{T_s}$

$const\_2 = -const\_1$

$const\_3 = \left(\frac{2}{T_s}\right)^2$

$const\_4 = -const\_3$

**Input:** $\omega$

1: $omega\_squared = \omega * \omega$

2: $var\_1 = const\_3 + omega\_squared$

3: $denom\_inv = 1 \ / \ (const_1 * \omega + var\_1)$

4: $D\_b0 = const\_1 * \omega * denom\_inv$

5: $D\_a1 = 2 * (const\_4 + omega\_squared) * denom\_inv$

6: $D\_a2 = (var\_1 + const\_2 * \omega) * denom\_inv$

7: $Q\_b0 = k * omega\_squared * denom\_inv$

Once the coefficients are calculated, the outputs of the cascaded QSG-SOGI can be calculated using Algorithm 2, requiring 10 sums/subtractions and 10 multiplications.

**Algorithm 2** Cascaded QSG-SOGI algorithm

**Input:** $x[n]$

1: $xf = DC_0^b * (x[n] - prev2\_x) - DC_1^a * prev1\_xf - DC_2^a * prev2\_xf$

2: $dxf = DC_0^b * (xf - prev2\_xf) - DC_1^a * prev1\_dxf - DC_2^a * prev2\_dxf$

3: $qxf = QC_0^b * (xf + 2 * prev1\_xf + prev2\_xf) - DC_1^a * prev1\_qxf - DC_2^a * prev2\_qxf$

4: $prev2\_x = prev1\_x$

5: $prev1\_x = x[n]$

6: $prev2\_xf = prev1\_xf$

7: $prev1\_xf = xf$

8: $prev2\_dxf = prev1\_dxf$

9: $prev1\_dxf = dxf$

10: $prev2\_qxf = prev1\_qxf$

11: $prev1\_qxf = qxf$

### 6.2.3 Hardware Benchmark

To benchmark the performance of the cascaded QSG-SOGI algorithm and subsequent algorithms in this chapter, a XMC4700-F144 microcontroller evaluation board is used[39]. The microcontroller is based on the Cortex-M4 processor. The algorithms are translated into plain C which is compiled with GCC using the **-O3** flag, and execution time is measured via a debug pin that is set high while the microcontroller is processing algorithms 1 and 2. Table 6.1 shows the execution times for both algorithms and also for the total time required for two QSG-SOGIs sharing the same coefficients.

Table 6.1: Cascaded QSG-SOGI measured worst-case execution times

| Coefficients | QSG-SOGI | Coefficients + two QSG-SOGIs |
|---|---|---|
| 125 ns | 350 ns | 750 ns |

The measurements were performed separately, as the additional processor instructions to set and reset the multiple debug pins to measure all execution times simultaneously had a noticeable impact. While the total execution time should be equivalent to one calculation of Algorithm 1 and two calculations of Algorithm 2, the reason why this is not the case is twofold. The first reason is bandwidth, as the logic analyzer used has a maximum bandwidth of 25Mhz and the microcontroller pins have non-negligible rise and fall times as compared to the execution times of the algorithms. The second reason is the additional instructions that the microcontroller has to execute to change the state of the debug pin, which not only take time to execute but also affect instruction pipelining[38]. Therefore, the measured execution times should always be taken as rough estimates.

One logical approach to achieve better measurements of the execution time is to take the measurements of multiple runs of the algorithms and then dividing the execution time by the number of runs. Unfortunately, this produces unexpected side-effects. While instruction pipelining reduces the cycles required for each random-access memory (RAM) access, the values still have to be loaded into the floating point unit (FPU) register banks to perform calculations[38]. Since these algorithms have a relatively high number of variables and most calculations are additions, subtractions and multiplications which are executed in a single cycle, the overhead of moving data between RAM and FPU registers cannot be ignored. When an algorithm is executed multiple times without any other operations in-between, the compiler can optimize variable access by reutilizing registers across its multiple iterations. Thus, measuring consequent runs of the algorithms results in a significant decrease in execution time which cannot be justified simply by the logic analyzer's bandwidth or the debug pin's rise and fall speeds. To counter this, the algorithms are executed periodically, with the microcontroller running FreeRTOS with a few tasks, such as blinking an LED and displaying some text on a display, thus preventing such register-based optimizations between different runs of the algorithms.

## 6.3 Frequency Feed-Forward

### 6.3.1 Discretization

As discussed in Chapter 4.3.4, the FFF mechanism consists of computing the arctangent to obtain an estimation of the angle, and then computing the derivative and low-pass filter the result.

One possible way to define a derivative is[37]:

$$\frac{dy}{dt} = \lim_{h \to 0} \frac{y(x) - y(x-h)}{h} \tag{6.7}$$

The discretization can be realized by the backward difference approximation (sometimes known as the backward Euler approximation), which simply takes the previous equation and removes the limit[37]:

$$\frac{dy}{dt} \approx \frac{y(x) - y(x-h)}{h} \tag{6.8}$$

Since $h$ represents the time difference between $y(n-h)$ and $y(n)$, then in the context of digital systems it represents the sampling period $T_s$. Similarly, $y(x)$ represents the current sample, while $y(x-h)$ is the previous sample. Thus, the equation for calculating the feed-forward frequency (before the LPF) is:

$$\omega_{ff\ pre\ filter}[n] = \frac{\theta_{ff}[n] - \theta_{ff}[n-1]}{T_s} \tag{6.9}$$

As $\frac{1}{T_s}$ can be calculated in advance, the derivative only requires 1 subtraction and 1 multiplication.

A Butterworth first-order LPF with a cut-off frequency of $\omega_c$ is given by the following transfer function[25]:

$$H_{LPF}(s) = \frac{1}{1 + \frac{s}{\omega_c}} = \frac{\omega_c}{s + \omega_c} \tag{6.10}$$

The discretization of this LPF via the backward difference method results in the following difference equation[2]:

$$y[n] = a \cdot x[n] + (1-a) \cdot y[n-1] \tag{6.11}$$

Where:

$$a = \frac{\omega_c \cdot T_s}{1 + \omega_c \cdot T_s} \tag{6.12}$$

Since the coefficients $a$ and $a-1$ can be pre-calculated, the LPF requires 2 multiplications and 1 sum.

[2]Proof provided in Appendix G

### 6.3.2 Implementation

While the implementation of the FFF mechanism seems quite straightforward, the calculation of the atan2 function is quite computationally intensive. To solve this, an approximation is used.

In literature, there are a variety of well-documented arctangent approximations. The following approximation appears to be quite popular[36, 40]:

$$\operatorname{atan}\left(\frac{y}{x}\right) = \frac{\frac{y}{x}}{1 + 0.28125 \cdot \left(\frac{y}{x}\right)^2} \tag{6.13}$$

Where $y$ and $x$ are the coordinates of a given point. This technique achieves a maximum error of 0.26 ° and requires 1 division, 1 sum and 2 multiplications, assuming the result of the division $\frac{y}{x}$ is already known. It should be noted that the output of this approximation is in degrees and not radians.

Note that while the division $\frac{y}{x}$ is unavoidable in any approximation, equation 6.13 requires another division. Since division is an order of magnitude slower than other operations[38], an approximation without divisions is preferred.

Another approximation which does not require the divisions is given by[40]:

$$\operatorname{atan}\left(\frac{y}{x}\right) = \frac{\pi}{4} \cdot \frac{y}{x} + 0.273 \cdot \frac{y}{x} \cdot \left(1 - \left|\frac{y}{x}\right|\right) \tag{6.14}$$

This approximation features a maximum error of 0.22 °, and requires 3 multiplications, 2 sums and 1 absolute value calculation. In the ARM Cortex-M4, a floating-point absolute value can be calculated in a single cycle[38], which makes this approximation much more efficient than the one given in 6.13.

A third, less well-known option is a simple 3rd order polynomial which is given by[41]:

$$\operatorname{atan}\left(\frac{y}{x}\right) = 0.9724 \cdot \frac{y}{x} - 0.1919 \cdot \left(\frac{y}{x}\right)^3 \tag{6.15}$$

This can be rewritten as:

$$\operatorname{atan}\left(\frac{y}{x}\right) = \left(0.9724 - 0.1919 \cdot \left(\frac{y}{x}\right)^2\right) \cdot \frac{y}{x} \tag{6.16}$$

This approximation has a maximum error of 0.29 °. The rewritten version requires 1 sum and 3 multiplications, which is slightly more efficient than the approximation given by equation 6.14.

While many other approximations exist[40], for the purposes of the FFF mechanism an error of 0.29 ° is acceptable since the LPF should filter out the fluctuations in frequency resulting from the approximation's error. Since the approximation given by equation 6.16 is very light on computational resources, this is the one chosen for the implementation of the FFF. Then, implementing the atan2 function consists of checking the sign of $x$ and $y$ and exploiting simple trigonometric relationships[41].

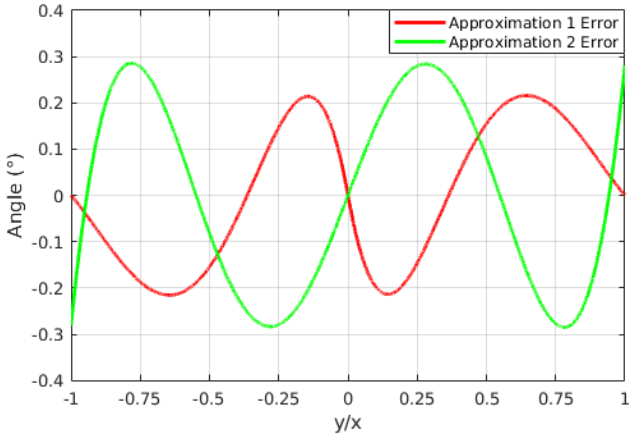



Figure 6.1: Comparison between two arctangent approximations: Approximations 1 and 2 correspond to equations 6.14 and 6.16, respectively

Another implementation problem specific to the FFF mechanism arises in the form of derivative spiking. Derivative spiking in the FFF mechanism can occur in two different ways: noise and angle wrap-around. Derivative spiking can create significant problems when dealing with floating-point arithmetic, as a loss of precision occurs the bigger the represented numbers are.

Since the derivative is by definition very sensitive to high-frequency signals, noise can be particularly problematic. While in Chapter 4 the FFF mechanism is presented as having the LPF after the derivative, a more desirable implementation would have the LPF before the derivative to avoid derivative spiking. Since both the LPF and the derivative are linear and time invariant, the output is unaffected by swapping the order of the two blocks.

The reason why angle wrap-around can cause derivative spiking is somewhat self-explanatory. As the output of atan2 increases between iterations, it eventually reaches $\pi$, at which point the atan2 function considers the angle to be negative, effectively wrapping around to $-\pi$. To the derivative, this appears as a extreme variation in angle, as if it decreased by $2\pi$ in a single time-step, when in reality the angle just varied slightly. As the actual angle did not decrease between iterations, placing the LPF after the atan2 also does not solve this problem. The LPF will also perceive the fictitious decrease in angle, which, while avoiding derivative spiking, will produce an erroneous FFF output as the frequency appears to decrease for a brief period of time. To solve this problem, the FFF algorithm needs to additionally compute the smallest angular distance between the current angle and the angle of the last iteration.

Algorithm 3 demonstrates how to compute the FFF mechanism while avoiding derivating spiking. It should be noted that the variable *angular_dist* is reused as the output of the LPF to reduce the number of variables required. In order to avoid the costly division in the derivative, the sample frequency $F_s$ is used instead of the sampling rate $T_s$.

**Algorithm 3** FFF Algorithm

**Require:**

$a = \frac{\omega_c \cdot T_s}{1+\omega_c \cdot T_s}$

$b = 1 - a$

$F_s = \frac{1}{T_s}$

**Input:** $v_\alpha^+,\ v_\beta^+$

1: $theta = \text{atan2}(v_\beta^+, v_\alpha^+)$
2: $angular\_dist = theta - prev\_theta$
3: **if** $angular\_dist \leq -\pi$ **then**
4: $\quad angular\_dist = angular\_dist + 2 \cdot \pi$
5: **else if** $angular\_dist \geq \pi$ **then**
6: $\quad angular\_dist = angular\_dist - 2 \cdot \pi$
7: **end if**
8: $angular\_dist = a \cdot angular\_dist + b \cdot prev\_angular\_dist$
9: $frequency = angular\_dist \cdot F_s$
10: $prev\_theta = theta$
11: $prev\_angular\_dist = angular\_dist$

In the worst-case execution path the FFF algorithm requires 1 atan2 calculation, 2 comparisons, 3 multiplications, 3 sums/subtractions. This path is the one where the first comparison in line 3 fails and the second one in line 5 succeeds, thus performing the calculation in line 6.

Ignoring the comparisons and adding the operations required for the atan computation using the approximation from equation 6.16, the algorithm totals 1 division, 6 multiplications and 4 sums/subtractions. This also ignores the branching and additional calculations required for computing the atan2 from the atan approximation.

### 6.3.3 Hardware Benchmark

In the implementation process where the algorithms are translated into plain C, the trigonometric approximations were implemented in a different source file. While this should not pose any performance difference, more than two microseconds were required just for the calculation of the atan2 function. While the atan2 function with the approximation is still somewhat intensive due to the division and branching required, this execution time suggested that the compiler was missing some crucial optimization steps.

After further investigation, it was clear from the output of the disassembler that some registers were being used in a non-optimal way. While the GCC compiler was optimizing individual compilation units, it was not performing optimizations once the final machine code was linked together, which results in missed optimization opportunities such as inlining functions and skipping unnecessary register context switches.

To solve this, link-time optimization was enabled by adding the **-flto** optimization flag[42], but this unfortunately resulted in a compilation error in FreeRTOS's internal function **vTaskSwitchContext()**. Since FreeRTOS's source code only performs calls to this function via inline assembly, GCC removes **vTaskSwitchContext()** because it does not realize that this function is used[43]. A simple workaround is to place a call to this function after the FreeRTOS's scheduler's function call, which due to it being an infinite loop, the call to **vTaskSwitchContext()** never occurs. Other options include altering FreeRTOS's source code and using GCC's **__attribute__((used))** instruction.

Table 6.2: FFF measured worst-case execution times

| FFF with standard atan2 | FFF with approximate atan2 |
|---|---|
| 30.44 µs | 1 µs |

## 6.4 I-SRF-PLL

### 6.4.1 Discretization

#### Amplitude Normalization Scheme

As the ANS is based around simple arithmetic computations and low-pass filtering (as previously shown in Figure 4.10), the LPF discretization process used for the FFF mechanism can simply be re-used.

#### PI Controller

The PI controller, in its most usual form, is given by[44]:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau) \cdot d\tau \tag{6.17}$$

Where $u(t)$ is the controller's output, $e(t)$ is the input error and $K_p$ and $K_i$ are the gains for the proportional and integral parts of the controller, respectively.

Using the backward difference approximation, the difference equation becomes[3]:

$$u[n] = u[n-1] + K_p \cdot (e[n] - e[n-1]) + K_i \cdot T_s \cdot e[n] \tag{6.18}$$

The PI requires 3 sums and 2 multiplications, given that $K_i \cdot T_s$ is calculated in advance.

#### Controllable Oscillator

The controllable oscillator follows the same logic as the integral component of the PI controller, hence the difference equation is:

$$\hat{\theta}[n] = \hat{\theta}[n-1] + T_s \cdot \omega_i[n] \tag{6.19}$$

[3]Proof provided in Appendix H

### Circular Dependency Problem

When discretizing any system it is important to note where circular dependencies might exist. Directly substituting all continuous-time components of the I-SRF-PLL by their previously described discrete-time counterparts results in a circular dependency as the PLL's output $\hat{\theta}[n]$ depends on the PI controller's output, which is determined by the output of the ANS, which in turn uses the output of the DQ0-QA transform. Unfortunately, the output of the DQ0-QA transform is dependent on $\theta_r[n]$, which is equal to $\hat{\theta}[n]$. Therefore, computing $\hat{\theta}[n]$ requires $\hat{\theta}[n]$ to be computed, which is a logical impossibility.

The easiest way to solve this circular dependency is to add a unit delay in the feedback path, with $\theta_r[n]$ becoming $\hat{\theta}[n-1]$. This approach introduces a slight phase delay and results in non-zero steady-state error. The reason for this is that in order for $v_q[n]$ to be zero, the D axis must be coincident with the space vector, and the only way for this to happen is to have the DQ0-QA's rotational angle $\theta_r[n]$ be equal to the space vector's angle, i.e. the grid angle assuming a perfectly balanced grid.

Another solution is to place the delay in the forward path, i.e. somewhere between the output of the DQ0-QA transform and the controllable oscillator. The delay is therefore placed before the controllable oscillator, making $\theta_r[n]$ equal to $\hat{\theta}[n]$ at the cost of the I-SRF-PLL not being able to react to a disturbance in a single time-step.

This solution is also equivalent to substituting the backward difference approximation by the forward difference approximation[37]:

$$\hat{\theta}[n] = \hat{\theta}[n-1] + T_s \cdot \omega_i[n-1] \tag{6.20}$$

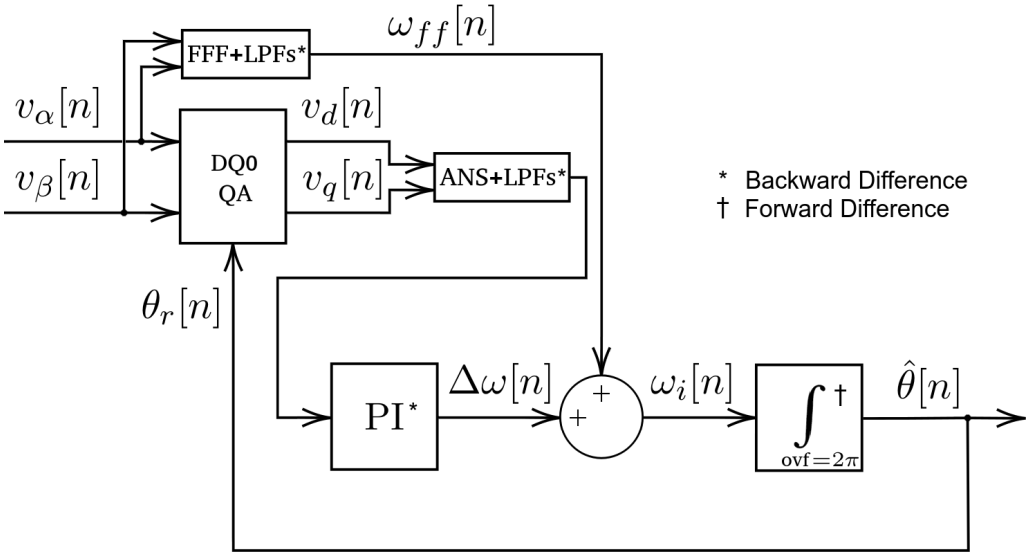



Figure 6.2: Discrete version of the I-SRF-PLL with FFF mechanism

### 6.4.2 Implementation

#### DQ0 Transform

Not unlike the FFF mechanism, the implementation of the I-SRF-PLL is performance-bound by the sine and cosine functions in the DQ0 transform. Due to the symmetries of the sine function, it is easy to create a look-up table (LUT) for only a quarter-wave cycle, and adapt the LUT values depending on which quadrant the angle is located, as shown in Figure 6.3. Then, implementing the cosine function is trivial since it is simply a shift of the sine function.

A useful approximation to interpolate between LUT positions is derived from the following relation[45]:

$$\sin(a+b) = \sin(a) \cdot \cos(b) + \sin(b) \cdot \cos(a) \tag{6.21}$$

If the LUT input is composed of $\theta + h$, where $\theta$ is some exact entry of the LUT and $h$ is a small difference which is not enough to reach the next LUT entry, then interpolation can be performed based on the assumption that $h$ is small enough such that $\cos(h) \approx 1$ and $\sin(h) \approx h$. This leads to the following approximation:

$$\sin(\theta + h) \approx \sin(\theta) + h \cdot \cos(\theta) \tag{6.22}$$

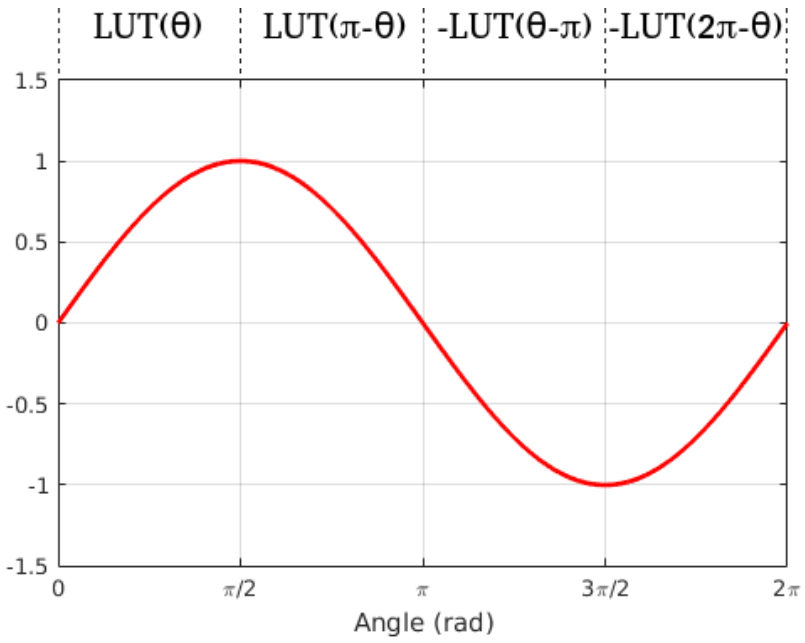



Figure 6.3: Quarter-wave sine function LUT

Table 6.3: Sine LUT maximum error comparison

| Size / Approx. 6.22 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|
| With | 1.07 ° | 0.52 ° | 0.26 ° | 0.13 ° | 0.07 ° | 0.04 ° |
| Without | 2.91 ° | 1.43 ° | 0.71 ° | 0.36 ° | 0.18 ° | 0.09 ° |

Table 6.3 presents the maximum error of the LUT-based approach. The least significant digit is rounded up in every case, so the true maximum error is guaranteed to be below the value on the table. It should be noted that due to the LUT being only a quarter of the wave, the resolution is effectively quadrupled when compared to a full-wave LUT, albeit at a slight computational cost. Additionally, the approximation from equation 6.22 has a similar effect to more than doubling the table size, again at a slight computational cost.

After programming the sine LUT in C, the DQ0-QA transform can be effortlessly calculated via equation 3.16.

### Amplitude Normalization Scheme

As previously discussed in Chapter 4.3.2, the cost of computing the norm of the space vector is quite significant due to the square-root operation. Fortunately, this can be mitigated by using the Alpha-max plus Beta-min algorithm[36]. This approximation computes the norm of any given vector $Z = (a, b)$ and is given by:

$$\|Z\| \approx \alpha \cdot \mathrm{Max}(a, b) + \beta \cdot \mathrm{Min}(a, b) \tag{6.23}$$

While there are many possible values for $\alpha$ and $\beta$, the optimum floating-point values that give the lowest maximum error are[46]:

$$\begin{cases} \alpha = \frac{2 \cdot \cos(\frac{\pi}{8})}{1 + \cos(\frac{\pi}{8})} \approx 0.96043387010341996524 \\ \beta = \frac{2 \cdot \sin(\frac{\pi}{8})}{1 + \cos(\frac{\pi}{8})} \approx 0.39782473475931601382 \end{cases} \tag{6.24}$$

This approximation requires 1 comparison, 2 multiplications and 1 sum, which for the specific case of the ARM Cortex-M4 is faster than the 14 cycle long square-root instruction[38]. The approximation's maximum error is about 3.96 %.

### PI Controller

While the implementation of the PI controller is easy, adding limits to the output can be useful to avoid instability. Limiting the output can be done in such a way that the internal variables of the PI controller are also bounded - which is effectively a form of anti-windup. Another easy to add feature is limiting the amount of integral gain per cycle, which might prove useful later but for now is left unused. Algorithm 4 demonstrates how to implement the PI controller with these features.

**Algorithm 4** PI Algorithm

**Require:**

$const\_1 = ki \cdot T_s$

**Input:** $error\_input$

$proportional\_gain = kp * (error\_input - prev\_error\_input)$
$integral\_gain = const\_1 \cdot error\_input$
**if** $integral\_gain > integral\_limit$ **then**
    $integral\_gain = integral\_limit$
**else if** $integral\_gain < -integral\_limit$ **then**
    $integral\_gain = -integral\_limit$
**end if**
$output = prev\_output + proportional\_gain + integral\_gain$
**if** $output > max\_output$ **then**
    $output = max\_output$
**else if** $output < min\_output$ **then**
    $output = min\_output$
**end if**
$prev\_output = output$
$prev\_error\_input = error\_input$

#### Controllable Oscillator

In order to make sure that the outputs from the FFF mechanism and PI controller do not generate a negative frequency input for the controllable oscillator during transients, input limiting logic is added. The rest of the implementation is simply a direct application of equation 6.20.

### 6.4.3 Hardware Benchmark

For the I-SRF-PLL benchmarks, the performance with and without the amplitude and trigonometry approximations is compared in Table 6.4. It is clear that the standard sine and cosine functions take a very significant amount of time, even when compared to the expensive square-root operation required for the ANS.

Table 6.4: I-SRF-PLL measured worst-case execution times

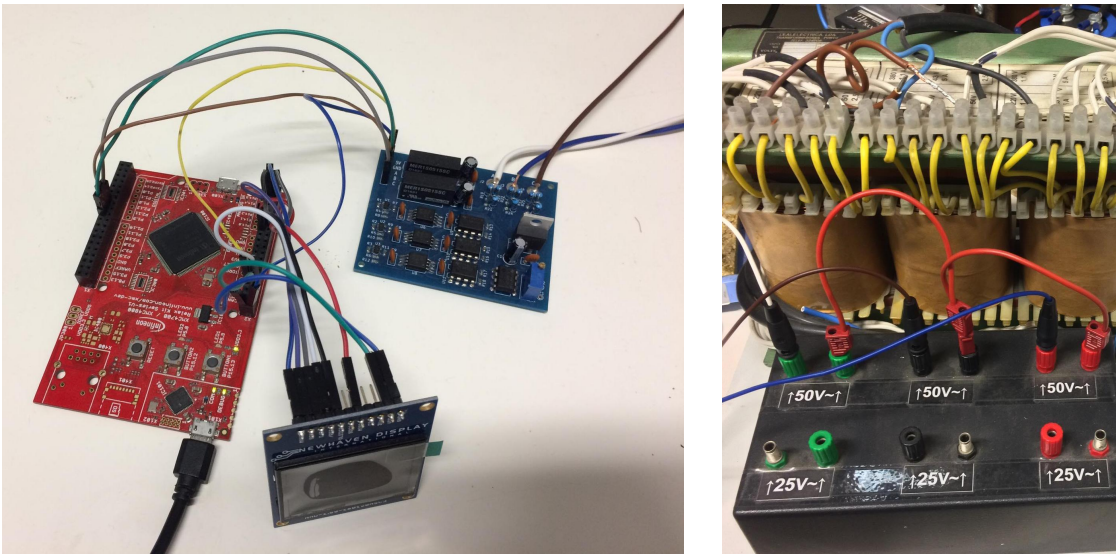
| No approximations/LUT | Amplitude approx. only | LUT only | Both |
|---|---|---|---|
| 44.75 µs | 31.75 µs | 12 µs | 3.75 µs |

## 6.5 Results

In order to test the discretization and implementation steps in a real-world application, a custom signal conditioning board featuring full isolation was developed to convert real three-phase voltages from a transformer into voltage levels appropriate for the XMC's analog-to-digital converter (ADC). A full schematic is available in Appendix I.

A timer was set up to trigger ADC conversions at a rate of 10 kHz (100 µs). After each conversion, the CPU executes the grid synchronization algorithm, taking approximately 10 µs. Using two digital-to-analog converters (DACs), phase A (as converted by the ADC) and the angle of the positive sequence component can be measured by an oscilloscope, as shown in Figure 6.5. The sine wave from phase A was also recreated from the estimated angle and amplitude of the positive sequence component, as shown in Figure 6.6. As expected from the previous simulation results in Figure 5.9, two grid sine wave cycles are required to achieve steady state. In this experimental setup, the grid was close to ideal, thus the negative and zero sequence components had near-zero amplitudes. However, this made it possible to verify that the estimation of the positive sequence component was correct, thus demonstrating that all components of the advanced grid synchronization algorithm are working properly and according to simulations.

Since the CPU is idling (or executing non-critical tasks) for approximately 90% of the time, this leaves enough time to execute the control of a power electronics converter. Hence, these results prove the feasibility of this technique for real-world applications which do not require frequencies significantly above 10 kHz.




(a) XMC board connected to a custom signal conditioning board and a display

(b) Three-phase transformer used

Figure 6.4: Experimental setup

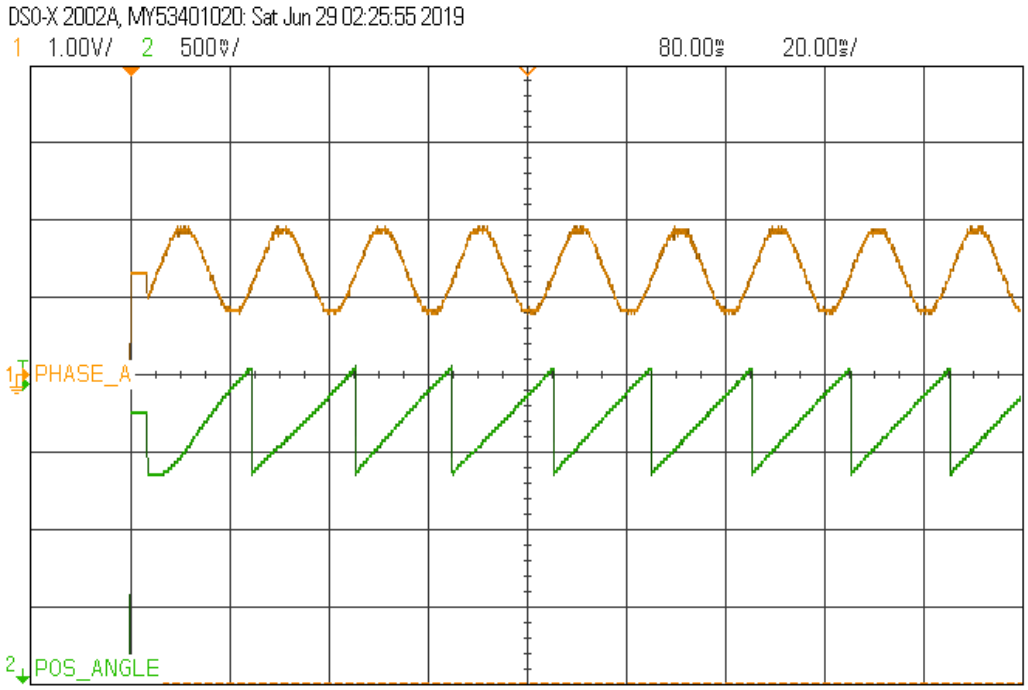



Figure 6.5: Positive sequence I-SRF-PLL angle estimation

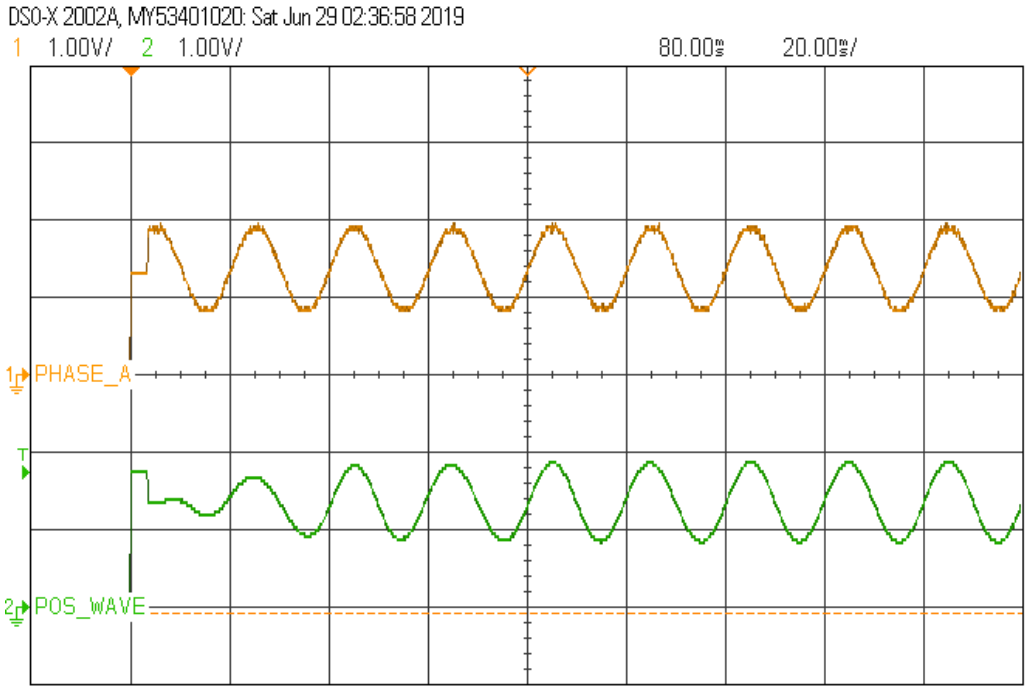



Figure 6.6: Recreated phase A sine wave

Despite the timestamps in the above figures, these measurements were **not** performed at 2 a.m.

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Chapter 7

# Optimization

## 7.1 Introduction

In previous chapters, the optimality of tuning parameters such as PI controller gains was never a concern as their values were picked by trial and error until they were "good enough" to carry the point of the explanations given. This chapter attempts to find tuning parameters that are better suited for real world applications.

## 7.2 Simulation Setup

In order to execute any optimization algorithm, it is necessary to setup a simulation environment from which a "score" that rates how well a given set of tuning parameters behaves can be obtained. In this context, a set of tuning parameters is known as a solution, and the "score" of each solution is known as its fitness.

The simulation accepts as input the individual amplitudes and phase shifts of each phase, and using the novel equations 5.15 and 5.18 it generates the true amplitudes and angles of the positive, negative and zero sequence components. These are of crucial importance to compute the fitness of each solution, as they can be compared with the estimations from the I-SRF-PLLs.

Optimizing parameters for a perfect starting condition can lead to a situation where an I-SRF-PLL can only start when given an ideal start. To ensure that the starting conditions are far from ideal, the initial phase of the grid angle can be configured. The grid frequency is also configurable and can vary during simulation, ensuring the optimization algorithm does not overfit the tuning parameters for a single grid frequency and also finds values that can properly compensate for frequency variations. The simulation features a configurable phase jump to test the responsiveness of the FFF mechanism. The generation of symmetrical sequence components is implemented via a custom function block as seen in Figure 7.1, with additional blocks to implement the phase jump.

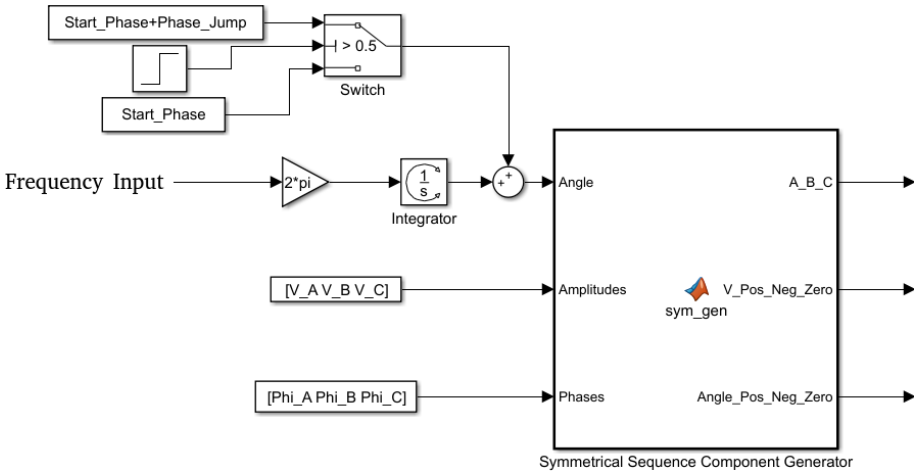



Figure 7.1: Generation of symmetrical sequence components in simulated environment

DC offset, noise and positive sequence harmonics can also be added to the input of the grid synchronization algorithm as demonstrated in Figure 7.2.

While in previous chapters placing the FFF after the cascaded QSG-SOGIs was the only option presented, it is actually unclear whether the FFF should be placed before the filtering or after the cascaded QSG-SOGIs. Placing it after the filters has the advantage that harmonics and noise will not affect the performance of the FFF as much, but has the disadvantage that if the QSG-SOGIs' input frequency does not match the grid frequency then the filtered alpha and beta signals will have different amplitudes, as shown in Figure 5.4. Placing the FFF before the filters ensures that the frequency estimation is not affected by a mismatch between the estimated frequency and the true grid frequency, but leaves it more susceptible to noise and harmonics. To allow the optimization algorithm to choose which one is best, two FFF mechanisms are included in the simulation (Figure 7.3), and a variable controls which one is used (Figure 7.4).

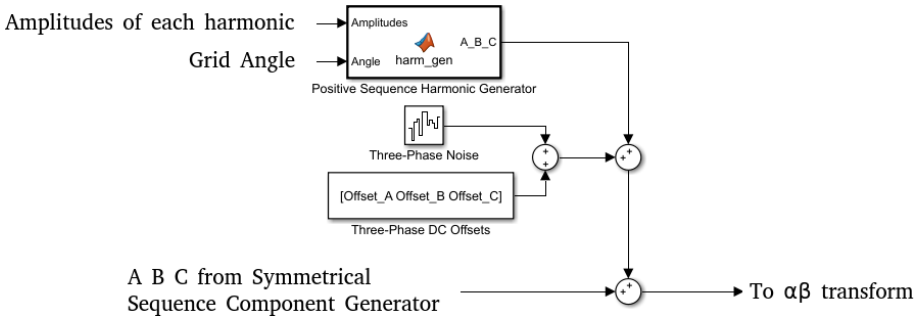



Figure 7.2: Adding DC offset, noise and harmonics to the simulated three-phase voltages

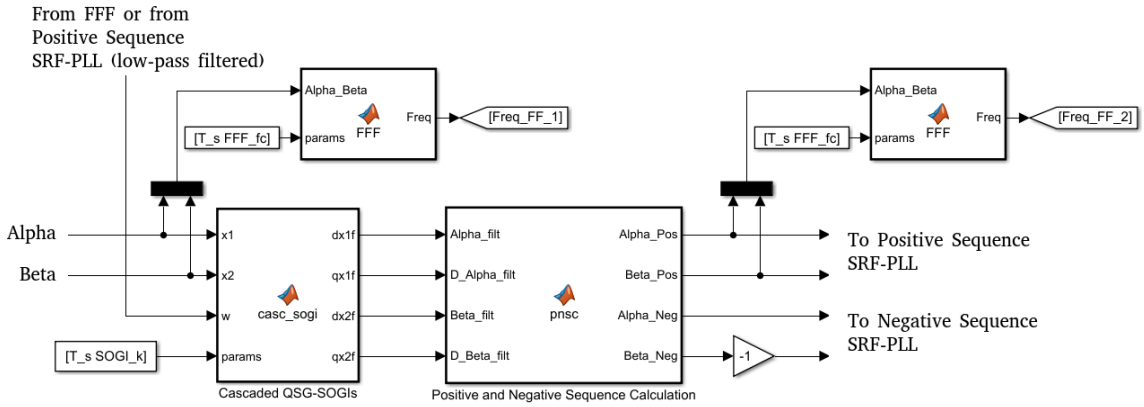



Figure 7.3: Simulation with two FFF mechanisms placed before and after the cascaded QSG-SOGI filters

Additionally, it is unclear which frequency estimation, either from FFF mechanism or from the positive sequence I-SRF-PLL, is better for the cascaded QSG-SOGIs. As such, another variable selects the frequency estimation source for the cascaded QSG-SOGIs, shown in Figure 7.4. To avoid circular dependencies, a unit delay at the output of the LPF for the input frequency of the cascaded QSG-SOGIs is required if either the positive sequence I-SRF-PLL or the FFF placed after the filter are used.

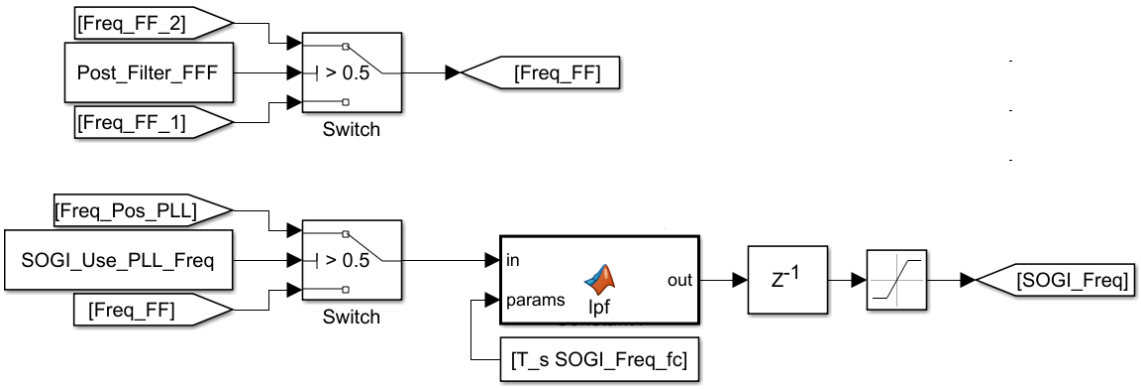



Figure 7.4: FFF placement selection and cascaded QSG-SOGI frequency input selection

In total, there are 18 simulation control variables and 19 tuning parameters. Table 7.1 summarizes the control variables, while Table 7.2 summarizes the tuning parameters to be optimized.

Note that the noise power in Table 7.1 is referring to the power spectral density of band-limited white-noise, with a correlation time of 10 µs.

Also note that Table 7.2 does not include tuning parameters for the zero sequence I-SRF-PLL as this PLL is not simulated. The reason to not simulate this I-SRF-PLL is that the effects of the zero sequence component can be nullified by delta-wye transformers. Since a delta connection provides no neutral point and since the zero sequence component is the exact same voltage across all phases, this component produces no potential difference in this type of transformer and hence can be ignored in most application scenarios. For this reason, this chapter purposefully does not include estimation of the zero sequence component, but the simulation could be easily adapted to do so.

Using different tuning parameters for the I-SRF-PLLs allows the optimization algorithm to find better tuning parameters for the negative sequence component by trading response time with noise immunity, as this component tends to be relatively small when compared to the positive sequence component and thus has a smaller signal-to-noise ratio.

Table 7.1: Simulation control variables

| Variable Name | Allowed Range | Explanation |
|---|---|---|
| T_s | 100 µs | Sampling period of grid synchronization algorithm |
| V_A | $\geq 0$ V | Amplitude of phase A |
| V_B | $\geq 0$ V | Amplitude of phase B |
| V_C | $\geq 0$ V | Amplitude of phase C |
| Phi_A | $\pm\infty$ ° | Phase shift of phase A |
| Phi_B | $\pm\infty$ ° | Phase shift of phase B |
| Phi_C | $\pm\infty$ ° | Phase shift of phase C |
| Offset_A | $\pm\infty$ V | DC offset of phase A |
| Offset_B | $\pm\infty$ V | DC offset of phase B |
| Offset_C | $\pm\infty$ V | DC offset of phase C |
| Noise_A | $\geq 0$ W/Hz | Height of power spectral density of phase A's noise |
| Noise_B | $\geq 0$ W/Hz | Height of power spectral density of phase B's noise |
| Noise_C | $\geq 0$ W/Hz | Height of power spectral density of phase C's noise |
| Start_Phase | $\pm\infty$ ° | Initial phase of the grid angle |
| Phase_Jump | $\pm\infty$ ° | Magnitude of phase jump |
| Phase_Jump_T | [0; Sim. Time] s | When the phase jump occurs |
| Signal_Freq | [40; 70] Hz | Grid frequency for each time-step |
| V_Harmonics | $\geq 0$ V | Array containing the amplitudes of all harmonics |

Table 7.2: Tuning parameters

| Variable Name | Allowed Range | Explanation |
|---|---|---|
| SOGI_k | $\geq 0$ | Tuning parameter of the QSG-SOGIs |
| SOGI_Freq_fc | $\geq 0\,\text{Hz}$ | Cutoff of QSG-SOGIs frequency input LPF |
| SOGI_Use_PLL_Freq | 0 or 1 | Frequency input: FFF (0) or PLL (1) |
| Post_Filter_FFF | 0 or 1 | FFF before QSG-SOGIs (0) or after (1) |
| FFF_fc | $\geq 0\,\text{Hz}$ | Cutoff of the FFF LPF |
| POS_ANS_Ampl_fc | $\geq 0\,\text{Hz}$ | Pos. Seq. PLL ANS Amplitude LPF cutoff |
| POS_ANS_Vq_fc | $\geq 0\,\text{Hz}$ | Pos. Seq. PLL ANS $v_q(t)$ LPF cutoff |
| POS_PI_kp | $\geq 0$ | Pos. Seq. PLL PI proportional gain |
| POS_PI_ki | $\geq 0$ | Pos. Seq. PLL PI integral gain |
| POS_PI_max_out | $\geq 0$ rad/s | Pos. Seq. PLL PI maximum output |
| POS_PI_min_out | $\leq 0$ rad/s | Pos. Seq. PLL PI minimum output |
| POS_PI_integral_limit | $\geq 0$ | Pos. Seq. PLL PI integral gain limit |
| NEG_ANS_Ampl_fc | $\geq 0\,\text{Hz}$ | Neg. Seq. PLL ANS Amplitude LPF cutoff |
| NEG_ANS_Vq_fc | $\geq 0\,\text{Hz}$ | Neg. Seq. PLL ANS $v_q(t)$ LPF cutoff |
| NEG_PI_kp | $\geq 0$ | Neg. Seq. PLL PI proportional gain |
| NEG_PI_ki | $\geq 0$ | Neg. Seq. PLL PI integral gain |
| NEG_PI_max_out | $\geq 0$ rad/s | Neg. Seq. PLL PI maximum output |
| NEG_PI_min_out | $\leq 0$ rad/s | Neg. Seq. PLL PI minimum output |
| NEG_PI_integral_limit | $\geq 0$ | Neg. Seq. PLL PI integral gain limit |

The specific values of the control variables used in this project can be found in Table 7.3. These were chosen such that they simulate a harsh but all-encompassing scenario. The amplitudes and phase shifts of the three-phase voltages are chosen such that the negative sequence component is of small amplitude when compared to the positive sequence component. Additionally, while the noise power used seems to be excessive, voltage sensors can be susceptible to switching noise from the power electronics, hence including an appropriate amount of noise can help the optimization algorithm in finding tuning parameters with good noise immunity. Double and triple frequency positive sequence harmonics are introduced to further increase the need for proper filter tuning.

As can be seen in Figure 7.5, the frequency starts at 50 Hz and increases to 60 Hz in a span of 50 ms. This frequency is maintained for 50 ms and then decreases to 50 Hz, again in a span of 50 ms. Furthermore, a phase jump of 30 ° is introduced after 0.1 s of simulated time. Together, these frequency and phase variations test the frequency adaptability of the grid synchronization technique.

Table 7.3: Values of the simulation control variables used

| Variable Name | Value |
|---|---|
| T_s | 100 µs |
| V_A | 55 V |
| V_B | 50 V |
| V_C | 45 V |
| Phi_A | 0 ° |
| Phi_B | −125 ° |
| Phi_C | 120 ° |
| Offset_A | 5 V |
| Offset_B | −2 V |
| Offset_C | 1 V |
| Noise_A | 0.1 mW/Hz |
| Noise_B | 0.1 mW/Hz |
| Noise_C | 0.1 mW/Hz |
| Start_Phase | 100 ° |
| Phase_Jump | 30 ° |
| Phase_Jump_T | 0.1 s |
| Signal_Freq | See Figure 7.5 |
| V_Harmonics | [1.0 0.5] V |

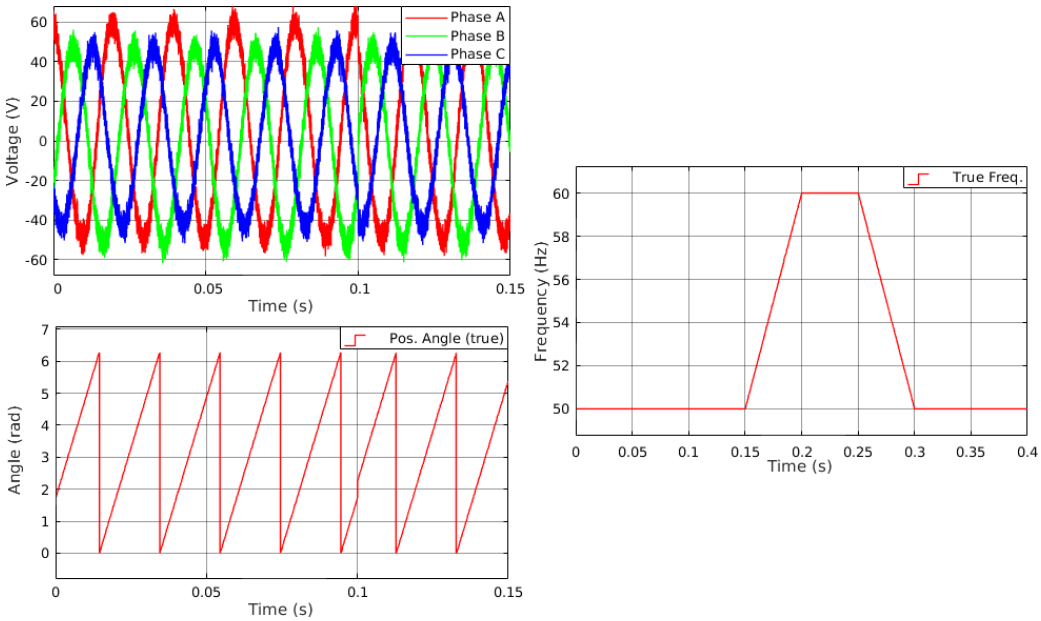



Figure 7.5: Simulated scenario

## 7.3 Optimization via Genetic Algorithm

### 7.3.1 Overview of Genetic Algorithms

Genetic algorithms (GAs) are optimization algorithms based on biologic evolution, effectively modeling the principles of genetics and natural selection[47].

A GA starts with an initial population, where each individual in the population is a solution - a set of variables to be optimized. In each iteration of a GA, the fitness (or its inverse, the cost) of each individual is calculated. The population goes through a natural selection process, where a part of the population "dies" to make room for new offspring. The survivors are selected for cross-breeding based on their fitness until the population is replenished back to its original size, which means that not all survivors might get a chance to reproduce. Finally, random mutations are introduced to allow the emergence of new traits that were not present in the original population. Usually, different populations are denoted by their generation number.

The stop conditions for GAs can vary wildly. These can be the number of stall generations (i.e. consecutive generations that show no improvement), a fixed number of generations or even a certain fitness threshold.

GAs are well suited for multi-variable problems with complex fitness functions that can not be solved by traditional algorithms. They do not require derivative information, work on both discrete and continuous variables, simultaneously search multiple regions of the fitness function and therefore are less prone to get stuck on local maxima[47].

GAs can treat variables either as encoded binary strings or as continuous variables where numbers are represented using floating-point and hence are known to the full machine precision. GAs that use continuous variables tend to be slightly faster because they do not need to decode the binary strings in order to compute the fitness of the individuals.

### 7.3.2 Custom Genetic Algorithm

For this project, a custom GA using continuous variables was developed. As with any GA, it starts by configuring the allowed range for the variables to be optimized. However, these are not necessarily the same ranges as the ones from Table 7.2. For instance, consider the example of the tuning parameter SOGI_k. While it is true that any positive value is valid, most values of interest are particularly close to zero as demonstrated in Chapter 5. Similarly, the various LPFs present in the grid synchronization algorithm are useless if their cutoff frequencies are several orders of magnitude above the grid frequency.

In order to reduce the search space, the ranges of the tuning parameters are constrained. These constraints are shown in Table 7.4. Additionally, the minimum output for each PI controller is always set to the negative of its maximum output value, thus reducing the number of variables to optimize.

It should be noted that for boolean variables such as SOGI_Use_PLL_Freq the GA is allowed to generate any value between 0 and 1, which are then rounded for simulation.

Algorithm 5 presents a high-level overview of the custom GA. This custom GA implements additional features known as elitism, tournament selection and heuristic crossover.

In a elitist GA, the equally best solutions from each generation are propagated to the next generation completely unchanged. In the case of the custom GA developed for this project only one of the best solutions is propagated, as shown in line 16 of Algorithm 5.

Algorithm 6 demonstrates the process of tournament selection. By picking 3 random individuals from the population and selecting the best one as a parent, this process mimics the mating competitions that occur in nature [47].

Algorithm 7 shows how the heuristic crossover is performed. A value of $\beta$ decides how much the offspring inherits from each parent. While $\beta$ typically varies between 0 and 1, in this GA it is allowed to vary between -0.2 and 1.2 to allow the crossbreeding process to introduce new genetic information. This circumvents the problem with continuous GAs where simpler crossover algorithms are used, which rely solely on mutations to introduce new genetic material[47].

Table 7.4: Constraints for each GA variable

| Variable Name | Constraints |
|---|---|
| SOGI_k | [0; 5] |
| SOGI_Freq_fc | $[0; 500]$ Hz |
| SOGI_Use_PLL_Freq | [0; 1] |
| Post_Filter_FFF | [0; 1] |
| FFF_fc | $[0; 500]$ Hz |
| POS_ANS_Ampl_fc | $[0; 500]$ Hz |
| POS_ANS_Vq_fc | $[0; 500]$ Hz |
| POS_PI_kp | $[0; 10^6]$ |
| POS_PI_ki | $[0; 10^6]$ |
| POS_PI_max_out | $[0; 10^6]$ rad/s |
| POS_PI_min_out | -POS_PI_max_out |
| POS_PI_integral_limit | $[0; 10^6]$ |
| NEG_ANS_Ampl_fc | $[0; 500]$ Hz |
| NEG_ANS_Vq_fc | $[0; 500]$ Hz |
| NEG_PI_kp | $[0; 10^3]$ |
| NEG_PI_ki | $[0; 10^3]$ |
| NEG_PI_max_out | $[0; 10^6]$ rad/s |
| NEG_PI_min_out | -NEG_PI_max_out |
| NEG_PI_integral_limit | $[0; 10^6]$ |

**Algorithm 5** Custom GA Algorithm

**Input:** $N\_pop$, $N\_keep$, $mutation\_probability$

Generate random *population* with $N\_pop$ individuals
**while** $optimization\_complete \neq true$ **do**
  **for each** *individual* in *population* **do**
    Compute $fitness$ of *individual*
  **end for**
  Sort *population* by $fitness$ in descending order
  Keep first $N\_keep$ individuals in *population*
  **while** size(*population*) $< N\_pop$ **do**
    $parent\_a$ = tournamentSelection(*population*, $fitness$)
    $parent\_b$ = tournamentSelection(*population*, $fitness$)
    $population(\text{size}(population + 1))$ = crossBreed($parent_a$ ,$parent_b$);
  **end while**
  **if** Stop conditions are met **then**
    $optimization\_complete = true$
  **else**
    **for** $i = 2;\ i < N\_pop;\ i = i + 1$ **do**
      $population(i)$ = mutate($population(i)$, $mutation\_probability$)
    **end for**
  **end if**
**end while**

**Algorithm 6** Tournament Selection

**Input:** *population*, $fitness$

**for** $i = 1;\ i \leq 3;\ i = i + 1$ **do**
  $candidate(i) = population(\text{random}(1, \text{size}(population)))$
**end for**
**return** *candidate* with highest $fitness$

**Algorithm 7** Crossbreeding via Heuristic Crossover

**Input:** $parent\_a$, $parent\_b$

$offspring$ = emptyIndividual()
**for** $i = 1;\ i \leq size(offspring);\ i = i + 1$ **do**
  $\beta$ = random(-0.1, 1.1)
  $offspring(i) = \beta \cdot parent\_a + (1 - \beta) \cdot parent\_b$
  Limit $offspring(i)$ variable according to its constraints from Table 7.4
**end for**
**return** $offspring$

Finally, the mutation process of the custom GA is a standard variable randomization which occurs based on a fixed probability.

**Algorithm 8** Mutation

**Input:** *individual*

1: **for each** *variable* in *individual* **do**
2: **if** random(0,1) $\leq$ *mutation_probability* **then**
3: *variable* = random(*lower_bound*, *upper_bound*)
4: **end if**
5: **end for**
6: **return** *individual*

### 7.3.3 Fitness Function

The selection of a proper fitness function is vital to the success of any optimization algorithm. While it is obvious that the error of the positive and negative sequence component must play a central role in computing the fitness of an individual, how to do so may not be as self-evident.

Consider the following hypothetical fitness function which only considers a single error signal, which is a $N$ sized array containing the simulated errors at each time-step for a given solution.

$$fitness = \frac{1}{\sum_{i=1}^{N} |error_i|} \tag{7.1}$$

The reasoning behind this fitness function is quite simple: since the objective is to minimize the error, simply sum up all the errors in each time-step, and the solution with the lowest cumulative error is considered to be the best. The problem with this simplistic approach is that this fitness function fails to place the proper emphasis on the steady-state behaviour and disproportionately favours solutions with shorter transient times.

In Figure 7.6, two hypothetical error signals from two solutions are generated. Solution A generates a error signal which starts at 1 and smoothly decays to zero in an exponential fashion. Solution B also starts at 1, but decays slightly faster than solution A, albeit with Gaussian noise added to it. This hypothetical simulation has a time-step of 1 ms.

Using the fitness function 7.1, the fitness of solution A is 0.0100 and the fitness of solution B is 0.0103. From this, it is clear that even a tiny decrease in transient time is considered "better" over adding a significant amount of noise.

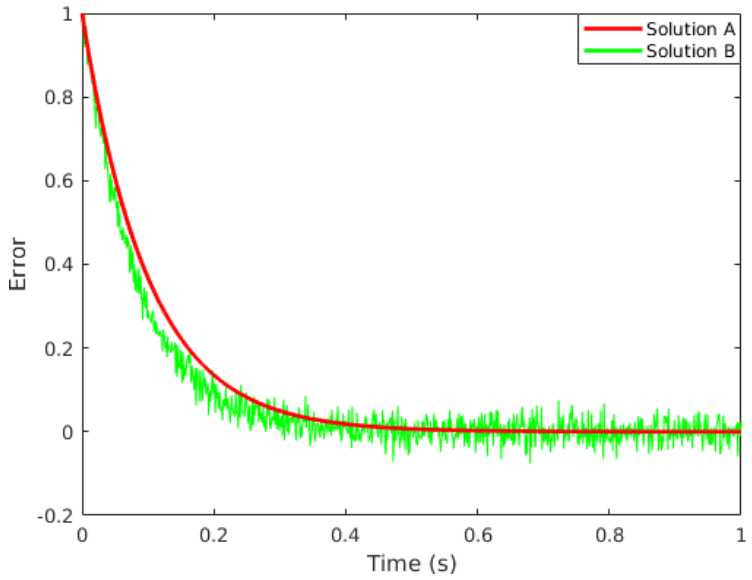



Figure 7.6: Error signals of two hypothetical solutions for fitness comparison

Intuitively, the results of solution A are much better than the results of solution B. However, this intuition is often difficult to express in mathematical terms. For instance, one might think that perhaps the root-mean-square (RMS) provides more accurate results:

$$fitness = \frac{1}{\sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} error_i^2}} \tag{7.2}$$

However, using equation 7.2 for the exact same hypothetical scenario, the fitness of solution A is 4.45 while solution B is 4.83. The RMS-based fitness function still favours a marginally faster response time at the expense of excessive noise.

While the examples from 7.6 are relatively extreme and hand-crafted to exacerbate the problem, it is not difficult to conceive that with such a big number of variables and a huge search space, many "noisy" solutions that provide quick response times at the expense of noise immunity exist. Hence, it is important to select a proper fitness function that correctly balances the importance of the error during transients and the error during steady state.

A relatively easy solution to the fitness problem is revert back to the cumulative error fitness function from equation 7.1, but now weighting each individual error value:

$$fitness = \frac{1}{\sum_{i=1}^{N} |w_i \cdot error_i|} \tag{7.3}$$

Choosing values for these weights can be done in a variety of different ways. For this project, the weights start at zero and increase by a fixed amount each time-step, but are reset at the instant a transient is introduced in the simulation. This places more emphasis on the error during steady state while reducing the importance of the error during transients.

In the hypothetical scenario of Figure 7.6 the only transient is at the start, hence the weights would simply increase over time without ever resetting. Using this technique with a 0.1 increment in weight per time-step, the fitness of solutions A and B become 0.001 and 0.0007, respectively.

For the actual simulation of the grid synchronization algorithm, since the fitness function must evaluate the performance of both positive and negative sequence errors, then they can be combined in a single equation:

$$fitness = \frac{1}{\sqrt{\left(\sum_{i=1}^{N} |w_i \cdot error\_pos_i|\right)^2 + \left(\sum_{i=1}^{N} |w_i \cdot error\_neg_i|\right)^2}} \tag{7.4}$$

The above equation simply takes two cost functions - the errors of the positive and negative sequence components - and combines them using the standard non-preferential method[48]. This avoids adding the increased layer of complexity of multi-objective optimization by effectively collapsing two objectives into one - a single fitness to be optimized.

In the simulated scenario, disturbances are introduced at the moments $t = 0.1$ s, $t = 0.15$ s, $t = 0.2$ s, $t = 0.25$ s and $t = 0.3$ s as shown in Figure 7.5. Hence, Figure 7.7 shows how the weights are reset in these moments.

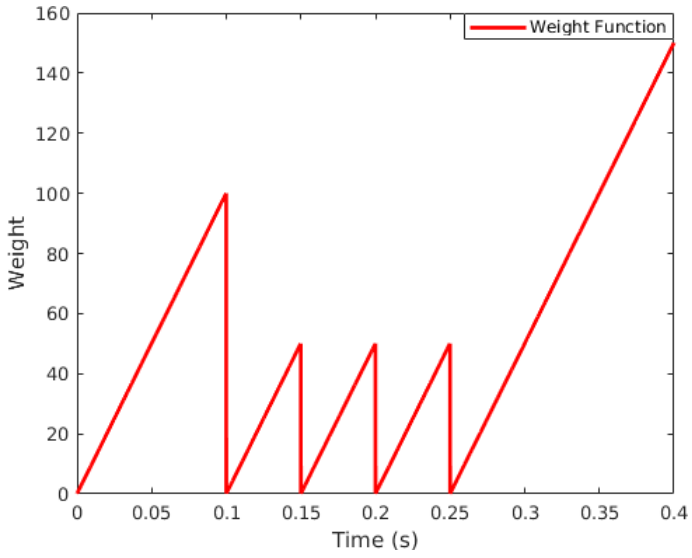



Figure 7.7: Weights for the simulated scenario

### 7.3.4 Optimization Results

Using the fitness function from equation 7.4, the custom genetic algorithm previously described was run for 100 generations, with a population size of 200 where only 100 individuals survive per iteration and a mutation probability of 50%.

In order to have a basis of comparison, the fitness of the parameters previously used in simulations in other chapters was calculated. These parameters are shown in Table 7.5, and the simulated results are shown in Figure 7.8. Similarly, the parameters optimized by the GA are also shown in Table 7.5 and the results in Figure 7.9.

As can be seen in Table 7.5, the GA luckily agrees on the original architecture proposed on previous chapters, using the frequency of the positive sequence I-SRF-PLL as the input to the cascaded QSG-SOGIs and also placing the FFF after them.

Interestingly, it seems that the integral limit functionality has little to no impact on the performance, as the GA optimized these values to be so high that they do not interfere with the normal operation of the PI controllers. Hence, it is possible that this feature is not very well suited for a grid synchronization algorithm.

Table 7.5: Tuning parameters picked by trial and error and optimized by the GA

| Variable Name | Trial and error | GA optimized |
|---|---|---|
| SOGI_k | 3 | 1.947 |
| SOGI_Freq_fc | 10 Hz | 23.927 Hz |
| SOGI_Use_PLL_Freq | 1 | 1 |
| Post_Filter_FFF | 1 | 1 |
| FFF_fc | 50 Hz | 371.987 Hz |
| POS_ANS_Ampl_fc | 100 Hz | 256.343 Hz |
| POS_ANS_Vq_fc | 100 Hz | 104.728 Hz |
| POS_PI_kp | 500 | 62712.771 |
| POS_PI_ki | 5000 | 26821.432 |
| POS_PI_max_out | $2\pi 100$ rad/s | 68661.008 rad/s |
| POS_PI_min_out | $-2\pi 100$ | -68661.008 rad/s |
| POS_PI_integral_limit | $+\infty$ | 60333.621 |
| NEG_ANS_Ampl_fc | 100 Hz | 284.626 Hz |
| NEG_ANS_Vq_fc | 100 Hz | 471.541 Hz |
| NEG_PI_kp | 100 | 203.583 |
| NEG_PI_ki | 400 | 350.807 |
| NEG_PI_max_out | $2\pi 100$ rad/s | 46777.475 rad/s |
| NEG_PI_min_out | $-2\pi 100$ rad/s | -46777.475 rad/s |
| NEG_PI_integral_limit | $+\infty$ | 59492.315 |

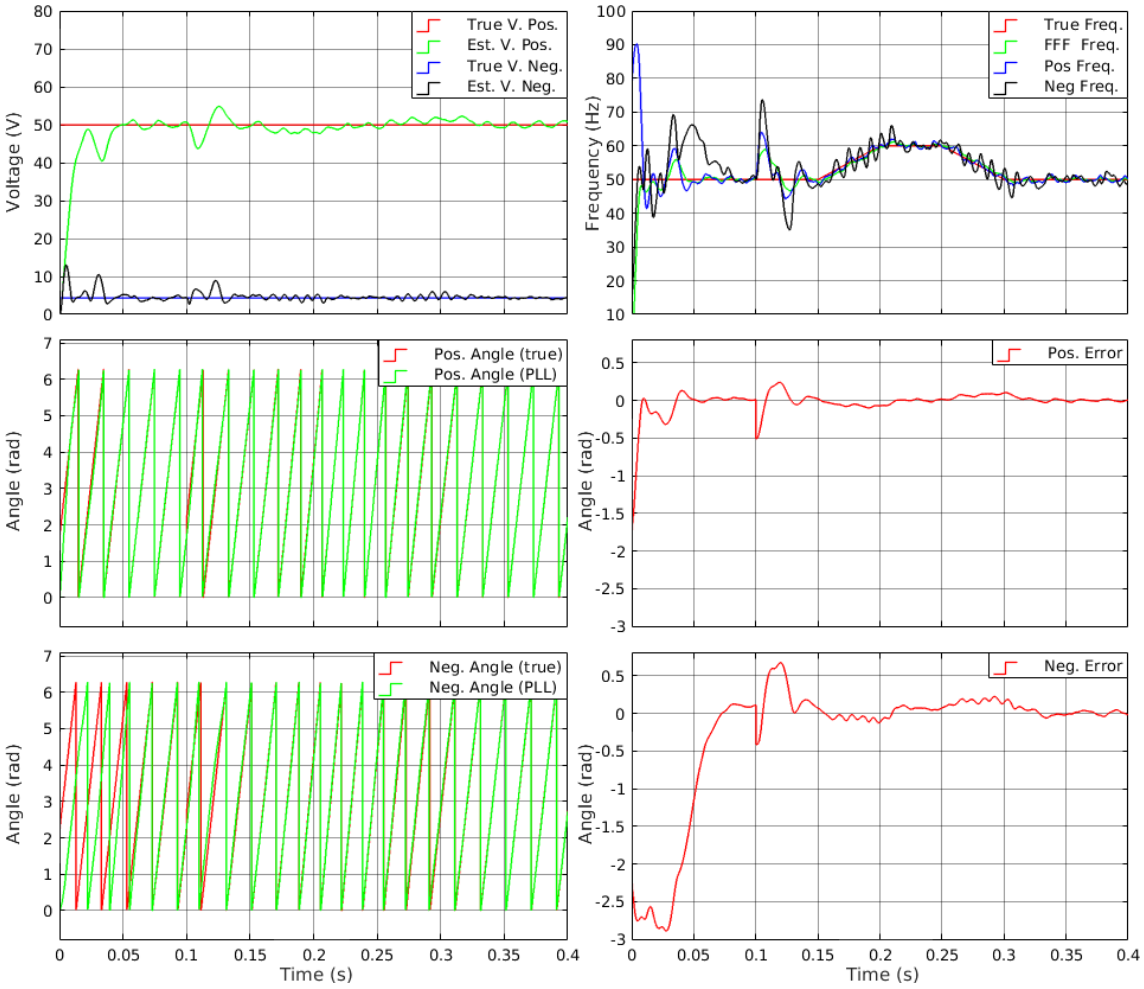



Figure 7.8: Results of tuning parameters picked by trial and error

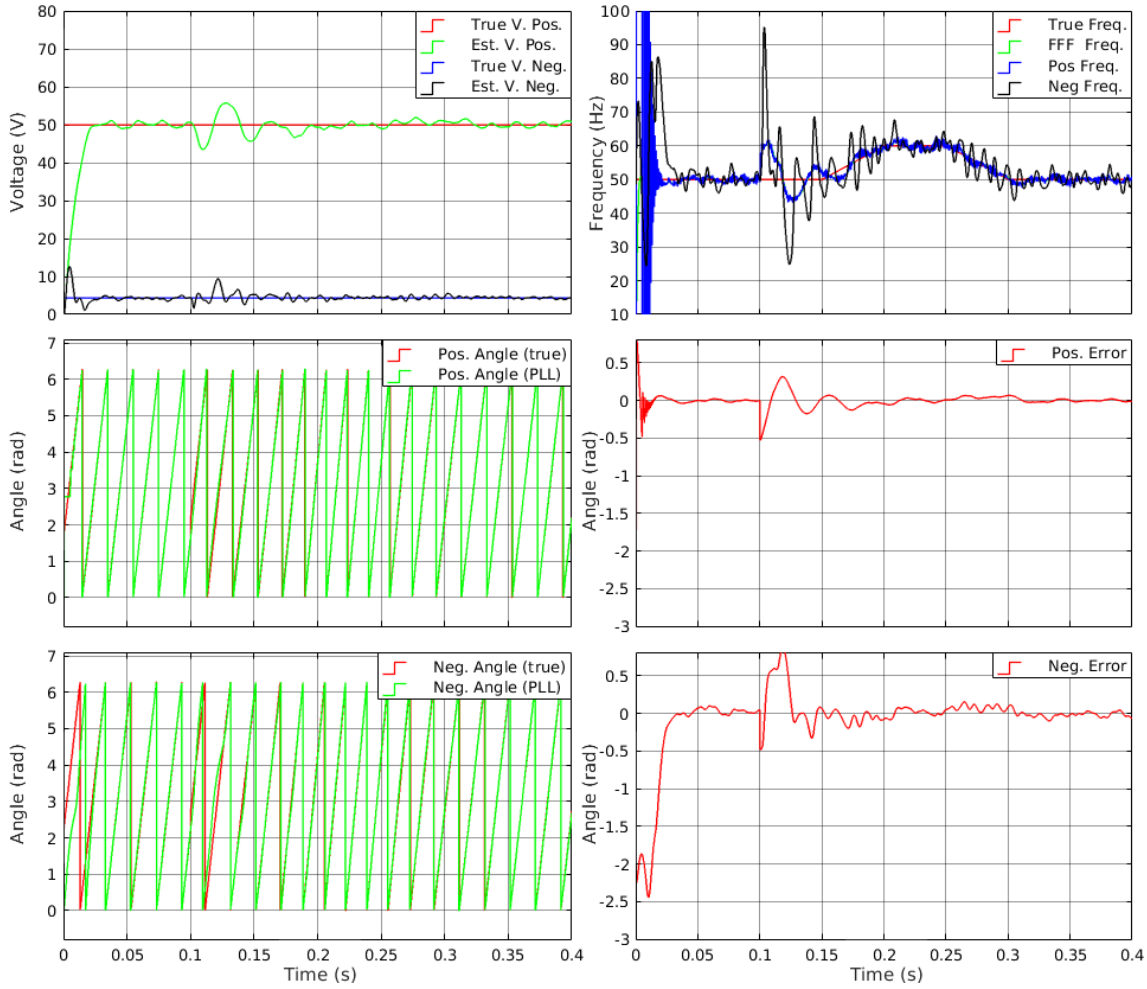



Figure 7.9: Results of tuning parameters optimized by the GA

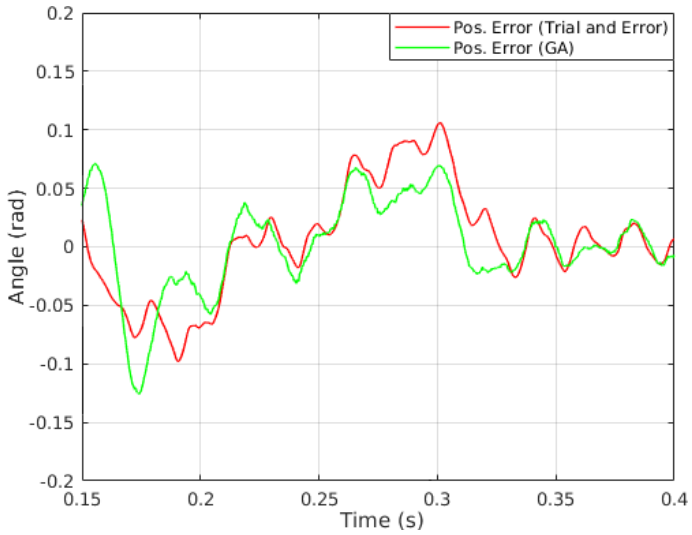



Figure 7.10: Comparison between positive sequence errors during frequency variations and during steady-state conditions

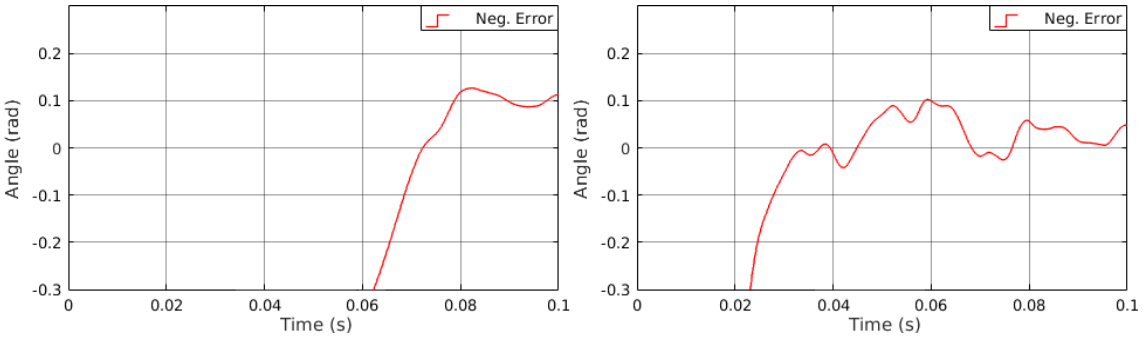



Figure 7.11: Comparison between negative sequence start-up errors of the parameters picked by hand (left) and the GA-optimized parameters (right)

The error of the positive sequence component's angle that the GA optimized for is very similar to the error obtained by the values picked by trial and error. However, it has achieved this by having high PI controller gains and a smaller SOGI_k value. This implies that the GA-optimized solution has higher noise immunity. In Figure 7.10, the comparison of the errors during the frequency variations and steady state is shown in detail to demonstrate how close the results are.

The estimation of the negative sequence component's angle by the GA-optimized parameters is slightly more sensitive to the phase jump, resulting in higher overshoot and undershoot during the moments $t = 0.1$ s and $t = 0.15$ s. It also tends to oscillate slightly more during the frequency variations, but the overall error is closer to zero.

It should also be noted that the GA-optimized solution is much quicker at locking-on, approximately 25 ms. By comparison, the hand-picked solution takes 3 times longer and does not even settle to zero error before the frequency variations at $t = 0.1$ s. This is quite apparent in Figure 7.11.

As can be seen in Figures 7.8 and 7.9, the GA does not "care" about high-amplitude variations in the estimated frequency as long as the overall error is lower. This is to be expected, as the fitness function does not evaluate the correctness of the estimated frequency. However, the frequency estimation of the positive sequence component is somewhat close to the true grid frequency, at least after the effects of the phase jump disappear.

The fitness of the parameters picked by trial and error is 0.000022 while the fitness of the GA-optimized parameters is 0.000068, approximately a threefold increase. In conclusion, the optimization via the custom genetic algorithm provided acceptable results. Nonetheless, it should be noted that while this demonstrates that this optimization technique is viable for this particular grid synchronization algorithm, it does not necessarily mean that these parameters are the best for every single case. A system designer must carefully simulate an environment that is representative of the real-world application in which the algorithm will be used.

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Chapter 8

# Conclusion and Future Work

## 8.1 Conclusion

This project aimed to develop an advanced grid synchronization algorithm. While some improvements were relatively easy to implement, others required a more careful analysis of the theory behind symmetrical sequence components. In doing so, novel equations that relate the parameters of unbalanced three-phase voltages to the parameters of their symmetrical sequence components were derived.

These equations proved useful not only for analysis but also for tuning parameters via a genetic algorithm. With these equations, a system designer can simulate a truly unbalanced system - where neither the amplitudes and phases are balanced - and extract the positive, negative and zero sequence components' angles. From these angles, one can now compare the errors between the estimated and true angles, and compute a fitness based on said errors.

The proposed advanced grid synchronization algorithm was also implemented in a modern microcontroller, proving that the implementation in a real-world product is feasible. Some approximations were used in order to speed up some of the most expensive calculations.

Overall, all goals of the project were accomplished. In doing so, this document now (hopefully) serves the purpose of presenting future students some of the methodologies usually employed in developing a grid synchronization algorithm. Even if said algorithms are different in nature - like algorithms based on the DFT or Kalman filters - the analysis of the $\alpha\beta$ and DQ0 transforms, symmetrical sequence components and the techniques used for parameters optimization still hold value.

## 8.2 Future Work

The following list provides some ideas and guidance for future derivative works:

- Use the novel equations relating unbalanced three-phase voltages to their symmetrical sequence components to implement other synchronization algorithms.
  - E.g. by integrating said equations into a Kalman filter.
  - E.g. by using single-phase techniques to estimate the amplitude and frequency of each phase, and then estimating the amplitudes and angles of each component via said equations
- Optimize for different scenarios
  - E.g. optimize the algorithm for use in motor control applications
  - E.g. optimize the algorithm to cope with grid failures
- Quantify the impact of the approximations used in the implementation
- Benchmark performance on FPGAs
  - Many parts of the algorithm can be adapted to perform in parallel, hence it is likely that even a low-end FPGA could provide significant performance improvements over a microcontroller
- Perform more strict performance tests according to international standards
- Compare with other synchronization algorithms

# Appendix A

# Proof that Sum of Balanced Three-Phase Voltages Is Zero

The sum of a set of balanced three-phase voltages is given by:

$$V \cdot \sin(\theta) + V \cdot \sin\left(\theta - \frac{2\pi}{3}\right) + V \cdot \sin\left(\theta + \frac{2\pi}{3}\right) \tag{A.1}$$

Applying the trigonometric rule[45]: $\sin(a \pm b) = \sin(a)\cos(b) \pm \sin(b)\cos(a)$:

$$\begin{aligned} V \cdot \sin(\theta) + V \cdot \sin(\theta) \cdot \cos\left(\frac{2\pi}{3}\right) - V \cdot \cos(\theta) \cdot \sin\left(\frac{2\pi}{3}\right) + V \cdot \sin(\theta) \cdot \cos\left(\frac{2\pi}{3}\right) \\ + V \cdot \cos(\theta) \cdot \sin\left(\frac{2\pi}{3}\right) \end{aligned} \tag{A.2}$$

By eliminating opposites and simplifying known trigonometric values:

$$\begin{aligned} & V \cdot \sin(\theta) - V \cdot \sin(\theta) \cdot \frac{1}{2} - V \cdot \sin(\theta) \cdot \frac{1}{2} \\ = & V \cdot \sin(\theta) - V \cdot \sin(\theta) = 0 \end{aligned} \tag{A.3}$$

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Appendix B

# Proof of Space Vector's Constant Magnitude

The norm of any given vector $\vec{X} = (x_1, x_2, ..., x_n)$ is given by[49]:

$$\|X\| = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{B.1}$$

The space vector $\vec{S}$, under balanced grid conditions, is given by:

$$\vec{S} = \left( V \cdot \sin(\theta), V \cdot \sin\left(\theta - \frac{2\pi}{3}\right), V \cdot \sin\left(\theta + \frac{2\pi}{3}\right)\right) \tag{B.2}$$

The norm of the space vector $\vec{S}$ is given by:

$$\|S\| = \sqrt{V^2 \cdot \sin(\theta)^2 + V^2 \cdot \sin\left(\theta - \frac{2\pi}{3}\right)^2 + V^2 \cdot \sin\left(\theta + \frac{2\pi}{3}\right)^2} \tag{B.3}$$

$$= \sqrt{V^2 \cdot \left(\sin(\theta)^2 + \sin\left(\theta - \frac{2\pi}{3}\right)^2 + \sin\left(\theta + \frac{2\pi}{3}\right)^2\right)} \tag{B.4}$$

$$= \sqrt{V^2} \cdot \sqrt{\sin(\theta)^2 + \sin\left(\theta - \frac{2\pi}{3}\right)^2 + \sin\left(\theta + \frac{2\pi}{3}\right)^2} \tag{B.5}$$

Applying the trigonometric rule[45]: $\sin(a \pm b) = \sin(a) \cdot \cos(b) \pm \sin(b) \cdot \cos(a)$ to equation B.5:

$$\|S\| = V \cdot \sqrt{\sin(\theta)^2 + \left(\sin(\theta) \cdot \cos\left(\frac{2\pi}{3}\right) - \cos(\theta) \cdot \sin\left(\frac{2\pi}{3}\right)\right)^2 + \left(\sin(\theta) \cdot \cos\left(\frac{2\pi}{3}\right) + \cos(\theta) \cdot \sin\left(\frac{2\pi}{3}\right)\right)^2} \tag{B.6}$$

Computing the known trigonometric values:

$$\|S\| = V \cdot \sqrt{\sin(\theta)^2 + \left(-\sin(\theta) \cdot \frac{1}{2} - \cos(\theta) \cdot \frac{\sqrt{3}}{2}\right)^2 + \left(-\sin(\theta) \cdot \frac{1}{2} + \cos(\theta) \cdot \frac{\sqrt{3}}{2}\right)^2} \tag{B.7}$$

Further simplification leads to:

$$\|S\| = V \cdot \sqrt{\sin(\theta)^2 + \frac{\left(-\sin(\theta) - \cos(\theta) \cdot \sqrt{3}\right)^2}{4} + \frac{\left(-\sin(\theta) + \cos(\theta) \cdot \sqrt{3}\right)^2}{4}} \tag{B.8}$$

$$= V \cdot \sqrt{\frac{4 \cdot \sin(\theta)^2 + \left(-\sin(\theta) - \cos(\theta) \cdot \sqrt{3}\right)^2 + \left(-\sin(\theta) + \cos(\theta) \cdot \sqrt{3}\right)^2}{4}} \tag{B.9}$$

Expansion of the squared bracketed expressions results in:

$$\|S\| = V \cdot \sqrt{\frac{4 \cdot \sin(\theta)^2 + \sin(\theta)^2 + 2 \cdot \sin(\theta) \cdot \sqrt{3} \cdot \cos(\theta) + 3 \cdot \cos(\theta)^2}{4} \frac{+ \sin(\theta)^2 - 2 \cdot \sin(\theta) \cdot \sqrt{3} \cdot \cos(\theta) + 3 \cdot \cos(\theta)^2}{4}} \tag{B.10}$$

Eliminating opposites and combining like terms yields:

$$\|S\| = V \cdot \sqrt{\frac{6 \cdot \sin(\theta)^2 + 6 \cdot \cos(\theta)^2}{4}} \tag{B.11}$$

Using the trigonometric rule[50]: $\sin(a)^2 + \cos(a)^2 = 1$, it follows that:

$$\begin{aligned} \|S\| &= V \cdot \sqrt{\frac{6}{4}} \\ &= V \cdot \sqrt{\frac{3}{2}} \end{aligned} \tag{B.12}$$

# Appendix C

# Proof of Trigonometric Relationship

Consider the following trigonometric rule[51]:

$$\sin(\alpha) + \sin(\beta) = 2 \cdot \sin\left(\frac{1}{2} \cdot (\alpha + \beta)\right) \cdot \cos\left(\frac{1}{2} \cdot (\alpha - \beta)\right) \tag{C.1}$$

Let $a$ and $b$ be defined as:

$$a = \frac{1}{2} \cdot (\alpha + \beta) \tag{C.2}$$

$$a - b = \frac{1}{2} \cdot (\alpha - \beta) \tag{C.3}$$

Solving for $\alpha$ and $\beta$:

$$\alpha = 2 \cdot a - b \tag{C.4}$$

$$\beta = b \tag{C.5}$$

Substituting the above in equation C.1:

$$\sin(2 \cdot a - b) + \sin(b) = 2 \cdot \sin(a) \cdot \cos(a - b) \tag{C.6}$$

Rearranging yields:

$$\cos(a - b) \cdot \sin(a) = \frac{1}{2} \cdot (\sin(2 \cdot a - b) + \sin(b)) \tag{C.7}$$

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Appendix D

# Simplification of $\alpha\beta$ Symmetrical Components

This appendix focuses on the simplification of the following equations (from equations 5.22 in Chapter 5):

$$\begin{cases} v_\alpha^+(t) &= \frac{2}{3} \cdot \left( v_a^+(t) - \frac{v_b^+(t)}{2} - \frac{v_c^+(t)}{2} \right) \\ v_\alpha^-(t) &= \frac{2}{3} \cdot \left( v_a^-(t) - \frac{v_b^-(t)}{2} - \frac{v_c^-(t)}{2} \right) \\ v_\beta^+(t) &= \frac{2}{3} \cdot \left( \frac{\sqrt{3}}{2} \cdot v_b^+(t) - \frac{\sqrt{3}}{2} \cdot v_c^+(t) \right) \\ v_\beta^-(t) &= \frac{2}{3} \cdot \left( \frac{\sqrt{3}}{2} \cdot v_b^-(t) - \frac{\sqrt{3}}{2} \cdot v_c^-(t) \right) \end{cases} \tag{D.1}$$

## D.1 $\alpha$ Symmetrical Components

Notice how similar $v_\alpha^+$ and $v_\alpha^-$ are. For compactness, the following equation represents both:

$$v_\alpha^\pm(t) = \frac{2}{3} \cdot V^\pm \cdot \left( \sin\left(\theta(t) + \phi^\pm\right) - \frac{\sin\left(\theta(t) + \phi^\pm \mp \frac{2\pi}{3}\right)}{2} - \frac{\sin\left(\theta(t) + \phi^\pm \pm \frac{2\pi}{3}\right)}{2} \right) \tag{D.2}$$

To make this deduction easier, only the above bracketed expression is simplified. Applying the trigonometric rule[45] $\sin(a \pm b) = \sin(a)\cos(b) \pm \sin(b)\cos(a)$ yields:

$$\begin{aligned} \sin\left(\theta(t) + \phi^\pm\right) - \frac{\sin\left(\theta(t) + \phi^\pm\right) \cdot \cos\left(\frac{2\pi}{3}\right) \mp \sin\left(\frac{2\pi}{3}\right) \cdot \cos\left(\theta(t) + \phi^\pm\right)}{2} \\ - \frac{\sin\left(\theta(t) + \phi^\pm\right) \cdot \cos\left(\frac{2\pi}{3}\right) \pm \sin\left(\frac{2\pi}{3}\right) \cdot \cos\left(\theta(t) + \phi^\pm\right)}{2} \end{aligned} \tag{D.3}$$

Using the substitution $u = \theta(t) + \phi^\pm$:

$$\sin(u) - \frac{\sin(u) \cdot \cos\left(\frac{2\pi}{3}\right) \mp \sin\left(\frac{2\pi}{3}\right) \cdot \cos(u)}{2} - \frac{\sin(u) \cdot \cos\left(\frac{2\pi}{3}\right) \pm \sin\left(\frac{2\pi}{3}\right) \cdot \cos(u)}{2} \tag{D.4}$$

Combining both fractions:

$$\sin(u) + \frac{-\sin(u)\cdot\cos\left(\frac{2\pi}{3}\right) \pm \sin\left(\frac{2\pi}{3}\right)\cdot\cos(u) - \sin(u)\cdot\cos\left(\frac{2\pi}{3}\right) \mp \sin\left(\frac{2\pi}{3}\right)\cdot\cos(u)}{2} \tag{D.5}$$

Eliminating opposites and combining like terms:

$$\sin(u) + \frac{-2\cdot\sin(u)\cdot\cos\left(\frac{2\pi}{3}\right)}{2} \tag{D.6}$$

Computing known cosine value:

$$\sin(u) + \frac{\sin(u)}{2} \tag{D.7}$$

Simplifying and undoing the $u = \theta(t) + \phi^{\pm}$ substitution:

$$\frac{3}{2}\cdot\sin(\theta(t) + \phi^{\pm}) \tag{D.8}$$

Plugging back the above result into equation D.2:

$$v_{\alpha}^{\pm}(t) = \frac{2}{3}\cdot V^{\pm}\cdot\frac{3}{2}\cdot\sin(\theta(t) + \phi^{\pm}) = V^{\pm}\cdot\sin(\theta(t) + \phi^{\pm}) \tag{D.9}$$

## D.2 $\beta$ Symmetrical Components

As in the previous deduction, notice how similar $v_{\beta}^{+}$ and $v_{\beta}^{-}$ are. Again, for compactness, the following equation represents both:

$$v_{\beta}^{\pm}(t) = \frac{2}{3}\cdot V^{\pm}\cdot\left(\frac{\sqrt{3}}{2}\cdot\sin\left(\theta(t) + \phi^{\pm} \mp \frac{2\pi}{3}\right) - \frac{\sqrt{3}}{2}\cdot\sin\left(\theta(t) + \phi^{\pm} \pm \frac{2\pi}{3}\right)\right) \tag{D.10}$$

Bringing out the common term inside the bracketed expression:

$$v_{\beta}^{\pm}(t) = \frac{\sqrt{3}}{3}\cdot V^{\pm}\cdot\left(\sin\left(\theta(t) + \phi^{\pm} \mp \frac{2\pi}{3}\right) - \sin\left(\theta(t) + \phi^{\pm} \pm \frac{2\pi}{3}\right)\right) \tag{D.11}$$

Taking only the above bracketed expression and applying both the trigonometric rule $\sin(a \pm b) = \sin(a)\cos(b) \pm \sin(b)\cos(a)$ and the substitution $u = \theta(t) + \phi^{\pm}$ yields:

$$\sin(u)\cos\left(\frac{2\pi}{3}\right) \mp \sin\left(\frac{2\pi}{3}\right)\cos(u) - \left(\sin(u)\cos\left(\frac{2\pi}{3}\right) \pm \sin\left(\frac{2\pi}{3}\right)\cos(u)\right) \tag{D.12}$$

Simplifying the signs:

$$\sin(u)\cos\left(\frac{2\pi}{3}\right) \mp \sin\left(\frac{2\pi}{3}\right)\cos(u) - \sin(u)\cos\left(\frac{2\pi}{3}\right) \mp \sin\left(\frac{2\pi}{3}\right)\cos(u) \tag{D.13}$$

Eliminating opposites and combining like terms:

$$\mp 2\cdot\sin\left(\frac{2\pi}{3}\right)\cos(u) \tag{D.14}$$

Computing the know sine value and undoing the substitution $u = \theta(t) + \phi^{\pm}$:

$$\mp\sqrt{3} \cdot \cos\left(\theta(t) + \phi^{\pm}\right) \tag{D.15}$$

Plugging back this result into equation D.11:

$$v_{\beta}^{\pm}(t) = \frac{\sqrt{3}}{3} \cdot V^{\pm} \cdot \left(\mp\sqrt{3} \cdot \cos\left(\theta(t) + \phi^{\pm}\right)\right) = \mp V^{\pm} \cdot \cos\left(\theta(t) + \phi^{\pm}\right) \tag{D.16}$$

Transforming the cosine into a sine:

$$v_{\beta}^{\pm}(t) = \pm V^{\pm} \cdot \sin\left(\theta(t) + \phi^{\pm} - \frac{\pi}{2}\right) \tag{D.17}$$

## D.3 Summary of Simplification Results

The equations at the beginning of this appendix were shown to be equal to:

$$\begin{cases} v_{\alpha}^{+}(t) & = V^{+} \cdot \sin\left(\theta(t) + \phi^{+}\right) \\ v_{\alpha}^{-}(t) & = V^{-} \cdot \sin\left(\theta(t) + \phi^{-}\right) \\ v_{\beta}^{+}(t) & = V^{+} \cdot \sin\left(\theta(t) + \phi^{+} - \frac{\pi}{2}\right) \\ v_{\beta}^{-}(t) & = -V^{-} \cdot \sin\left(\theta(t) + \phi^{-} - \frac{\pi}{2}\right) \end{cases} \tag{D.18}$$

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Appendix E

# Simplification of DQ0-QA Symmetrical Components

This appendix focuses on the simplification of the following equations (from equations 5.24 in Chapter 5):

$$\begin{cases} v_d(t) = \sin\left(\theta(t) + \phi^R\right) \cdot \left(v_\alpha^+(t) + v_\alpha^-(t) + v_\alpha^{DC}\right) - \cos\left(\theta(t) + \phi^R\right) \cdot \left(v_\beta^+(t) + v_\beta^-(t) + v_\beta^{DC}\right) \\ v_q(t) = \cos\left(\theta(t) + \phi^R\right) \cdot \left(v_\alpha^+(t) + v_\alpha^-(t) + v_\alpha^{DC}\right) + \sin\left(\theta(t) + \phi^R\right) \cdot \left(v_\beta^+(t) + v_\beta^-(t) + v_\beta^{DC}\right) \end{cases} \tag{E.1}$$

Before progressing further into this appendix, please note that this appendix makes extensive use of the following trigonometric product formulas[52]:

$$\begin{cases} \sin(\alpha) \cdot \sin(\beta) & = \frac{\cos(\alpha-\beta) - \cos(\alpha+\beta)}{2} \\ \cos(\alpha) \cdot \cos(\beta) & = \frac{\cos(\alpha+\beta) + \cos(\alpha-\beta)}{2} \\ \sin(\alpha) \cdot \cos(\beta) & = \frac{\sin(\alpha-\beta) + \sin(\alpha+\beta)}{2} \\ \cos(\alpha) \cdot \sin(\beta) & = \frac{\sin(\alpha+\beta) - \sin(\alpha-\beta)}{2} \end{cases} \tag{E.2}$$

## E.1 D Symmetrical Components

To make the deduction easier, it is best to break the $v_d(t)$ equation into smaller parts which correspond to the positive and negative sequence components and to the DC offsets:

$$v_d(t) = v_d^+(t) + v_d^-(t) + v_d^{DC}(t) \tag{E.3}$$

Where:

$$\begin{cases} v_d^+(t) & = \sin\left(\theta(t) + \phi^R\right) \cdot v_\alpha^+(t) - \cos\left(\theta(t) + \phi^R\right) \cdot v_\beta^+(t) \\ v_d^-(t) & = \sin\left(\theta(t) + \phi^R\right) \cdot v_\alpha^-(t) - \cos\left(\theta(t) + \phi^R\right) \cdot v_\beta^-(t) \\ v_d^{DC}(t) & = \sin\left(\theta(t) + \phi^R\right) \cdot v_\alpha^{DC} - \cos\left(\theta(t) + \phi^R\right) \cdot v_\beta^{DC} \end{cases} \tag{E.4}$$

Notice how $v_d^{DC}(t)$ is already fully simplified since $v_\alpha^{DC}$ and $v_\beta^{DC}$ are constants.

Taking a combined equation for $v_d^+(t)$ and $v_d^-(t)$ and substituting $v_\alpha^+(t)$, $v_\beta^+(t)$, $v_\alpha^-(t)$ and $v_\beta^-(t)$ (as per equations 5.23):

$$\begin{aligned} v_d^\pm(t) &= \sin\left(\theta(t)+\phi^R\right)\cdot V^\pm\cdot\sin\left(\theta(t)+\phi^\pm\right)\mp\cos\left(\theta(t)+\phi^R\right)\cdot V^\pm\cdot\sin\left(\theta(t)+\phi^\pm-\frac{\pi}{2}\right) \\ &= V^\pm\cdot\left(\sin\left(\theta(t)+\phi^R\right)\cdot\sin\left(\theta(t)+\phi^\pm\right)\pm\cos\left(\theta(t)+\phi^R\right)\cdot\cos\left(\theta(t)+\phi^\pm\right)\right) \end{aligned} \tag{E.5}$$

Expanding using the product formulas from E.2:

$$\begin{aligned} v_d^\pm(t) = V^\pm\cdot\frac{\cos\left(\theta(t)+\phi^R-(\theta(t)+\phi^\pm)\right)-\cos\left(\theta(t)+\phi^R+\theta(t)+\phi^\pm\right)}{2} \\ \pm V^\pm\cdot\frac{\cos\left(\theta(t)+\phi^R+\theta(t)+\phi^\pm\right)+\cos\left(\theta(t)+\phi^R-(\theta(t)+\phi^\pm)\right)}{2} \end{aligned} \tag{E.6}$$

Which when simplified results in:

$$\begin{aligned} v_d^\pm(t) = V^\pm\cdot\frac{\cos\left(\phi^R-\phi^\pm\right)-\cos\left(2\cdot\theta(t)+\phi^R+\phi^\pm\right)}{2} \\ \pm V^\pm\cdot\frac{\cos\left(2\cdot\theta(t)+\phi^R+\phi^+\right)+\cos\left(\phi^R-\phi^+\right)\right)}{2} \end{aligned} \tag{E.7}$$

Separating and simplifying the equations:

$$\begin{cases} v_d^+(t) = \frac{V^+}{2}\cdot\left(2\cdot\cos\left(\phi^R-\phi^+\right)\right) = V^+\cdot\cos\left(\phi^R-\phi^+\right) = V^+\cdot\cos\left(\phi^+-\phi^R\right) \\ v_d^-(t) = \frac{V^+}{2}\cdot\left(-2\cdot\cos\left(2\cdot\theta(t)+\phi^R+\phi^-\right)\right) = -V^-\cdot\cos\left(2\cdot\theta(t)+\phi^R+\phi^-\right) \end{cases} \tag{E.8}$$

These simplifications can then be plugged back into equation E.3:

$$\begin{aligned} v_d(t) = V^+\cdot\cos\left(\phi^+-\phi^R\right)-V^-\cdot\cos\left(2\cdot\theta(t)+\phi^R+\phi^-\right)+v_\alpha^{DC}\cdot\sin\left(\theta(t)+\phi^R\right) \\ -v_\beta^{DC}\cdot\cos\left(\theta(t)+\phi^R\right) \end{aligned} \tag{E.9}$$

## E.2 Q Symmetrical Components

Following the same logic as in the previous deductions, breaking $v_q(t)$ into smaller parts results in:

$$v_q(t) = v_q^+(t)+v_q^-(t)+v_q^{DC}(t) \tag{E.10}$$

Where:

$$\begin{cases} v_q^+(t) & = \cos\left(\theta(t)+\phi^R\right)\cdot v_\alpha^+(t)+\sin\left(\theta(t)+\phi^R\right)\cdot v_\beta^+(t) \\ v_q^-(t) & = \cos\left(\theta(t)+\phi^R\right)\cdot v_\alpha^-(t)+\sin\left(\theta(t)+\phi^R\right)\cdot v_\beta^-(t) \\ v_q^{DC}(t) & = \cos\left(\theta(t)+\phi^R\right)\cdot v_\alpha^{DC}+\sin\left(\theta(t)+\phi^R\right)\cdot v_\beta^{DC} \end{cases} \tag{E.11}$$

Taking a combined equation for $v_q^+(t)$ and $v_q^-(t)$ and substituting $v_\alpha^+(t)$, $v_\beta^+(t)$, $v_\alpha^-(t)$ and $v_\beta^-(t)$ (as per equations 5.23):

$$\begin{aligned} v_q^\pm(t) &= \cos\left(\theta(t)+\phi^R\right)\cdot V^\pm\cdot\sin\left(\theta(t)+\phi^\pm\right)\pm\sin\left(\theta(t)+\phi^R\right)\cdot V^\pm\cdot\sin\left(\theta(t)+\phi^\pm-\frac{\pi}{2}\right) \\ &= V^\pm\cdot\Big(\cos\left(\theta(t)+\phi^R\right)\cdot\sin\left(\theta(t)+\phi^\pm\right)\mp\sin\left(\theta(t)+\phi^R\right)\cdot\cos\left(\theta(t)+\phi^\pm\right)\Big) \end{aligned} \tag{E.12}$$

Expanding using the product formulas from E.2:

$$\begin{aligned} v_q^\pm(t) = V^\pm\cdot\frac{\sin\left(\theta(t)+\phi^R+\theta(t)+\phi^\pm\right)-\sin\left(\theta(t)+\phi^R-(\theta(t)+\phi^\pm)\right)}{2} \\ \mp V^\pm\cdot\frac{\sin\left(\theta(t)+\phi^R-(\theta(t)+\phi^\pm)\right)+\sin\left(\theta(t)+\phi^R+\theta(t)+\phi^\pm\right)}{2} \end{aligned} \tag{E.13}$$

Which when simplified results in:

$$\begin{aligned} v_q^\pm(t) = V^\pm\cdot\frac{\sin\left(2\cdot\theta(t)+\phi^R+\phi^\pm\right)-\sin\left(\phi^R-\phi^\pm\right)}{2} \\ \mp V^\pm\cdot\frac{\sin\left(\phi^R-\phi^\pm\right)+\sin\left(2\cdot\theta(t)+\phi^R+\phi^\pm\right)}{2} \end{aligned} \tag{E.14}$$

Separating and simplifying the equations:

$$\begin{cases} v_q^+(t) = \frac{V^+}{2}\cdot\left(-2\cdot\sin\left(\phi^R-\phi^+\right)\right) = -V^+\cdot\sin\left(\phi^R-\phi^+\right) = V^+\cdot\sin\left(\phi^+-\phi^R\right) \\ v_q^-(t) = \frac{V^-}{2}\cdot\left(2\cdot\sin\left(2\cdot\theta(t)+\phi^R+\phi^-\right)\right) = V^-\cdot\sin\left(2\cdot\theta(t)+\phi^R+\phi^-\right) \end{cases} \tag{E.15}$$

These simplifications can then be plugged back into equation E.10:

$$\begin{aligned} v_q(t) = V^+\cdot\sin\left(\phi^+-\phi^R\right)+V^-\cdot\sin\left(2\cdot\theta(t)+\phi^R+\phi^-\right)+v_\alpha^{DC}\cdot\cos\left(\theta(t)+\phi^R\right) \\ +\ v_\beta^{DC}\cdot\sin\left(\theta(t)+\phi^R\right) \end{aligned} \tag{E.16}$$

## E.3 Summary of Simplification Results

The equations at the beginning of this appendix were shown to be equal to:

$$\begin{cases} v_d(t) = V^+\cdot\cos\left(\phi^+-\phi^R\right)-V^-\cdot\cos\left(2\cdot\theta(t)+\phi^R+\phi^-\right)+v_\alpha^{DC}\cdot\sin\left(\theta(t)+\phi^R\right) \\ \qquad\qquad -\ v_\beta^{DC}\cdot\cos\left(\theta(t)+\phi^R\right) \\ v_q(t) = V^+\cdot\sin\left(\phi^+-\phi^R\right)+V^-\cdot\sin\left(2\cdot\theta(t)+\phi^R+\phi^-\right)+v_\alpha^{DC}\cdot\cos\left(\theta(t)+\phi^R\right) \\ \qquad\qquad +\ v_\beta^{DC}\cdot\sin\left(\theta(t)+\phi^R\right) \end{cases} \tag{E.17}$$

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Appendix F

# Discretization of a Generic Second-Order Transfer Function

A generic second-order transfer function is given by:

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_2 \cdot s^2 + b_1 \cdot s + b_0}{a_2 \cdot s^2 + a_1 \cdot s + a_0} \tag{F.1}$$

The bilinear transform is a method used to approximate a continuous transfer function in the discrete domain, which is given by[36]:

$$s \approx \frac{2}{T_s} \cdot \frac{1 - z^{-1}}{1 + z^{-1}} \tag{F.2}$$

Where $T_s$ is the sampling time of the discrete system. When this approximation is used on equation F.1, it yields:

$$\begin{aligned}
H(z) &= \frac{b_2 \cdot \left(\frac{2}{T_s} \cdot \frac{1-z^{-1}}{1+z^{-1}}\right)^2 + b_1 \cdot \left(\frac{2}{T_s} \cdot \frac{1-z^{-1}}{1+z^{-1}}\right) + b_0}{a_2 \cdot \left(\frac{2}{T_s} \cdot \frac{1-z^{-1}}{1+z^{-1}}\right)^2 + a_1 \cdot \left(\frac{2}{T_s} \cdot \frac{1-z^{-1}}{1+z^{-1}}\right) + a_0} \\
&= \frac{\frac{b_2 \cdot \left(\frac{2}{T_s}\right)^2 \cdot \left(1-z^{-1}\right)^2}{\left(1+z^{-1}\right)^2} + \frac{b_1 \cdot \left(\frac{2}{T_s}\right) \cdot \left(1-z^{-1}\right)}{1+z^{-1}} + b_0}{\frac{a_2 \cdot \left(\frac{2}{T_s}\right)^2 \cdot (1-z^{-1})^2}{(1+z^{-1})^2} + \frac{a_1 \cdot \left(\frac{2}{T_s}\right) \cdot (1-z^{-1})}{1+z^{-1}} + a_0} \\
&= \frac{\frac{b_2 \cdot \left(\frac{2}{T_s}\right)^2 \cdot \left(1-z^{-1}\right)^2 + b_1 \cdot \frac{2}{T_s} \cdot \left(1-z^{-1}\right) \cdot \left(1+z^{-1}\right) + b_0 \cdot \left(1+z^{-1}\right)^2}{\left(1+z^{-1}\right)^2}}{\frac{a_2 \cdot \left(\frac{2}{T_s}\right)^2 \cdot (1-z^{-1})^2 + a_1 \cdot \frac{2}{T_s} \cdot (1-z^{-1}) \cdot (1+z^{-1}) + a_0 \cdot (1+z^{-1})^2}{(1+z^{-1})^2}} \\
&= \frac{b_2 \cdot \left(\frac{2}{T_s}\right)^2 \cdot \left(1-z^{-1}\right)^2 + b_1 \cdot \frac{2}{T_s} \cdot \left(1-z^{-1}\right) \cdot \left(1+z^{-1}\right) + b_0 \cdot \left(1+z^{-1}\right)^2}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 \cdot (1-z^{-1})^2 + a_1 \cdot \frac{2}{T_s} \cdot (1-z^{-1}) \cdot (1+z^{-1}) + a_0 \cdot (1+z^{-1})^2} \\
&= \frac{b_2 \cdot \left(\frac{2}{T_s}\right)^2 \cdot \left(1-z^{-1}\right)^2 + b_1 \cdot \frac{2}{T_s} \cdot \left(1-z^{-1}\right) \cdot \left(1+z^{-1}\right) + b_0 \cdot \left(1+z^{-1}\right)^2}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 \cdot (1-z^{-1})^2 + a_1 \cdot \frac{2}{T_s} \cdot (1-z^{-1}) \cdot (1+z^{-1}) + a_0 \cdot (1+z^{-1})^2}
\end{aligned} \tag{F.3}$$

Since the top and bottom expressions are very similar, it is easier to work with just the top expressions:

$$\begin{aligned}
&b_2 \cdot \left(\frac{2}{T_s}\right)^2 \cdot \left(1 - z^{-1}\right)^2 + b_1 \cdot \frac{2}{T_s} \cdot \left(1 - z^{-1}\right) \cdot \left(1 + z^{-1}\right) + b_0 \cdot \left(1 + z^{-1}\right)^2 \\
&= b_2 \cdot \left(\frac{2}{T_s}\right)^2 \cdot \left(z^{-2} - 2 \cdot z^{-1} + 1\right) + b_1 \cdot \frac{2}{T_s} \cdot \left(1 - z^{-2}\right) + b_0 \cdot \left(z^{-2} + 2 \cdot z^{-1} + 1\right) \\
&= \left(b_2 \cdot \left(\frac{2}{T_s}\right)^2 - b_1 \cdot \frac{2}{T_s} + b_0\right) \cdot z^{-2} + \left(-2 \cdot b_2 \cdot \left(\frac{2}{T_s}\right)^2 + 2 \cdot b_0\right) \cdot z^{-1} \\
&\qquad + b_2 \cdot \left(\frac{2}{T_s}\right)^2 + b_1 \cdot \frac{2}{T_s} + b_0
\end{aligned} \tag{F.4}$$

By substitution:

$$H(z) = \frac{\left(b_2 \cdot \left(\frac{2}{T_s}\right)^2 - b_1 \cdot \frac{2}{T_s} + b_0\right) \cdot z^{-2} + \left(-2 \cdot b_2 \cdot \left(\frac{2}{T_s}\right)^2 + 2 \cdot b_0\right) \cdot z^{-1} + b_2 \cdot \left(\frac{2}{T_s}\right)^2 + b_1 \cdot \frac{2}{T_s} + b_0}{\left(a_2 \cdot \left(\frac{2}{T_s}\right)^2 - a_1 \cdot \frac{2}{T_s} + b_0\right) \cdot z^{-2} + \left(-2 \cdot a_2 \cdot \left(\frac{2}{T_s}\right)^2 + 2 \cdot a_0\right) \cdot z^{-1} + a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \tag{F.5}$$

Normalizing the constant term in the denominator, the following transfer function is obtained:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{C_0^b + C_1^b \cdot z^{-1} + C_2^b \cdot z^{-2}}{C_0^a + C_1^a \cdot z^{-1} + C_2^a \cdot z^{-2}} \tag{F.6}$$

Where:

$$\begin{cases} C_0^b = \frac{b_2 \cdot \left(\frac{2}{T_s}\right)^2 + b_1 \cdot \frac{2}{T_s} + b_0}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \\ C_1^b = \frac{-2 \cdot b_2 \cdot \left(\frac{2}{T_s}\right)^2 + 2 \cdot b_0}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \\ C_2^b = \frac{b_2 \cdot \left(\frac{2}{T_s}\right)^2 - b_1 \cdot \frac{2}{T_s} + b_0}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \end{cases} \qquad \begin{cases} C_0^a = 1 \\ C_1^a = \frac{-2 \cdot a_2 \cdot \left(\frac{2}{T_s}\right)^2 + 2 \cdot a_0}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \\ C_2^a = \frac{a_2 \cdot \left(\frac{2}{T_s}\right)^2 - a_1 \cdot \frac{2}{T_s} + a_0}{a_2 \cdot \left(\frac{2}{T_s}\right)^2 + a_1 \cdot \frac{2}{T_s} + a_0} \end{cases} \tag{F.7}$$

Thus, the difference equation is:

$$y[n] = C_0^b \cdot x[n] + C_1^b \cdot x[n-1] + C_2^b \cdot x[n-2] - C_1^a \cdot y[n-1] - C_2^a \cdot y[n-2] \tag{F.8}$$

# Appendix G

# Discretization of a First-Order Low-Pass Filter

A Butterworth first-order LPF transfer function is given by:

$$H_{LPF}(s) = \frac{Y(s)}{X(s)} = \frac{\omega_c}{s + \omega_c} \tag{G.1}$$

Where $\omega_c$ is the cutoff frequency of the filter.
The backward different approximation in the frequency domain is given by[37]:

$$s = \frac{1 - z^{-1}}{T_s} \tag{G.2}$$

By substituting this approximation in the LPF's transfer function:

$$\begin{aligned} H_{LPF}(z) = \frac{Y(z)}{X(z)} &= \frac{\omega_c}{\frac{1-z^{-1}}{T_s} + \omega_c} \\ &= \frac{\omega_c}{\frac{1-z^{-1}+\omega_c \cdot T_s}{T_s}} \\ &= \frac{\omega_c \cdot T_s}{1 + \omega_c \cdot T_s - z^{-1}} \\ &= \frac{\frac{\omega_c \cdot T_s}{1+\omega_c \cdot T_s}}{1 - \frac{1}{1+\omega_c \cdot T_s} \cdot z^{-1}} \end{aligned} \tag{G.3}$$

Let $a = \frac{\omega_c \cdot T_s}{1+\omega_c \cdot T_s}$, then:

$$H_{LPF}(z) = \frac{Y(z)}{X(z)} = \frac{a}{1 - (1 - a) \cdot z^{-1}} \tag{G.4}$$

Thus, the difference equation is:

$$y[n] = a \cdot x[n] + (1 - a) \cdot y[n - 1] \tag{G.5}$$

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Appendix H

# Discretization of the PI Controller

The PI controller is given by[44]:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau) \cdot d\tau \tag{H.1}$$

Applying the derivative to the above equation results in:

$$\frac{du}{dt} = K_p \cdot \frac{de}{dt} + K_i \cdot e(t) \tag{H.2}$$

Applying the backward difference approximation:

$$\frac{u[n] - u[n-1]}{T_s} = K_p \cdot \frac{e[n] - e[n-1]}{T_s} + K_i \cdot e[n] \tag{H.3}$$

Which when solved in order to $u[n]$ yields:

$$u[n] = u[n-1] + K_p \cdot (e[n] - e[n-1]) + K_i \cdot T_s \cdot e[n] \tag{H.4}$$

THIS PAGE
INTENTIONALLY
LEFT BLANK

# Appendix I

# Signal Conditioning Board

The signal conditioning board was designed to have the minimum amount of parts to reduce production costs. The board is based around the ACPL-C87B optical isolation amplifier[53].

Due to the amplifier's 2 V DC input restriction, the three-phase voltages must first be shifted. This offset is obtained by using a simple LM7805 voltage regulator and a trimmer potentiometer whose output is buffered by an operational amplifier (U5A). In a similar way, the output from the voltage dividers from phases A, B and C are also buffered by U9A, U10A and U11A, respectively.

Each phase's buffered voltage is added to the offset voltage, but due to the specific circuit topology used, the output voltage is actually divided by 2. These voltages are simply fed to the isolation amplifiers, and then the differential isolated output voltages are transformed into voltages referenced to the GND net by U1, U2 and U3. Since this board is simply for prototyping, different precision resistors were used because these were leftovers from previous projects, but ideally they should all be the same value.

The voltage dividers for the three-phase voltages feature multiple series resistors to ensure that the voltage ratings are not exceeded. The gain of the voltage dividers is approximately 13.216 mV/V, but due to the division by 2 from the operational amplifiers, the signal that reaches the microcontroller has a gain of approximately 6.608 mV/V. In other words, the microcontroller must multiply the voltages read by the ADC by the reciprocal of this number, which is exactly $151.\overline{3}$ V/V. Obviously, the recurring decimal $\overline{3}$ must be rounded to floating-point precision.

In Figures I.2, I.3 and I.4 the output of the board is compared with the real sine wave of each phase. As can be seen, no phase shift is introduced, the output voltages are centered around 1 V and have an amplitude of 480 mV. When compared to the 80 V amplitude of the grid, this closely matches the expected gain of 6.608 mV/V.

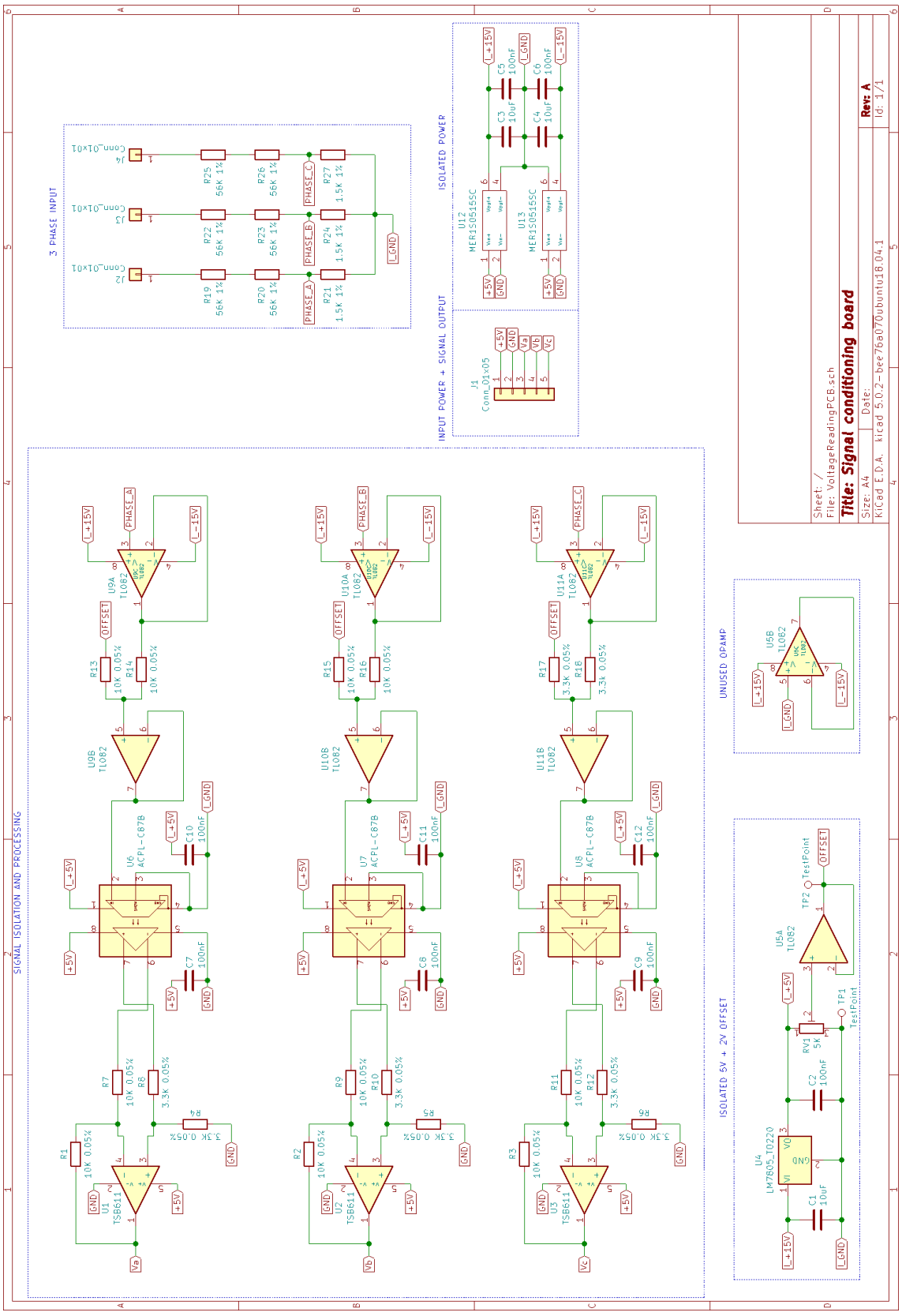



Figure I.1: Board Schematic

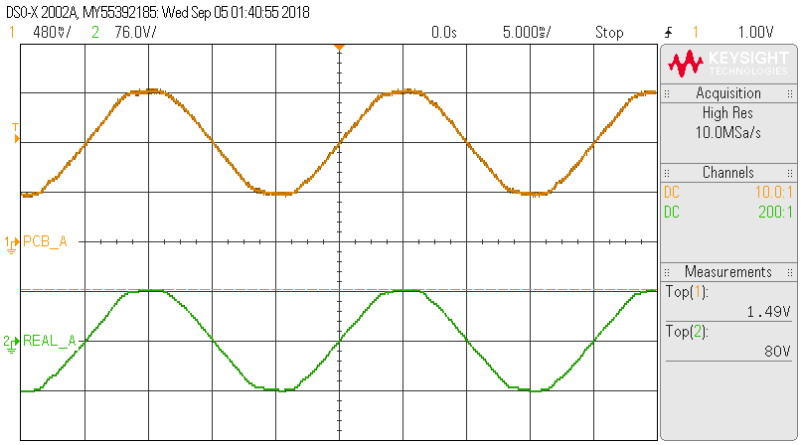



Figure I.2: Comparison between phase A and its corresponding board output

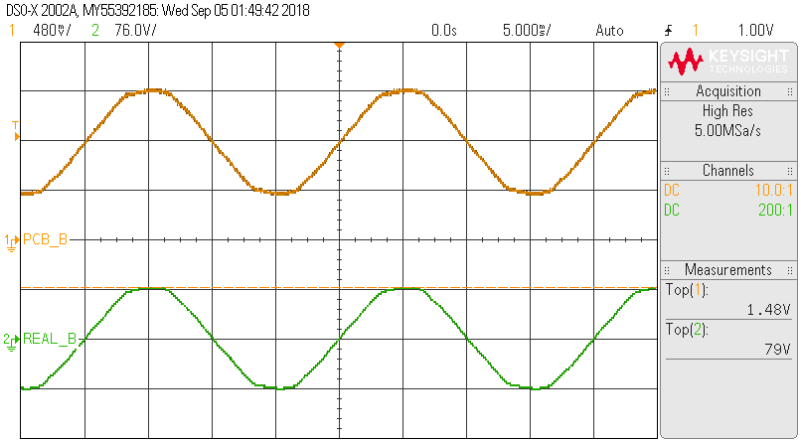



Figure I.3: Comparison between phase B and its corresponding board output

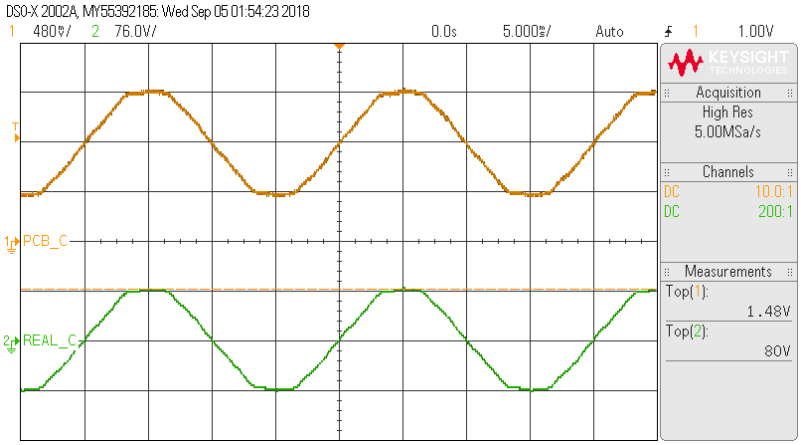



Figure I.4: Comparison between phase C and its corresponding board output

THIS PAGE
INTENTIONALLY
LEFT BLANK

# References


[1] Robert W. Erickson and Dragan Maksimović. *Fundamentals of Power Electronics*. 2nd ed. Springer, 2004, p. 883. ISBN: 0792372700.

[2] Dragan Jovcic and Khaled Ahmed. *High Voltage Direct Current Transmission: Converters, Systems and DC Grids*. 1st ed. Wiley, 2015, p. 456. ISBN: 978-1118846667.

[3] Sang-Hoon Kim. *Electric Motor Control: DC, AC, and BLDC Motors*. Elsevier, 2017, p. 438. ISBN: 9780128121382.

[4] Saeed Golestan, Mohammad Monfared, and Francisco D. Freijedo. "Design-Oriented Study of Advanced Synchronous Reference Frame Phase-Locked Loops". In: *IEEE Transactions on Power Electronics* (2013).

[5] R.W. Wall. "Simple Methods for Detecting Zero Crossing". In: *IECON* (2003).

[6] N. Jaalam et al. "A comprehensive review of synchronization methods for grid-connected converters of renewable energy source". In: *Renewable and Sustainable Energy Reviews* (2014).

[7] Farzam Baradani, Mohammad R. Dadash Zadeh, and M. Amin Zamani. "A Phase-Angle Estimation Method for Synchronization of Grid-Connected Power-Electronic Converters". In: *IEEE Transactions on Power Delivery* (2014).

[8] B.P. McGrath, D.G. Holmes, and J.J.H. Galloway. "Power Converter Line Synchronization Using a Discrete Fourier Transform (DFT) Based on a Variable Sample Rate". In: *IEEE Transactions on Power Electronics* (2005).

[9] R.E. Kálmán. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* (1960).

[10] D. Simon. "Kalman filtering with state constraints: a survey of linear and nonlinear algorithms". In: *IET Control Theory & Applications* (2010).

[11] Sudarshan Swain and Bidyadhar Subudhi. "A new grid synchronization scheme for a three phase PV system employing Kalman filtering". In: *IEEE Region 10 Symposium* (2017).

[12] Ming Sun and Zafer Sahinoglu. "Extended Kalman filter based grid synchronization in the presence of voltage unbalance for smart grid". In: *Innovative Smart Grid Technologies* (2011).

[13] Chenchen Wu, Mario E. Magaña, and Eduardo Cotilla-Sánchez. "Dynamic Frequency and Amplitude Estimation for Three-Phase Unbalanced Power Systems Using the Unscented Kalman Filter". In: *IEEE Transactions on Instrumentation and Measurement* (2018).

[14] Anh Tuan Phan, Gilles Hermann, and Patrice Wira. "A new state-space for unbalanced three-phase systems: Application to fundamental frequency tracking with Kalman filtering". In: *Mediterranean Electrotechnical Conference* (2016).

[15] Vivienne Sze et al. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". In: *Proceedings of the IEEE* (2017).

[16] L.L. Lai et al. "Real-time frequency and harmonic evaluation using artificial neural networks". In: *IEEE Transactions on Power Delivery* (1999).

[17] Henri de Bellescize. "La réception synchrone". In: *L'Onde Électrique* (1932).

[18] Saeed Golestan, Josep M. Guerrero, and Juan C. Vasquez. "Three-Phase PLLs: A Review of Recent Advances". In: *IEEE Transactions on Power Electronics* (2016).

[19] David Salomon. *Transformations and Projections in Computer Graphics*. 1st ed. Springer, 2006, p. 290. ISBN: 978-1846283925.

[20] Surajit Chattopadhyay, Madhuchhanda Mitra, and Samarjit Sengupta. *Electric Power Quality*. 1st ed. Springer, 2011, p. 182. ISBN: 978-9400706347.

[21] *Plane*. URL: `http://mathworld.wolfram.com/Plane.html` (visited on Mar. 13, 2019).

[22] Hirofumi Akagi et al. "Generalized Theory of Instantaneous Reactive Power and Its Application". In: *Electrical Engineering in Japan* (1983).

[23] *Rotation Matrix*. URL: `http://mathworld.wolfram.com/RotationMatrix.html` (visited on Mar. 27, 2019).

[24] Saeed Golestan, Josep M. Guerrero, and Abdullah M. Abusorrah. "MAF-PLL With Phase-Lead Compensator". In: *IEEE Transactions on Industrial Electronics* (2014).

[25] William McC. Siebert. *Circuits, Signals, and Systems*. 11th ed. MIT Press, 1998, p. 651. ISBN: 978-0262192293.

[26] B. Indu Rani et al. "A three phase PLL with a dynamic feed forward frequency estimator for synchronization of grid connected converters under wide frequency variations". In: *International Journal of Electrical Power and Energy Systems* (2012).

[27] C.L. Fortescue. “Method of Symmetrical Co-Ordinates Applied to the Solution of Polyphase Networks”. In: *Transactions of the American Institute of Electrical Engineers* (1918).

[28] K.F. Riley, M.P. Hobson, and S.J. Bence. *Mathematical Methods for Physics and Engineering*. 3rd ed. Cambridge University Press, 2006, p. 1359. ISBN: 978-0521679718.

[29] *Visualisation of symmetrical components*. URL: `http://stevenblair.github.io/seq` (visited on Oct. 8, 2019).

[30] *Euler Formula*. URL: `http://mathworld.wolfram.com/EulerFormula.html` (visited on Mar. 30, 2019).

[31] Tuomas Messo et al. “Improved delayed signal cancellation-based SRF-PLL for unbalanced grid”. In: *IEEE Energy Conversion Congress and Exposition* (2017).

[32] Pedro Rodríguez et al. “Multiresonant Frequency-Locked Loop for Grid Synchronization of Power Converters Under Distorted Grid Conditions”. In: *IEEE Transactions on Industrial Electronics* (2010).

[33] Menxi Xie et al. “DC Offset Rejection Improvement in Single-Phase SOGI-PLL Algorithms: Methods Review and Experimental Evaluation”. In: *IEEE Access* (2017).

[34] K. Sonam et al. “Implementation of single-phase modified SRF-PLL using model based development approach”. In: *North American Power Symposium* (2017).

[35] Pedro Rodriguez et al. “Multiple Second Order Generalized Integrators for Harmonic Synchronization of Power Converters”. In: *IEEE Energy Conversion Congress and Exposition* (2009).

[36] Richard G. Lyons. *Understanding Digital Signal Processing*. 3rd ed. Prentice Hall, 2010, p. 982. ISBN: 978-0137027415.

[37] John G. Proakis and Dimitris K. Manolakis. *Digital Signal Processing: Principles, Algorithms and Applications*. 3rd ed. Prentice Hall, 1995, p. 1016. ISBN: 9780133737622.

[38] *ARM Cortex-M4 Processor - Technical Reference Manual*. Revision r0p1. ARM, 2015.

[39] *XMC4700 Relax Kit*. URL: `https://www.infineon.com/cms/en/product/evaluation-boards/kit_xmc47_relax_v1/` (visited on June 17, 2019).

[40] Richard G. Lyons. *Streamlining Digital Signal Processing: A Tricks of the Trade Guidebook*. 2nd ed. Wiley, 2012, p. 496. ISBN: 9781118278383.

[41] *How to Find a Fast Floating-Point atan2 Approximation*. URL: `https://www.dsprelated.com/showarticle/1052.php` (visited on June 21, 2019).

[42] *Using the GNU Compiler Collection (GCC): Options That Control Optimization*. URL: `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html` (visited on June 23, 2019).

[43] *FreeRTOS Support Archive*. URL: https://www.freertos.org/FreeRTOS_Support_Forum_Archive/May_2014/freertos_FreeRTOS_fails_to_link_when_compiled_with_lto_86691e41j.html (visited on June 23, 2019).

[44] Karl J. Åström and Tore Hägglund. *PID Controllers - Theory, Design and Tuning*. 2nd ed. ISA, 1995, p. 343. ISBN: 9781556175169.

[45] *Trigonometric Addition Formulas*. URL: http://mathworld.wolfram.com/TrigonometricAdditionFormulas.html. (Visited on Mar. 18, 2019).

[46] *DSP Trick: Magnitude Estimator*. URL: http://dspguru.com/dsp/tricks/magnitude-estimator/ (visited on Aug. 31, 2019).

[47] Randy L. Haupt and Sue Ellen Haupt. *Practical Genetic Algorithms*. 2nd ed. Wiley-Interscience, 2004, p. 272. ISBN: 9780471455653.

[48] C. L. Hwang and A. S. M. Masud. *Multiple Objective Decision Making — Methods and Applications: A State-of-the-Art Survey*. 1st ed. Springer, 1979, p. 351. ISBN: 9783540091110.

[49] *Norm*. URL: http://mathworld.wolfram.com/Norm.html (visited on Mar. 17, 2019).

[50] *Trigonometry*. URL: http://mathworld.wolfram.com/Trigonometry.html. (Visited on Mar. 18, 2019).

[51] *Prosthaphaeresis Formulas*. URL: http://mathworld.wolfram.com/ProsthaphaeresisFormulas.html. (Visited on Mar. 20, 2019).

[52] *Werner Formulas*. URL: http://mathworld.wolfram.com/WernerFormulas.html (visited on Apr. 26, 2019).

[53] *ACPL-C87B: Precision Optically Isolated Voltage Sensor*. URL: https://www.broadcom.com/products/optocouplers/industrial-plastic/isolation-amplifiers-modulators/isolation-amplifiers/acpl-c87b (visited on June 29, 2019).