Neural Networks Based Space Vector Modulation and Control of Induction Machines

Akshat Vijaywargiya

Neural Networks Based Space Vector Modulation and Control of Induction Machines

Report submitted in fulfilment of the
BTech Project for the Academic year 2021-2022
of

Bachelor of Technology

in
Electrical Engineering

&

Master of Technology

in
Power Electronics and Drives
(Dual Degree)

by

Akshat Vijaywargiya
18EE02004

Under the supervison of

Dr. Dipankar De & **Dr. N.B Puhan**
Assistant Professor Assistant Professor
SCHOOL OF ELECTRICAL SCIENCES
INDIAN INSTITUTE OF TECHNOLOGY BHUBANESWAR
April 2022

CERTIFICATE

This is to certify that the BTech Project entitled **"Neural Networks Based Space Vector Modulation and Control of Induction Machines"** submitted by **Akshat Vijaywargiya** (18EE02004) to Indian Institute of Technology Bhubaneswar is a record of the research work under my supervision and I accept the report submitted is worthy of consideration for the final evaluation of the BTech Project(BTP) work.

(Supervisor I)

(Supervisor II)

DECLARATION

I declare that

1. The work contained in the thesis is original and has been done by myself under the general supervision of my supervisor.
2. The work has not been submitted to any other Institute for any degree or diploma.
3. I have followed the guidelines provided by the Institute in writing the report.
4. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
5. Wherever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.
6. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Akshat Vijaywargiya
18EE02004

Abstract

Neural Networks Based Space Vector Modulation and Control of Induction Machines

by

Akshat Vijaywargiya

Space Vector Modulation (SVM) in three phase voltage source and current source converters has gained popularity in recent times as the switching capabilities of semiconductor devices have improved. Inclusion of hybrid zone techniques into conventional SVM has considerably reduced THD levels of the SVM technique. However, the calculation of switching times causes more delay since hybrid zones need to be compared for THD and optimally chosen, which takes more time for determination.Faster switching in SVM would result in lesser THD. However, the delay which SVM causes limits the maximum switching frequency. Machine Learning techniques which are recently being developed can provide efficient generation of switching logic by parallelizing the major part of switching logic. This project is aimed to incorporate Machine Learning techniques to efficiently generate the switching logic for inverter.

Vector control of induction machine is one of the most popular control technique for induction machines because of better steady as well as transient state performance. However, having two cascaded PI controllers sometimes result in comparatively poor performance of the overall system, as in case of lower sampling frequencies. This project also aims to replace the inner two current control loops of the vector control scheme with a recurrent neural network to improve the performance of overall system.

A preset model of 3 phase induction motor having rated values of line voltage 400V, speed 1430 RPM, frequency 50 Hz and power 4 KW was used to verify the expected results. Various calculations related to the motor like the calculation of the d and q axes fluxes and errors in the same were made and verified using the Simulink model. These calculations were further extended to incorporate the overmodulation range into the picture. Both the neural networks were used together to check their compatability and combined performance.

Contents

List of Figures

List of Tables

Abbreviations

PWM	:	Pulse Width Modulation
SVM	:	Space Vector Modulation
VSI	:	Voltage Source Inverter
THD	:	Total Harmonic Distortion
ML	:	Machine Learning
NN	:	Neural Networks
OM	:	Overmodulation
FOC	:	Field Oriented Control
AC	:	Alternating Current
PI	:	Proportional-Integral
V/f	:	Voltz/Hertz
IM	:	Induction Machine
ANN	:	Artificial Neural Networks
RNN	:	Recurrent Neural Networks

Chapter 1

Introduction and Theory

1.1 Introduction to Space Vector Modulation

Pulse Width Modulation (PWM) is a technique which is used to convert DC voltage to AC by effectively chopping the DC voltage into discrete parts of varying width. Various types of PWM techniques are available today, out of which Space Vector Modulation (SVM) is the most commonly used because of its varied advantages.

The rapid increase in the semiconductor switching capabilities in the recent past has opened the gates for fast, accurate and simpler PWM techniques. The Space Vector Modulation (SVM) is among the most simple yet efficient PWM techniques that could be used to effectively use the recently developed fast switching capabilities. Though it is very simple to implement in digital systems but in practice it requires a bit of calculations which might restrict the minimum sampling time.

By using the recently developed machine learning techniques and using vector computations which are computationally faster can significantly reduce the computation time for calculating switching times for each sample thereby increasing the maximum sampling frequency possible. The higher frequency of sampling decreases the error in generated voltages and hence ensures more smooth performance of the connected electrical equipment.

Since the connected electrical equipment is usually a 3 phase AC Motor (Induction Motor) hence the modulation technique used should be effective for operation of these motors. The induction motor gives best performance if the input voltage to it is perfectly sinusoidal and the motor is operated under rated conditions. The higher sampling frequency in modulation techniques helps in reducing the higher order harmonics in generated voltage hence the motor operation and performance are considerably improved.

1.2 Space Vector Modulation

The three phase voltages of the balanced 3 phase supply always add upto zero. This

implies that if any two phase voltages are known at any point of time, then the third phase voltage could be found by using the two given voltages. Hence we can represent the three phases as a function of two independent variables which are orthogonal. Let the three phase voltages be V_a, V_b, V_c and the corresponding independent variables be V_α and V_β. The corresponding relation can be written as:

$$V = \begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix} = \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ 1 & \frac{\sqrt{3}}{2} & \frac{-\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \quad \ldots \quad (1.1)$$

This transformation can be written for currents as well in a similar fashion. This kind of transformation to the permitted switching states of a voltage source inverter (VSI) results in six non-zero phase voltage space vectors ($\vec{V_k}$, $k = 1, \ldots, 6$), forming an hexagon centered at the origin of the $\alpha\beta$ plane, and the remaining two zero line voltage space vectors ($\vec{V_0}$, $\vec{V_7}$) located at the origin of the plane and correspond to zero voltages. These six space vectors form an hexagon as shown in Fig 1.2. The two zero space vectors are also shown on the origin. The plane is divided into six parts called *sectors* which are numbered from one to six in anti-clockwise fashion as shown in Fig 1.2.

The circuit diagram for a standard 3 phase VSI is as shown in Fig 1.1. The voltages generated due to various (allowed) combinations of the different switches is shown in Table 1.1 in α-β frame.

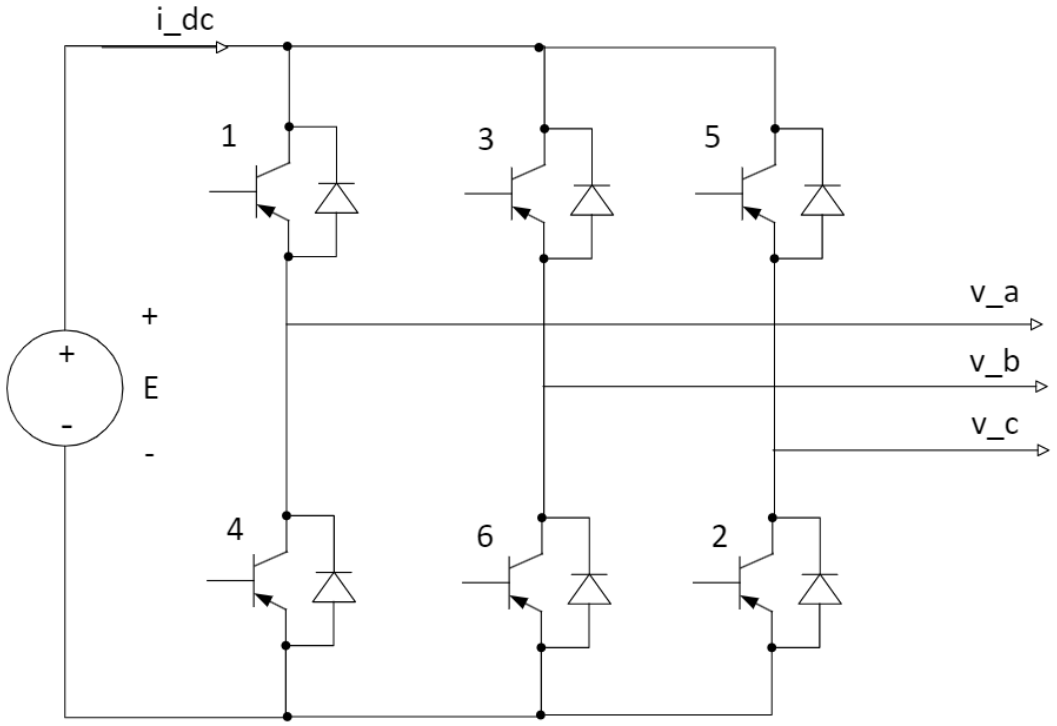


Figure 1.1: Circuit Diagram for 3 Phase VSI

Table 1.1: Switching Combinations for SVM

State (k)	**On Switches**	v_{an}	v_{bn}	v_{cn}	V_k
0.	2,4,6	0	0	0	0
1.	1,2,6	E	0	0	E
2.	1,3,2	E	E	0	$(1/2 + \mathrm{j}\sqrt{3}/2)E$
3.	2,3,4	0	E	0	$(-1/2 + \mathrm{j}\sqrt{3}/2)E$
4.	3,4,5	0	E	E	-E
5.	4,5,6	0	0	E	$-(1/2 + \mathrm{j}\sqrt{3}/2)E$
6.	1,5,6	E	0	E	$(1/2 - \mathrm{j}\sqrt{3}/2)E$
7.	1,3,5	E	E	E	0

The rotating $\vec{V_{ref}}$ voltage vector is the voltage desired to be generated. It rotates along the plane with constant angular frequency ω and has constant magnitude throughout its cycle for ideal sinusoidal voltage. $\vec{V_{ref}}$ needs to be generated for each sampled angle θ. It can be generated using the existing 8 vectors ($\vec{V_k}, [k = 1, \ldots, 8]$) for any θ by applying the adjacent state vectors and the zero vectors for appropriate amount of time within the sampling interval so that the average vector generated within a switching time interval is same as $\vec{V_{ref}}$. The SVM strategy, therefore, defines the sequence of the space vectors (switching states) in such a way that their time average reproduces the reference vector within one cycle. Note that $\vec{V_{ref}}$ can only be reproduced exactly only when the tip of it lies inside the shown hexagon.

So we need to define a switching time sequence for each sample of $\vec{V_{ref}}$ which

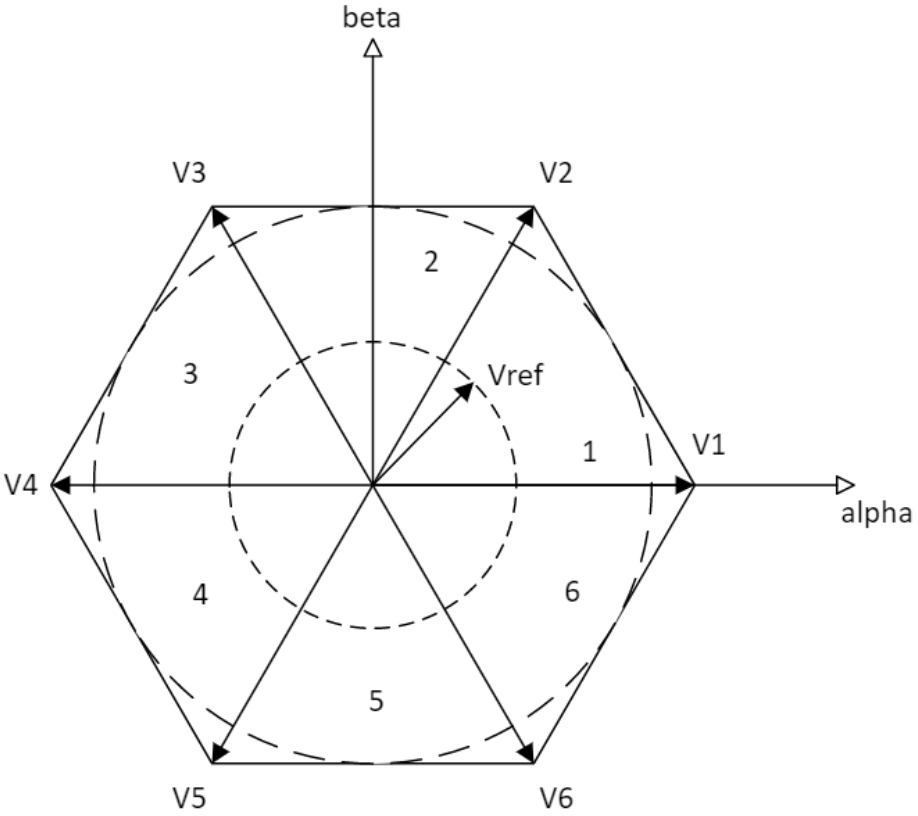


Figure 1.2: Sectors and space vectors in the complex plane for VSI

obeys the above condition. The next section details how to calculate the duration for which the vectors need to be applied.

1.2.1 Calculation of Switching Times of Space Vectors

The best tracking of the reference vector is obtained when the switching sequence includes only the two vectors adjacent to the reference and a zero vector. Also, the converter operates in its linear mode if the length of the reference vector does not exceed the radius of the inscribed circle of the hexagon. Thus in the linear mode of operation, the reference space vector is generated by the time average of the three switching state vectors located adjacent to it (including one null space vector). This yields:

$$\vec{V_{ref}} T_s = \vec{V_i} t_i + \vec{V_{i+1}} t_{i+1} + \vec{V_0} t_0 \qquad \dots \quad (1.2)$$

where T_s is the sampling period (or cycle period) and t_i, t_{i+1} and t_0 are the respective on-duration times of the adjacent switching state vectors. These timings are given by the following equation:

$$\begin{bmatrix} t_i \\ t_{i+1} \end{bmatrix} = m \begin{bmatrix} sin(60-\theta) \\ sin(\theta) \end{bmatrix} T_s \qquad \dots \quad (1.3)$$

$$t_0 = T_s - t_i - t_{i+1} \qquad \ldots \quad (1.4)$$

where θ is the angle between the reference vector and the closest clockwise state vector and m is the modulation index and is given by:

$$m = \frac{2}{\sqrt{3}} \frac{||\vec{V_{ref}}||}{E} \qquad \ldots \quad (1.5)$$

The above equation yields $m = 1$ when the locus of tip of $\vec{V_{ref}}$ just touches the outer hexagon but never crosses it. For $m < 1$ the locus lies entirely inside the hexagon and this is called the linear mode of operation. When $m > 1$ the locus crosses the hexagon at some points and the equation (1.3) does not give correct results since t_0 becomes negative. This is called the overmodulation mode of operation. There are special ways to recreate the original $\vec{V_{ref}}$ in overmodulation mode as shown in the next subsection.

For Overmodulation Mode I $(1 < m < 1.05)$ **:**

The original trajectory of $\vec{V_{ref}}$ is modified as shown in the below figure.

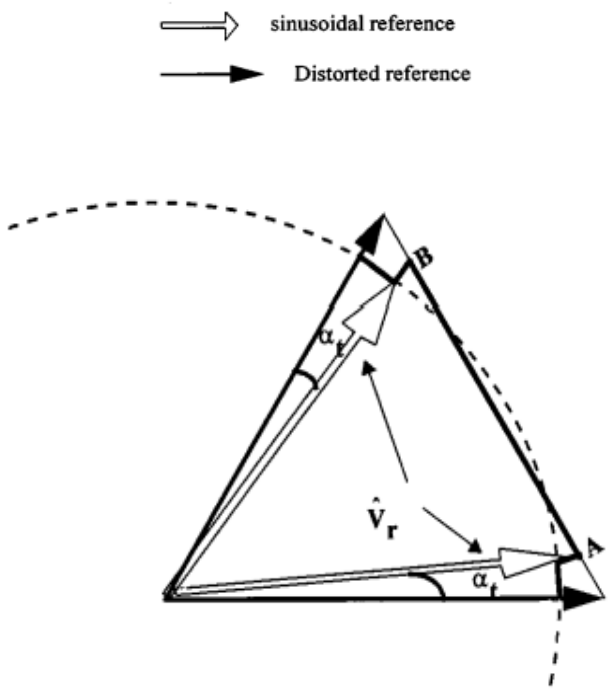


Figure 1.3: Sinusoidal and distorted $\vec{V_{ref}}$ in overmodulation mode I

The sinusoidal reference shown by blank arrow in above figure is the ideal $\vec{V_{ref}}$. The distorted $\vec{V_{ref}}$ shown by filled arrow is the $\vec{V_{ref}}$ we generate. Let the $\vec{V_{ref}}$ vector cross the hexagon at some angle $2\alpha_t$. Then the switching times are determined for $\theta < \alpha_t$ as per the equation (1.3) directly. Otherwise, $\vec{V_{ref}}$ is clamped to the hexagon and it moves along on the hexagon edge for greater angles. The switching times are then calculated as

per equation (1.3) on this clamped $\vec{V_{ref}}$. This process is symmetrically repeated for other half of the sector as well.

For Overmodulation Mode II $(1.05 < m < 1.10)$ **:**

The original trajectory of $\vec{V_{ref}}$ is modified as shown in the below figure.

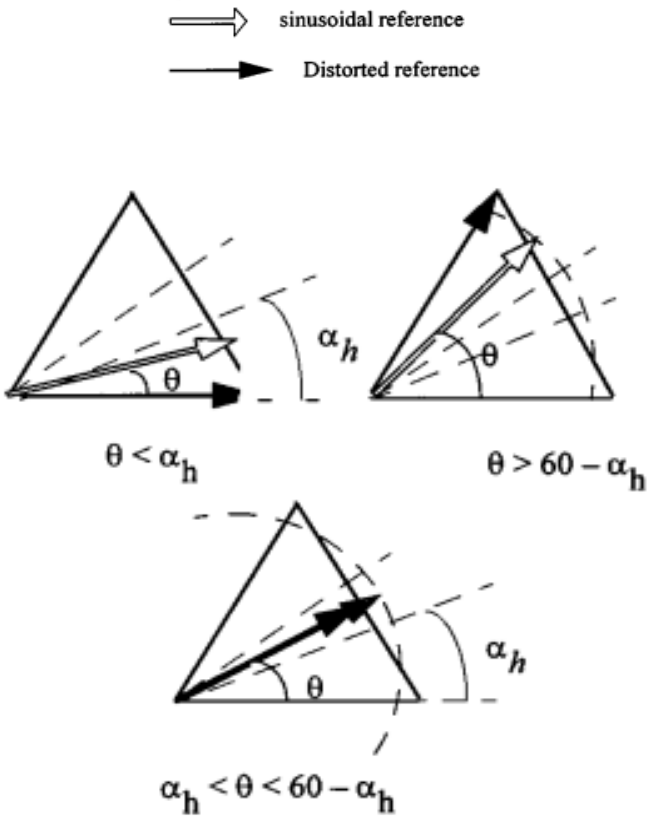


Figure 1.4: Sinusoidal and distorted $\vec{V_{ref}}$ in overmodulation mode II

The sinusoidal reference shown by blank arrow in above figure is the ideal $\vec{V_{ref}}$. The distorted $\vec{V_{ref}}$ shown by filled arrow is the $\vec{V_{ref}}$ we generate. Let the $\vec{V_{ref}}$ vector cross the hexagon at some angle α_h. If $\theta < \alpha_h$ then $\vec{V_{ref}}$ gets clamped to the nearest fixed vector $(\vec{V_1} - \vec{V_6})$ and switching calculations are calculated as per the equation (1.3). Otherwise, $\vec{V_{ref}}$ is clamped to the hexagon and it moves along on the hexagon edge for greater angles. The switching times are then calculated as per equation (1.3) on this clamped $\vec{V_{ref}}$. This process is symmetrically repeated for other half of the sector as well.

The main reason to incorporate overmodulation region into SVM is to better utilize the DC bus. DC bus is only 86.6% utilized in linear modulation whereas it is utilized to about 95.3% when both the overmodulation ranges are incorporated into SVM. Although seems beneficial, a major problem of higher THD comes into picture when overmodulation ranges are used in SVM. Hence even efficient ways to apply vectors are needed to improve performance.

The on-line implementation of the SVM requires that at every sample time the sector where the space vector reference lies be established, the modulation index (m) calculated, and equations (1.3) and (1.4) computed. Therefore, the trigonometric function *sin* in equation (1.3) must be on-line computed. In conventional implementations, the *sin* function is stored in a look-up table and by means of interpolation, the value can be reasonably approximated. However, this approach presents three main disadvantages:

a) the use of a, look-up table implies the need for additional memory,

b) interpolation of non-linear functions leads to poor accuracy in the calculations and thereby, contributes to the deterioration of the harmonic spectrum of the PWM waveforms, and

c) the use of a look-up table and interpolation as a method of approximation, demands additional computing time that limits the maximum switching frequency of the converter.

The use of Machine Learning techniques can be used to address these problems. Although the ML method would have its own delay but it is faster comparatively.

1.3 Switching Times of Switches from Switching Times of Space Vectors

The space vectors do not exist independently but need to be generated by the six switches of the three phase VSI. Hence when it is said to apply a particular space vector for some time, it means that a particular combination of those six switches must be applied for that duration.

The space vectors which needs to be applied depends upon the present *sector* in which $\vec{V_{ref}}$ lies. Depending upon the space vectors which needs to be applied and their duration, the switching times of different switches is calculated.

To prevent any dependency of the calculation of switching times of different switches upon previous switching pattern (of the previous sample), it is important that all the switches remain in same state after before and after generating the required $\vec{V_{ref}}$ in a particular sampling interval. One easy way to do this is to apply the vectors in a symmetrical manner around half of the sampling interval i.e. apply a sequence of vectors in some valid manner till $T_s/2$ time and then apply the same sequence in opposite manner in the rest $T_S/2$ time. Of course, the vector timings need to be satisfied to ensure correct $\vec{V_{ref}}$ gets created.

Many sequence of application of vectors are possible which satisfy the above criteria. One of the most common switching sequence, the 0127 sequence, is described below.

In this sequence, first the zero vector ($\vec{V_0}$) is applied for $T_0/4$ time, followed by the first vector $\vec{V_1}$ applied for $T_1/2$ time, then second vector $\vec{V_2}$ for $T_2/2$ time and finally

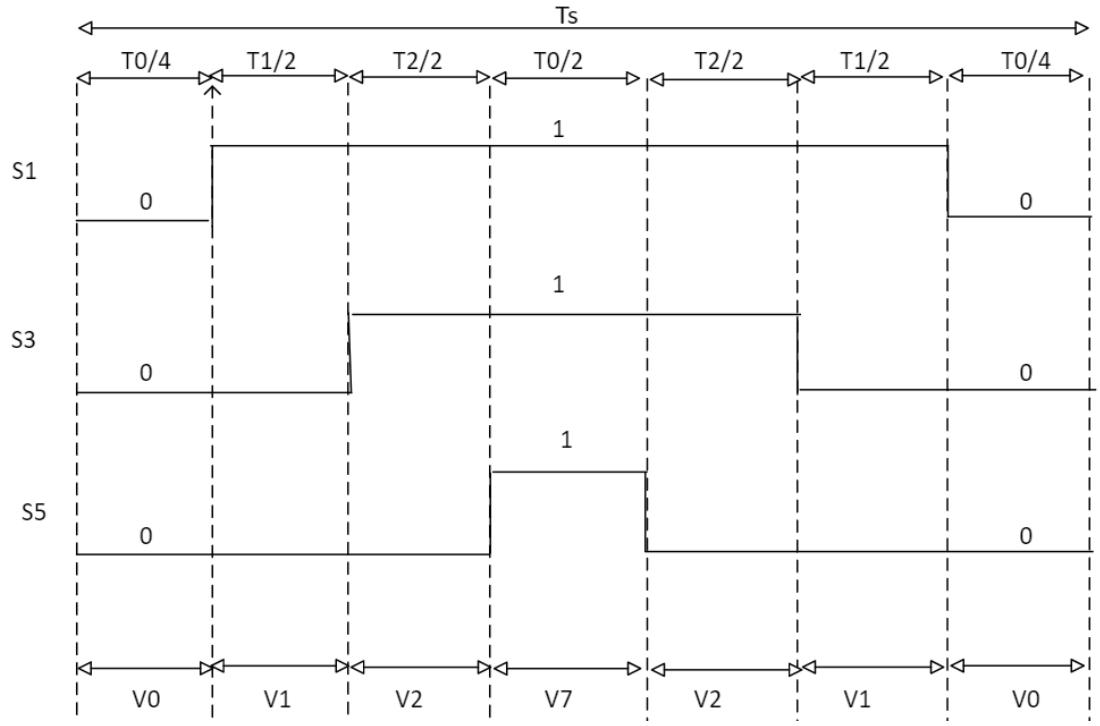


Figure 1.5: Switching Sequence 0127 Pattern

zero vector $\vec{V_7}$ again for $T_0/4$ time. The exact pattern is applied in other half of sampling period but in exact opposite manner so that symmetry is obtained. Figure 1.5 shows depicts the switching pattern. The name of the sequence is 0127 because it is the way in which the vectors are applied sequentially in the first half of sampling period. Similarly other switching sequences are possible, like 0121, 7212, 1012, 2721, etc. Their switching pattern can be obtained similarly.

One clear advantage of the switching pattern that can be seen from Fig 1.5 is that during the change of application of space vectors, only one switch's needs to be changed, which reduces the burden on computer and thus helps in high frequency operation. Also, it can be easily observed from Fig 1.5 that the switching pattern is such that not only in the end of sampling period that the average voltage vector generated is same as $\vec{V_{ref}}$ but at the half cycle ($\frac{T_2}{2}$) also the average vector generated is same as $\vec{V_{ref}}$. This helps is reducing the error in corresponding torque and fluxes generated by the motor. This applies to other switching sequences as well.

1.4 Introduction to Vector Control of Induction Machines

Vector control, also called field-oriented control (FOC), is a variable-frequency drive (VFD) control method in which the stator currents of a three-phase AC or brushless DC electric motor are identified as two orthogonal components that can be visualized

with a vector. One component defines the magnetic flux of the motor, the other the torque. The control system of the drive calculates the corresponding current component references from the flux and torque references given by the drive's speed control. Typically proportional-integral (PI) controllers are used to keep the measured current components at their reference values. The pulse-width modulation of the variable-frequency drive defines the transistor switching according to the stator voltage references that are the output of the PI current controllers.

FOC is used to control AC synchronous and induction motors. It was originally developed for high-performance motor applications that are required to operate smoothly over the full speed range, generate full torque at zero speed, and have high dynamic performance including fast acceleration and deceleration. However, it is becoming increasingly attractive for lower performance applications as well due to FOC's motor size, cost and power consumption reduction superiority. It is expected that with increasing computational power of the microprocessors it will eventually nearly universally displace single-variable scalar volts-per-Hertz (V/f) control.

Vector control is of two types: Direct Vector Control and Indirect Vector Control. In direct vector control, the rotor angle or control vector is obtained by the terminal voltage and currents directly by using flux estimators whereas in indirect vector control, the angle is obtained by using rotor position measurement and machine parameter estimation. In terms of performance, the indirect vector controlled IM drive gives better performance due to elimination of additional sensors and is also cheap. In further sections, only indirect vector control method is discussed and implemented.

1.5 Indirect Vector Control of Induction Machines

1.5.1 Induction Machine Equations

In vector control method, first we have to define the dq axes for deriving the control equations. In induction machines, the stator current can be impressed in the machine so that the rotor current induced and their interaction will produce the torque. Hence we will give stator current commands by controlling the inverter output voltage. But the rotor flux changes are slow and does not get disturbed suddenly by the stator side commands. Hence the alignment of the reference dq frame should be with rotor flux.

The induction machine can be represented by three dynamic equations as written below:

$$T_r \frac{d\psi_r}{dt} + \psi_r = M i_{sd} \ldots \quad (1.6)$$

$$\omega_{mr} = \omega + \frac{M i_{sq}}{T_r \psi_r} \ldots \quad (1.7)$$

$$J\frac{d\omega_m}{dt} = \frac{2}{3}\frac{P}{2}\frac{1}{1+\sigma_r}\psi_r i_{sq} - m_l \ldots \quad (1.8)$$

The above equations are valid if our d axis is aligned with the rotor flux, as stated earlier. From the first equation, if the input i_{sd} is given to system, then ψ_r will respond with time constant T_r which is high. This means that response in i_{mr} change or ψ_r change is slow. From the second equation, it can be seen that slip of rotor flux with respect to rotor or simply the slip of the machine is controlled by i_{sq}. From the third equation it can be seen that the response in machine speed is also slow as its inertia J is large for high power machines.

Also, under steady state $d\psi_r/dt = 0$, which makes $\psi_r = Mi_{sd}$. This means the flux will get controlled by i_{sd}.

Also, w_{mr} is the rotor flux speed which is the same as synchronous speed. Let the rotor flux phasor be at an angle ρ with respect to the stationary $\alpha - \beta$ axis. Then the speed of rotation of rotor flux phasor can be given by the rate of change of this angle ρ. Mathematically,

$$\omega_{mr} = \frac{d\rho}{dt} = \omega + \frac{Mi_{sq}}{T_r\psi_r} \ldots \quad (1.9)$$

Since our d-axis is fixed on rotor flux phasor only, so the angle ρ is also the angle the d-axis makes with the α axis. Hence we can properly align our d-axis using the above equation.

1.5.2 Control Scheme

The overall control block diagram for indirect vector control is as shown in Fig 1.6.

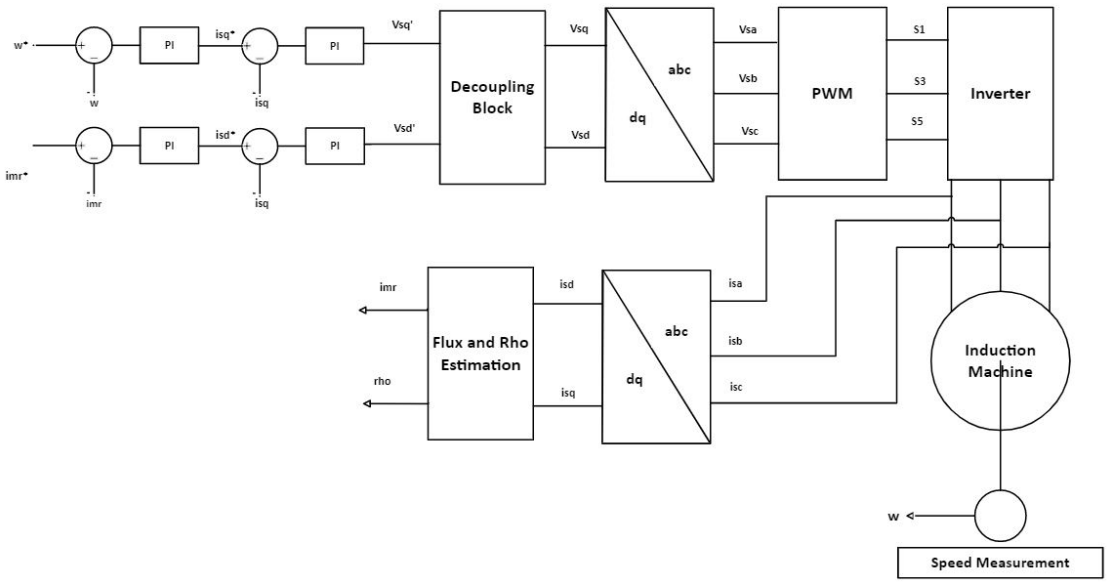


Figure 1.6: Vector Control Block Diagram

As can be seen from the block diagram, there are 4 controllers (PI) in the system,

namely the speed controller, torque/q axis current controller, flux controller and the d axis current controller. From the previous section, it was observed that if the d axis is fixed at rotor flux, then only the q axis component of stator current controls the electromagnetic torque of the machine and d axis stator current controls the flux in the machine. Hence the flux controller output is the reference d axis current we want to ensure some amount of flux in the machine which another another PI controller in series (d axis current controller) ensures. Similarly the speed controller comes before q axis current controller because as load increases, the machine speed will fall if i_{sq} remains constant. Hence in order to maintain speed constant to some reference value, i_{sq} must be controlled accordingly. So, the output of the speed controller is fed as reference to the q axis current controller.

But the d axis and q axis current controls are not fully independent. In order to make them independent from each other, a decoupling block is added to the the output of the two current controllers. The next subsection describes the decoupling block.

Assuming the decoupling block is able to completely decouple the d and q axis currents, the output is then provided to the 3 phase voltage source inverter which is the SVM based inverter previously discussed. The inverter generates 3 phase sinusoidal voltage at its output which is fed to the 3 phase induction machine.

The three phase machine input stator currents are measured and flux estimation is done using them (indirect way of control). Firstly the three phase currents are converter to dq currents using Park's transform. The equations described in section 1.5.1 are used to relate these dq currents and meaured machine speed (ω) to ψ_r as well as the synchronous machine speed ω_{mr}. The same set of equations are also used to determine the dq frame orientation (ρ). The four values (ω, i_{sq}, i_{mr} and i_{sd}) are feedbacked to the four PI controllers respectively. The value of ρ is given to inverter as input to properly align the dq axis.

1.5.3 Decoupling Terms

As seen previously, the machine equations for speed and torque not only depend on i_{sq} but also on ψ_r which makes the speed controller depend on flux controller and vice versa. In order to remove these kind of dependencies, decoupling block is introduced, which subtracts some product of some machine variables from the output of both the current controllers. These terms inherently get added up in machine and in order to account for that we subtract these terms from the input only to make overall addition and subtraction nullified.

The decoupling terms relating the output of the current controllers (V'_{sq} and V'_{sd}) to the input given to 3 phase voltage source inverter (V_{sq} and V_{sd}) are given as follows.

$$V_{sq} = V'_{sq} + \frac{\psi_r w_{mr}}{G(1+\sigma_r)} + \frac{\sigma L_s w_{mr} i_{sd}}{G} \dots \quad (1.10)$$

$$V_{sd} = V'_{sd} + \frac{1}{G(1+\sigma_r)}\frac{d\psi_r}{dt} - \frac{\sigma L_s w_{mr} i_{sq}}{G} \ldots \quad (1.11)$$

1.6 Challenges and Motivation

The main challenge for SVM is the on-line implementation of it. The computers available today have the limits of computational capacity and cost and hence efficient use of them is needed. The computation speed for calculation of switching times should be such so as to keep the delay within some specified limits. The conventional method causes considerable delay in calculation of switching times. We want to make voltages as analogous as possible to make the life of electrical appliances longer as well as to get minimum error and corresponding power losses. This can be done through high frequency sampling of the desired $\vec{V_{ref}}$. With such high frequencies of operation of SVM, delay starts limiting the sampling frequency thereby not allowing higher frequency operation.

One interesting approach to deal with such kind of high frequency operation challenges is to make use of Machine Learning techniques, which have recently proved to be beneficial in a varied kind of tasks. The on-line computer could just be a Machine Learning program and depending on various factors, it can adjust the switching time values by itself. The main challenge to develop such an algorithm is to figure out the "loss function" for the ML model. One such function could be the "RMS Flux Error Function" of the motor since the stator flux generated by motor would possess ripples if the input voltage to the motor is not perfectly sinusoidal.

Similarly, another places where Machine Learning techniques can be used are: 1) to replace the inner two current loops and decoupling block with a neural network, and 2) in flux estimation. The replacement of inner two current loops with a neural network has potential to outperform the PI controller based current controllers in case of lower sampling frequencies.

1.7 Objectives of the Project

This project aims to:

- Replace the conventional SVM technique with a modified hybrid zones technique. Then replace this hybrid zones SVM technique with a neural network and obtain equivalent performance.
- Replace the inner two current loops along with the decoupling block by a neural network to improve the controller as well as system performance at lower sampling frequencies.

During the course of project, different research papers were read and a literature survey table for the same is shown in Table 1.2.

Table 1.2: Literature Survey of Important References

Reference Number	**Description**
1.	• This reference presents ideas about how overmodulation range can be incorporated into SVPWM to increase DC Bus Utilization. • The different modes of OM are presented, namely, OM-I, OM-II and six-step mode.
2.	• This reference presents ideas of how a hybrid zones SVM method can be used to reduce ripple and THD in machine currents waveforms. • Equations for stator flux rms error due to SVM are also presented.
3.	• This reference presents how ANN can be used to generate conventional SVM logic. • ANN only generates vector timings but not switch timings in this case.
4.	• This reference also shows about how a different architectured ANN can be used to generate conventional SVM logic.
5.	• This reference presents information about how ANN can be used to replace flux estimation. • Idea of how PI controllers can be replaced by ANN is also presented.
6.	• This reference uses a sophisticated RNN to perform vector control. • Inner two loops in vector control are replaced by RNN. • The logic used here is FATT which tries to make error zero while my project assumes the current response to be first order response.

Chapter 2

Hybrid Zones SVM Technique

2.1 Introduction

As discussed in Chapter 1 about the use of different switching techniques in order to further improve the SVM technique, this chapter presents a hybrid zones methodology for determining the efficient switching sequences as per the position of $\vec{V_{ref}}$. In order to measure the performance of a switching technique, its error calculation is needed. Stator flux errors are a good measure for these switching sequence errors. Different alternative switching sequences must be compared for their errors and the best one is to be chosen for every position of $\vec{V_{ref}}$ in space.

This chapter first shows the equations for error in fluxes followed by the discussion of three and five zone hybrid SVM techniques. A MATLAB Simulink model was then prepared to verify the results and important observations were made.

2.2 RMS Flux Equations for Different Switching Sequences[2]

RMS error values are a good measure of the *loss* a switching technique faces. The rms flux error equations for different switching sequences, namely 0127, 0121, 7212, 1012 and 2721 are given by following equations:

For 0127 Switching Sequence:

$$\Delta\phi_{rms,0127} = \left[\frac{1}{3}\left(0.5Q_z\right)^2\frac{T_z}{2T} + \frac{1}{3}\left[\left(0.5Q_z\right)^2 + 0.5Q_z\left(0.5Q_z + Q_1\right) + \left(0.5Q_z + Q_1\right)^2\right]\frac{T_1}{T} + \frac{1}{3}\left[\left(0.5Q_z + Q_1\right)^2 - \left(0.5Q_z + Q_1\right)0.5Q_z + \left(-0.5Q_z\right)^2\right]\frac{T_2}{T} + \frac{1}{3}\left(-0.5Q_z\right)^2\frac{T_z}{2T} + \frac{1}{3}D^2\frac{T_1 + T_2}{T}\right]^{1/2} \quad \ldots \quad (2.1)$$

For 0121 Switching Sequence:

$$\Delta\phi_{rms,0121} = \left[\frac{1}{3}0.5Q_z^2\frac{T_z}{T} + \frac{1}{3}\left[Q_z^2 + Q_z\left(Q_z + 0.5Q_1\right) + \left(Q_z + 0.5Q_1\right)^2\right]\frac{T_1}{2T} + \frac{1}{3}\left[\left(Q_z + 0.5Q_1\right)^2 - \left(Q_z + 0.5Q_1\right)0.5Q_1 + \left(-0.5Q_1\right)^2\right]\frac{T_2}{T} + \frac{1}{3}\left(-0.5Q_1\right)^2\frac{T_1}{2T} + \frac{1}{3}\left(0.5D\right)^2\frac{T_1+T_2}{T}\right]^{1/2} \quad \ldots \quad (2.2)$$

For 7212 Switching Sequence:

$$\Delta\phi_{rms,7212} = \left[\frac{1}{3}0.5Q_z^2\frac{T_z}{T} + \frac{1}{3}\left[Q_z^2 + Q_z\left(Q_z + 0.5Q_2\right) + \left(Q_z + 0.5Q_2\right)^2\right]\frac{T_2}{2T} + \frac{1}{3}\left[\left(Q_z + 0.5Q_2\right)^2 - \left(Q_z + 0.5Q_2\right)0.5Q_2 + \left(-0.5Q_2\right)^2\right]\frac{T_1}{T} + \frac{1}{3}\left(-0.5Q_2\right)^2\frac{T_2}{2T} + \frac{1}{3}\left(0.5D\right)^2\frac{T_1+T_2}{T}\right]^{1/2} \quad \ldots \quad (2.3)$$

For 1012 Switching Sequence:

$$\Delta\phi_{rms,1012} = \left[\frac{1}{3}0.5Q_1^2\frac{T_1}{2T} + \frac{1}{3}\left[0.5Q_1^2 + 0.5Q_1\left(0.5Q_1 + Q_z\right) + \left(0.5Q_1 + Q_z\right)^2\right]\frac{T_z}{T} + \frac{1}{3}\left[\left(0.5Q_1 + Q_z\right)^2 - \left(0.5Q_1 + Q_z\right)Q_2 + \left(-Q_2\right)^2\right]\frac{T_1}{2T} + \frac{1}{3}\left(-Q_2\right)^2\frac{T_2}{T} + \frac{1}{3}\left(0.5D\right)^2\frac{T_1}{2T} + \left(0.5D\right)^2\frac{T_z}{T} + \frac{1}{3}\left[\left(0.5D\right)^2 + \left(0.5D\right)D + D^2\right]\frac{T_1}{2T} + \frac{1}{3}D^2\frac{T_2}{T}\right]^{1/2} \quad \ldots \quad (2.4)$$

For 2721 Switching Sequence:

$$\Delta\phi_{rms,2721} = \left[\frac{1}{3}0.5Q_2^2\frac{T_2}{2T} + \frac{1}{3}\left[0.5Q_2^2 + 0.5Q_2\left(0.5Q_2 + Q_z\right) + \left(0.5Q_2 + Q_z\right)^2\right]\frac{T_z}{T} + \frac{1}{3}\left[\left(0.5Q_2 + Q_z\right)^2 - \left(0.5Q_2 + Q_z\right)Q_1 + \left(-Q_1\right)^2\right]\frac{T_2}{2T} + \frac{1}{3}\left(-Q_1\right)^2\frac{T_1}{T} + \frac{1}{3}\left(0.5D\right)^2\frac{T_2}{2T} + \left(-0.5D\right)^2\frac{T_z}{T} + \frac{1}{3}\left[\left(-0.5D\right)^2 + \left(-0.5D\right)\left(-D\right) + \left(-D\right)^2\right]\frac{T_2}{2T} + \frac{1}{3}D^2\frac{T_1}{T}\right]^{1/2} \quad \ldots \quad (2.5)$$

where,

$$Q_1 = \left[cos\theta - V_{ref}\right]T_1$$

$$Q_2 = \left[cos(60^o - \theta) - V_{ref}\right]T_2$$

$$Q_z = -V_{ref}T_z$$

$$D = sin\theta T_1$$

The above equations are depicting error for over a sampling interval T_s. As can be seen, these equations would have non-zero values for almost all samples of $\vec{V_{ref}}$ taken and in a way depicts the amount of error that SVM causes in that sample of $\vec{V_{ref}}$.

2.3 Hybrid Zones SVM Method

2.3.1 3 Zone Hybrid SVM

The above equations for rms error in different switching sequences can be used to get which switching sequence is most efficient depending on the position of $\vec{V_{ref}}$ in the $\alpha - \beta$ plane. A comparision among the three discussed switching sequences at different positions of $\vec{V_{ref}}$ and choosing the one with the least error result in the following observation:

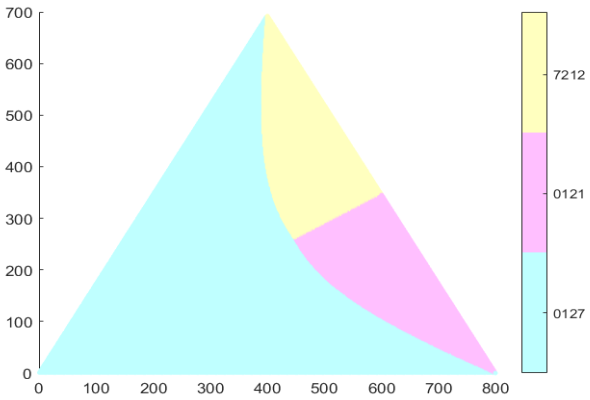


Figure 2.1: Sector Diagram Depicting Dominant Zones for Three Zone Hybrid SVM

From the figure it can be seen that there are regions inside a sector in which a single type of switching sequence results in the least error (in terms of THD). Hence when $\vec{V_{ref}}$ rotates inside the hexagon in a sector, depending on which region the tip of $\vec{V_{ref}}$ lies, that sequence is applied to generate $\vec{V_{ref}}$. The sequence which gives the least error is said to be dominant sequence in that region.

As can be seen from the above figure that no switching sequence is entirely dominant over all sector, rather each sequence is dominant over a portion of a sector. Hence, in order for most efficient operation, all the three sequences must be used. This type of technique of SVM is called Hybrid Zone Technique. Since there are three competing switching sequences, hence it is called Three Zone Hybrid SVM Technique.

The three zone hybrid SVM technique works as follows. As $\vec{V_{ref}}$ rotates in the hexagon (in linear modulation), it is sampled at start of each sampling period. This sampled $\vec{V_{ref}}$ is then generated by first determining the on duration of different vectors followed by the determination of a suitable switching sequence. The switching sequence is selected to be the dominant sequence at the place of tip of $\vec{V_{ref}}$.

For overmodulation regions, when $\vec{V_{ref}}$ does not get clamped and rotates purely as it should, then the same technique is followed as stated above. When $\vec{V_{ref}}$ gets clamped to either of adjacent state vectors or to the hexagon side, then the switching sequences are determined according to this new position of $\vec{V_{ref}}$. From Fig 2.1 it can be seen that in such "clamped overmodulation" situations, only 0121 or 7212 sequences would be applied. Moreover, if θ is less than $\pi/6$ then, 0121 gets applied otherwise 7212 sequence gets applied.

2.3.2 5 Zone Hybrid SVM

The above arguments can be extended to incorporate 5 number of possible switching sequences, three same as in three zone hybrid SVM method and two other sequences, namely 1012 and 2721. The dominant zones were found as previously stated, but more number of alternatives were there. The plot of dominant zones for 5 zone hybrid method comes as following plot.

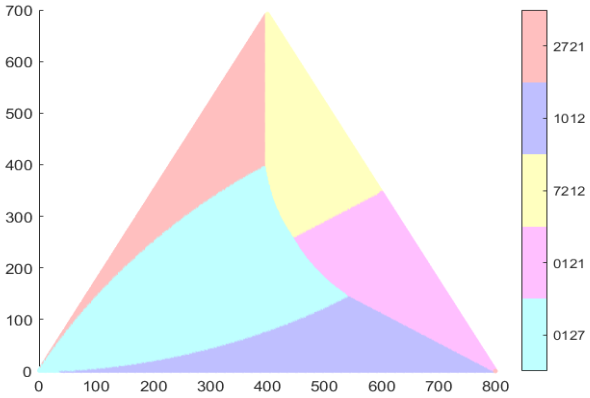


Figure 2.2: Sector Diagram Depicting Dominant Zones for Five Zone Hybrid SVM

The obvious purpose of incorporating 5 zones instead of 3 zones is that it will lead to even further reduction in THD of the system. The 5 zone method is better than 3 zone because more number of alternatives are there to choose the best alternative from.

The three and five zone hybrid SVM techniques are better than conventional SVM technique. These techniques were individually used in the "Space Vector PWM Based VSI" block in the Simulink model as shown in Appendix section. Another temporary Simulink model using conventional SVM technique was built to verify the performance improvement. The motor input current's THD was measured at different modulation indexes. The observed results (at frequency of 50Hz) are shown in Table 2.1.

It can be seen that the THD values in motor input current has significantly reduced in three and five zone hybrid techniques with five zone technique giving even better results. Next, THD values were compared against the variation of frequency. Fig 2.3 shows the results obtained.

Table 2.1: THD Comparison for Hybrid SVM (Frequency = 50Hz)

Number of Zones	**Current THD% (m=0.98)**	**Current THD% (m=1.02)**	**Current THD% (m=1.08)**
1.	3.1467	5.4267	17.043
3.	2.0867	4.3567	16.753
5.	2.0767	4.403	16.753

Frequency	THD (1 zone)%	THD (3 zone)%	THD (5 Zone)%
10	12.1233	12.1233	11.7767
15	4.1067	4.1067	2.7267
20	6.1767	6.1767	6.7067
25	2.7267	2.7267	2.9767
30	5.07	5.173	4.4267
35	4.56	2.78	3.03
40	3.603	2.76	2.8267
45	3.64	3.42	2.65
50	3.5967	2.46	2.67
50.5	4.69	3.32	3.173
51	5.273	4.0567	4.08
51.5	5.99	5.36	5.53
52	6.86	6.2	6
52.5	24.92	24.8567	24.8567
53	21.34	21.0367	21.0367
53.5	18.73	18.5167	18.5167
54	16.81	16.5433	16.5433
54.5	15.367	14.9867	14.9867
55	14.4467	13.653	13.6533

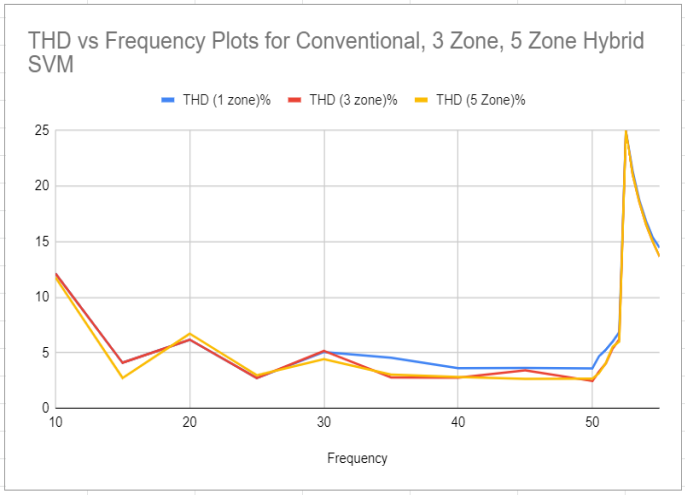


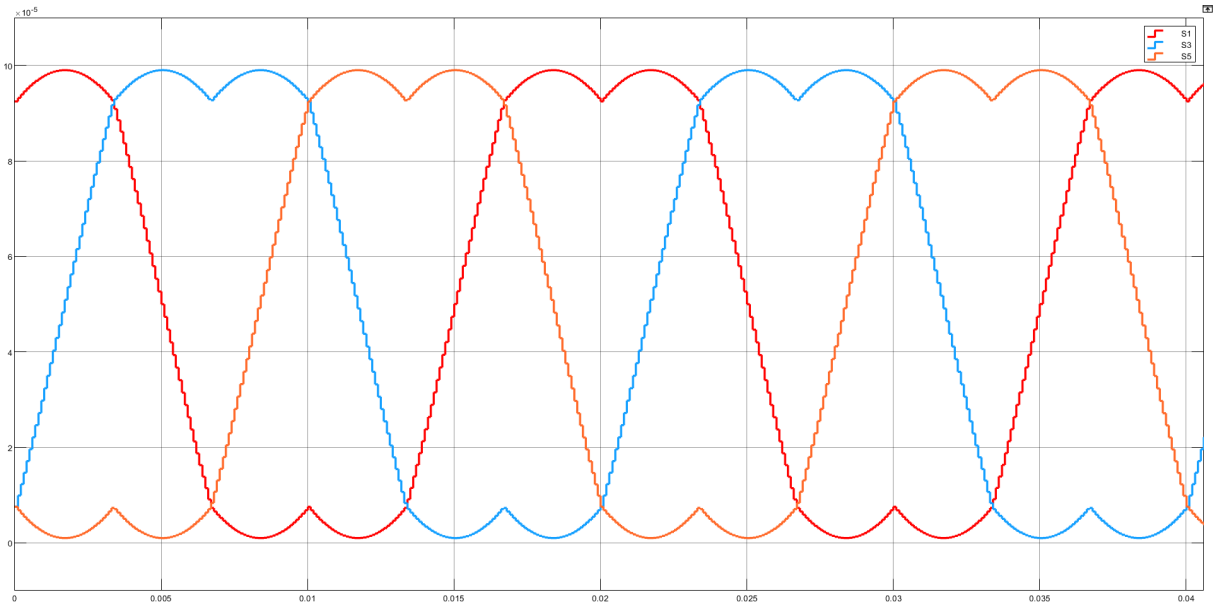
Figure 2.3: THD vs Frequency for Conventional, Three and Five Zone Hybrid SVM

V/f ratio was kept constant always. It can be seen from previous figure that the conventional SVM technique never gives lesser THD than the three and five zone hybrid techniques (Although it might give same THD as three zone hybrid technique sometimes because of low m). This further proves that the performance of SVM has improved.

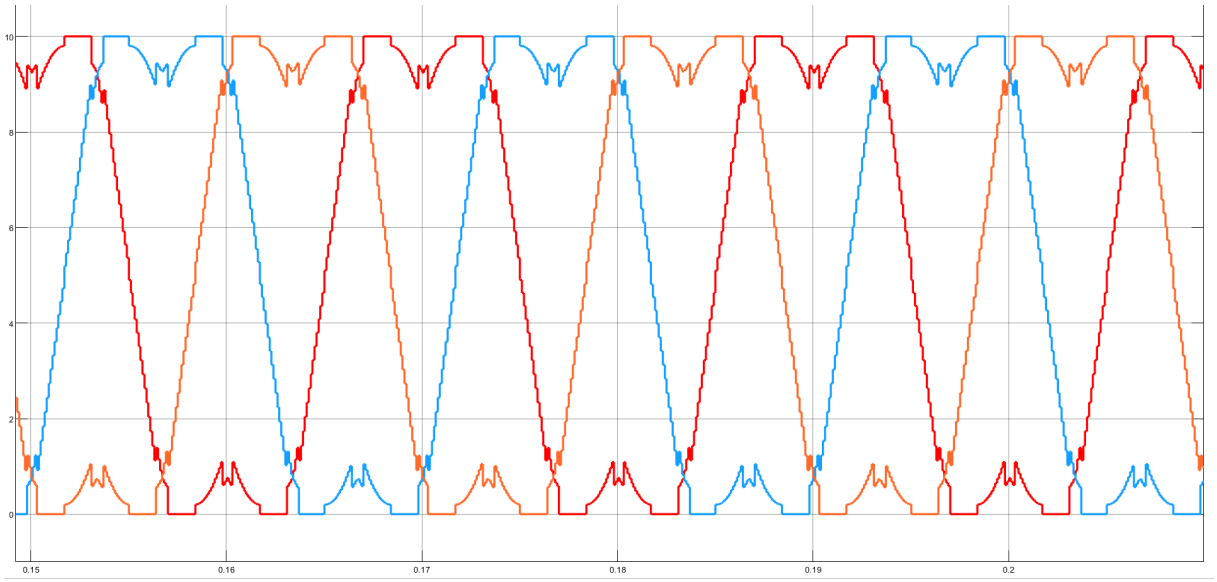
2.4 Resultant Waveforms

Switch ON Duration

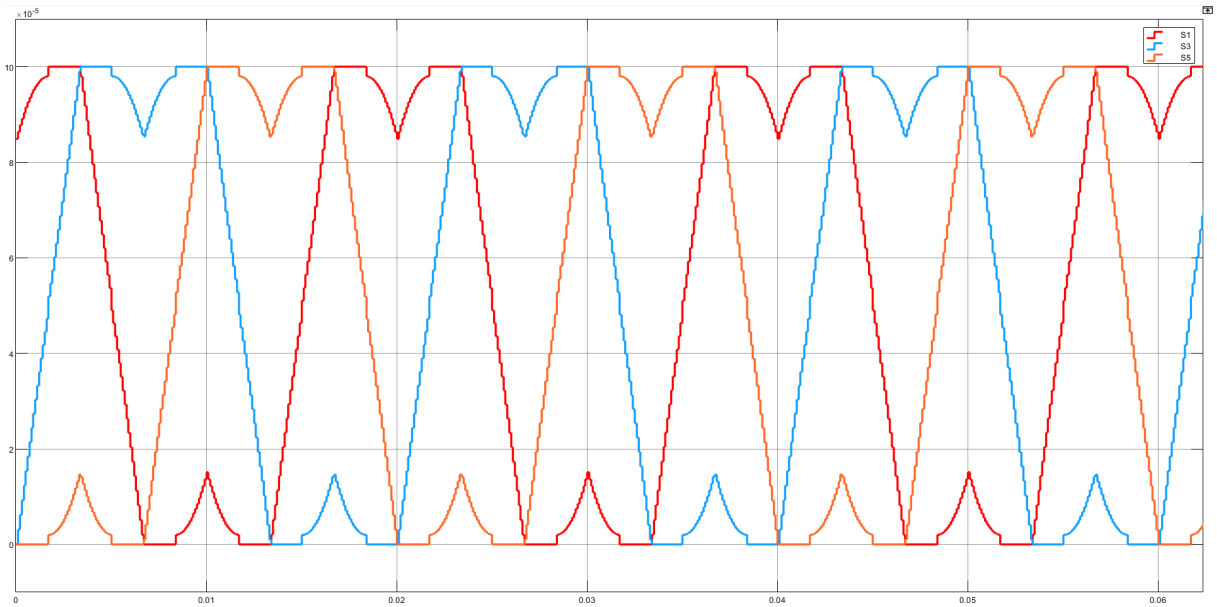
The switch ON durations were measured to check whether the system is working correctly and to observe patterns from it. Fig 2.4, Fig 2.5 and Fig 2.6 shows switch ON duration plots for different modulation indexes.



(a) For Conventional SVM

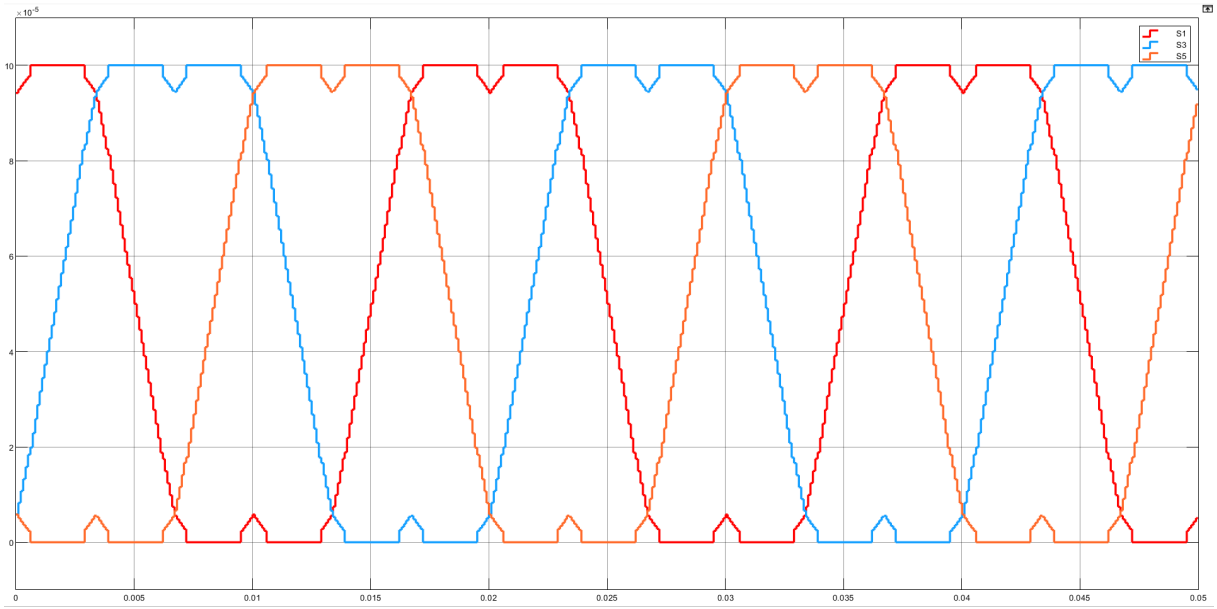


(b) For 3 Zone Hybrid SVM

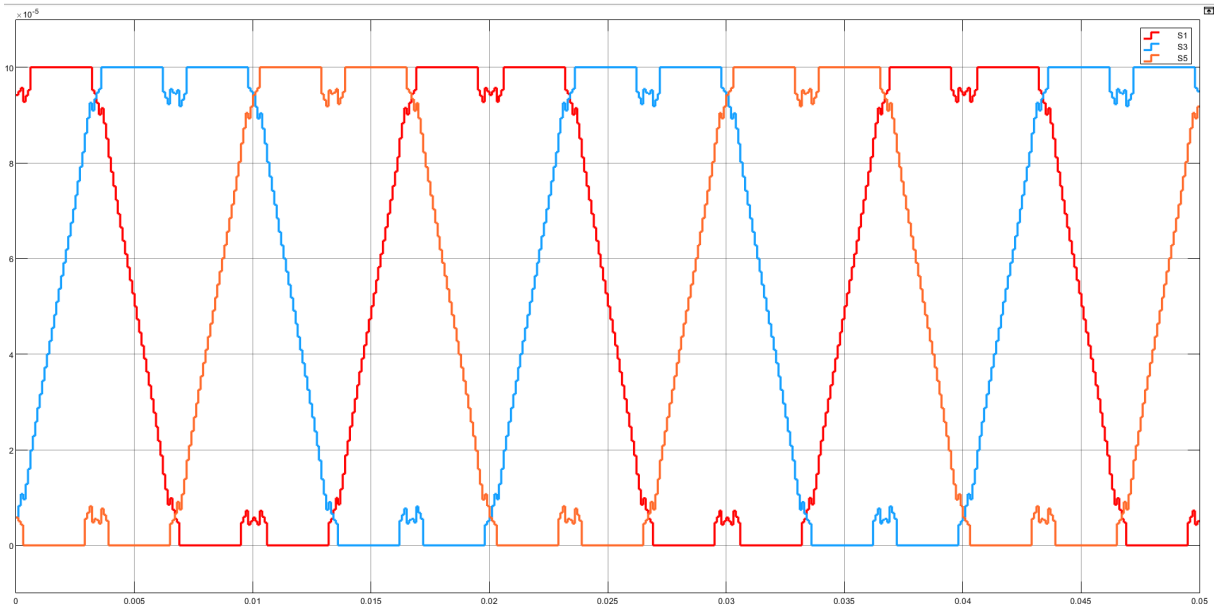


(c) For 5 Zone Hybrid SVM

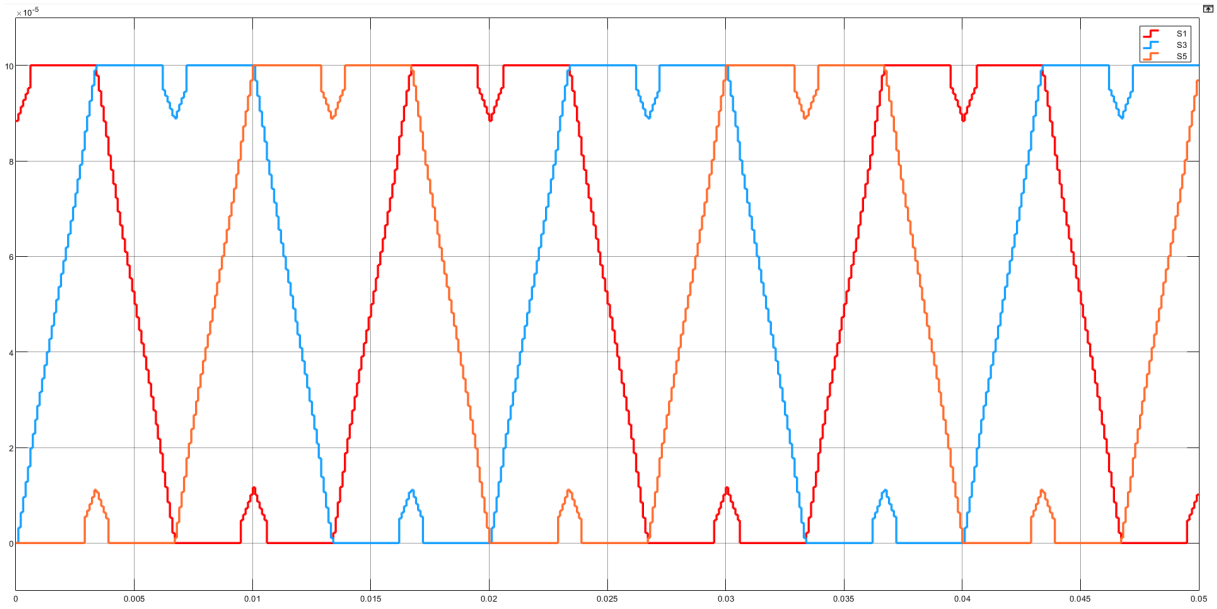
Figure 2.4: Switch ON Durations for $m = 0.98$



(a) For Conventional SVM

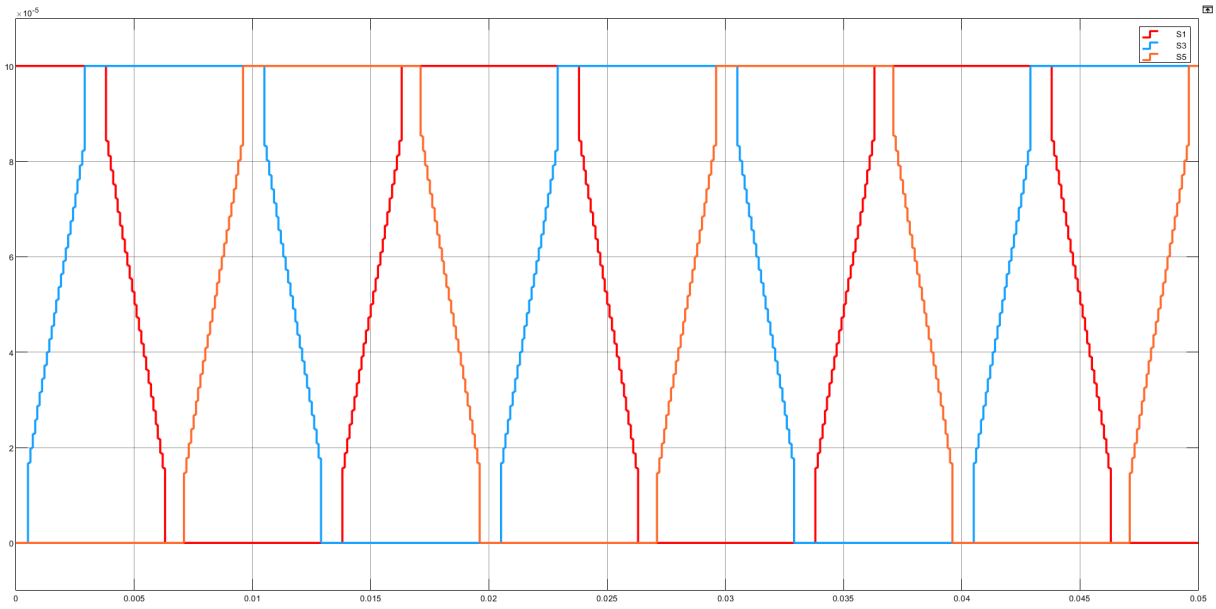


(b) For 3 Zone Hybrid SVM

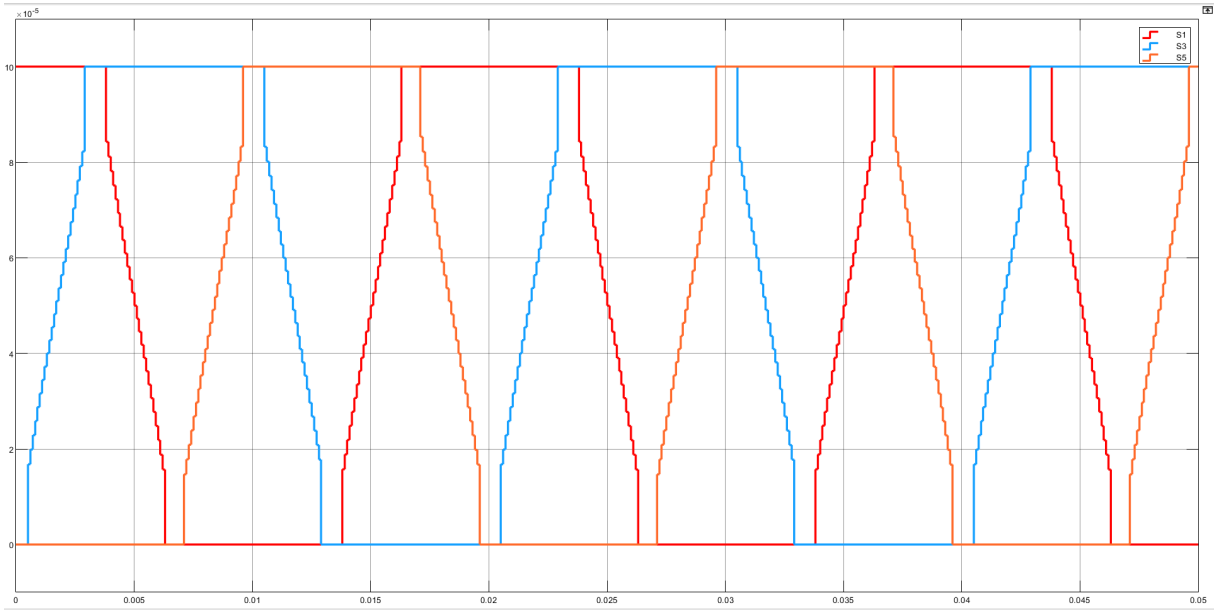


(c) For 5 Zone Hybrid SVM

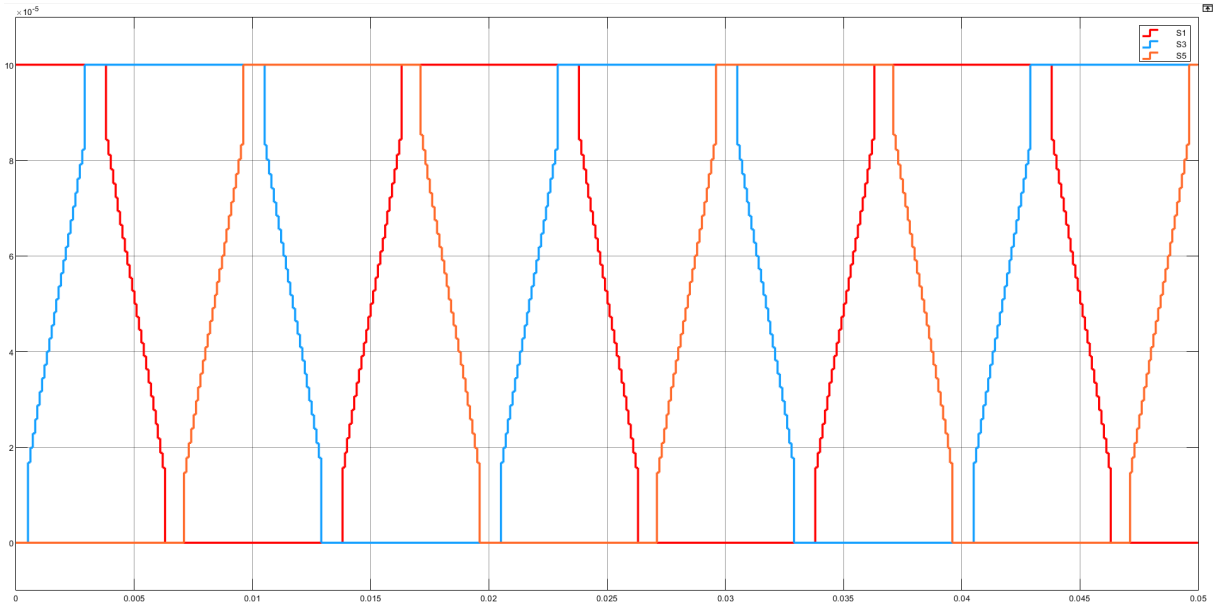
Figure 2.5: Switch ON Durations for $m = 1.02$



(a) For Conventional SVM



(b) For 3 Zone Hybrid SVM



(c) For 5 Zone Hybrid SVM

Figure 2.6: Switch ON Durations for $m = 1.08$

Gate Pulses, Switching Sequence and Sector Plots

The gate pulses along with the most efficient switching sequence and sector were plotted and observed to verify that the switching logic is correct. The following plot was observed.

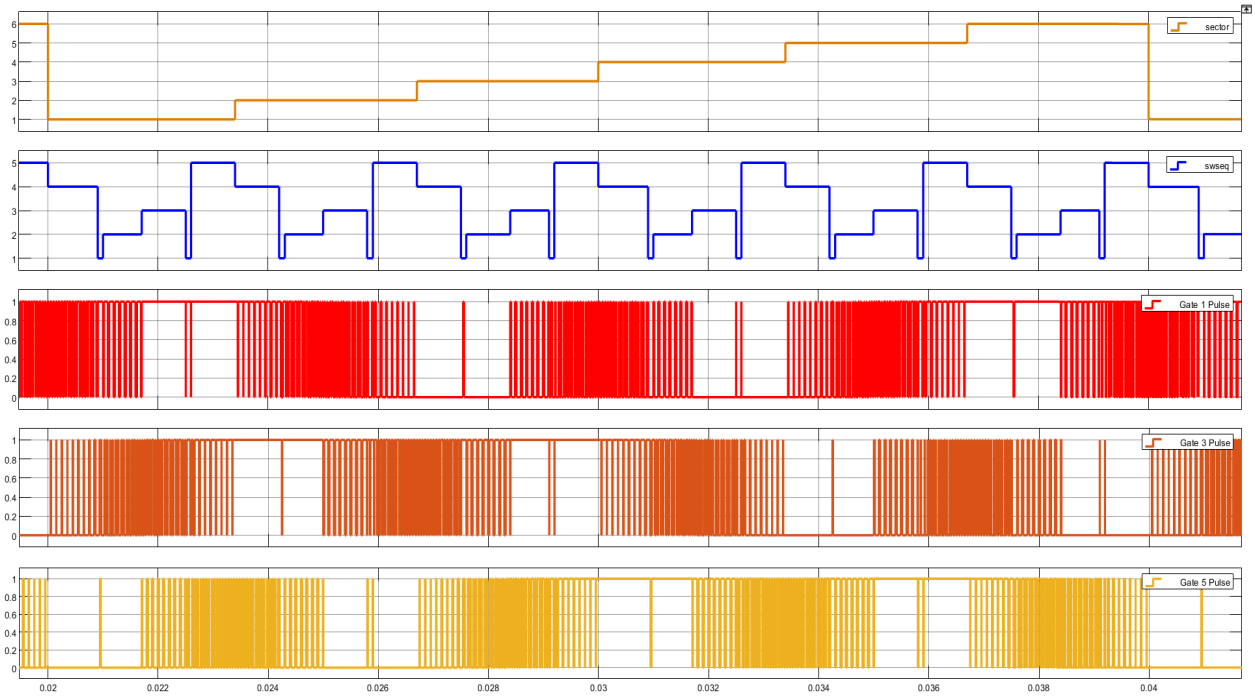


Figure 2.7: Gate Pulses and Switching Sequence Waveform ($m = 0.8$)

It was verified that the gate pulse pattern is coming in accordance with the switching sequence to be followed. The modulation index m was chosen to be such that all the five switching sequence happens at least once when $\vec{V_{ref}}$ rotates in $\alpha - \beta$ domain.

Motor Speed, Torque and Input Currents

The motor speed would eventually settle to a constant value (with a high frequency ripple) if $\vec{V_{ref}}$ gets properly generated due to SVM logic. The plots obtained are shown in Fig 2.8.

It can be seen that the motor speed settles to some constant value after some time. This was also seen when rated load torque was applied at 0.5s. This suggests that the SVM technique is predicting switching times correctly. The motor speed has a high frequency ripple though, which is expected due to non-ideal sinusoidal input from VSI. The motor input currents are all sinusoidal, as expected from a proper functioning SVM logic.

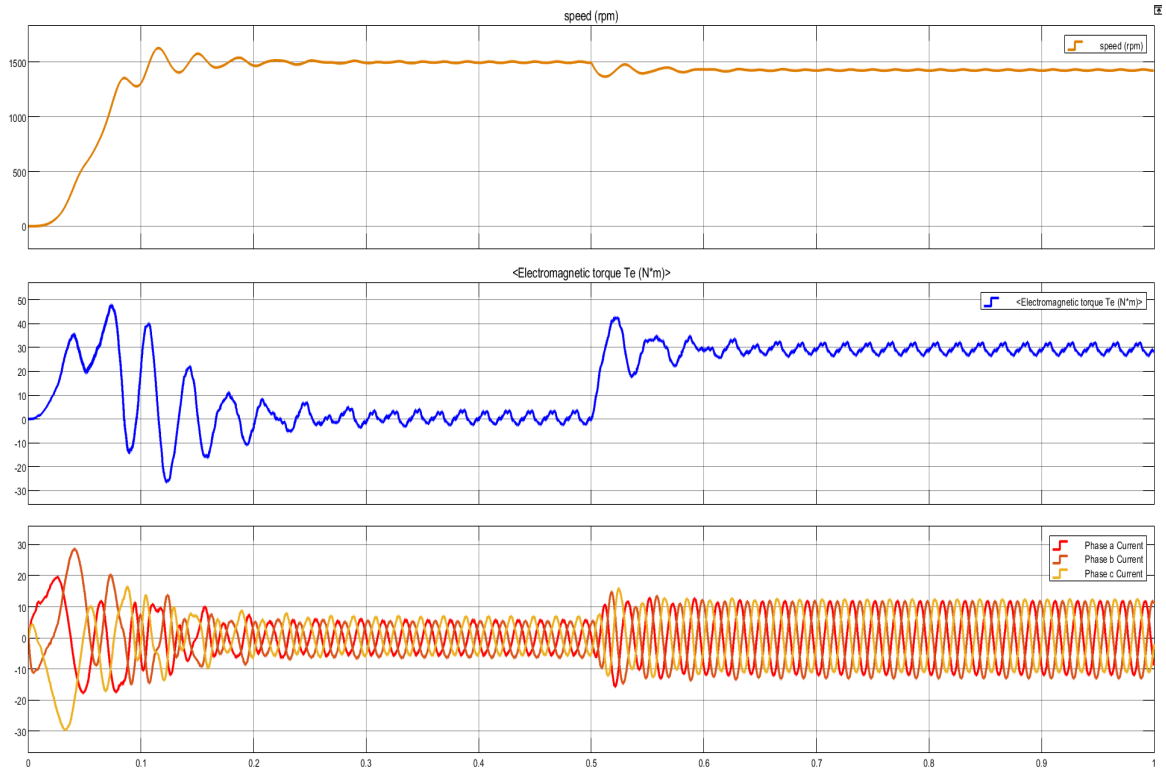


Figure 2.8: Motor Speed, torque and Motor Currents Waveforms

From the above results,it can be said that the hybrid zones SVM technique is working correctly and we can proceed for generating data for neural network training.

2.5 Computational Complexity

The computational complexity of the hybrid SVM techniques were then measured for different modulation index ranges. Fig 2.9 shows the number of basic mathematical operations that were taking place for determining just the switch timings. Other operations like variable assignments and number of comparisons were also measured. The total number of operations are shown in the last row of the tables.

On an average the SVM technique implementation had about 100-150 operations required for determining the switch timings. The MATLAB code for the same took about 250 lines. This all code gets to run serially one after another for computing switching times in this type of implementation of SVM. This causes delay in predicting values. Whereas using a neural network with let's say 2 hidden layers causes about 4-8 amount of serialized code. With parallelized computing possible today, this causes much lesser delay in computation time and more time can be given for incorporating complex switching sequences.

For Linear Modulation					
	Additions	Multiplications	Lookups	Additional Operations	Comparisions
Calculate theta	1	1	0	0	0
Calculate T1, T2, T0	5	12	2	6	1
Q1, Q2, Q0, D	3	7	3	4	1
F_0127	13	38	0	1	0
F_0121	13	34	0	1	0
F_7212	13	34	0	1	0
switchseq	0	0	0	3	4
Gmax, Gmed, Glow	3	4	1	5	2
S1, S3, S5	6	6	0	17	9
Total	**57**	**136**	**6**	**38**	**17**

(a) For Linear Modulation

For Overmodulation Zone 1 (Vref moves along original trajectory)					
	Additions	Multiplications	Lookups	Additional Operations	Comparisions
Calculate theta	1	1	0	0	0
Calculate T1, T2, T0	5	14	4	6	4
Q1, Q2, Q0, D	3	7	3	4	1
F_0127	13	38	0	1	0
F_0121	13	34	0	1	0
F_7212	13	34	0	1	0
switchseq	0	0	0	3	4
Gmax, Gmed, Glow	3	4	1	5	2
S1, S3, S5	6	6	0	17	9
Total	**57**	**138**	**8**	**38**	**20**

(b) When $\vec{V_{ref}}$ moves along original trajectory in OM mode 1

For Overmodulation Mode I &II (when Vref moves along hexagon side)					
	Additions	Multiplications	Lookups	Additional Operations	Comparisions
Calculate theta	1	1	0	0	0
Calculate T1, T2, T0	6	3	4	5	4
switchseq	0	0	0	1	2
Gmax, Gmed, Glow	1	2	1	5	2
S1, S3, S5	6	6	0	17	9
Total	**14**	**12**	**5**	**28**	**17**

(c) When $\vec{V_{ref}}$ moves along hexagon side (OM mode 1 and 2)

For Overmodulation Mode II (when Vref gets clamped to nearest vector)					
	Additions	Multiplications	Lookups	Additional Operations	Comparisions
Calculate theta	1	1	0	0	0
Calculate T1, T2, T0	1	1	1	4	6
switchseq	0	0	0	1	2
Gmax, Gmed, Glow	1	2	1	5	2
S1, S3, S5	6	6	0	17	9
Total	**9**	**10**	**2**	**27**	**19**

(d) When $\vec{V_{ref}}$ gets clamped to nearest vector (OM 2)

Figure 2.9: Operations Count of SVM Technique Under Various Modes

Chapter 3

Neural Network Based SVM

Now that the concept of why to use machine learning techniques is clear, a ML model is to be created and trained. This chapter first shows how training data was formed, followed by why neural network was needed and finally by finalizing the NN architecture, training and validating it. Note that ML model for only three zone hybrid method was created.

3.1 Training Dataset Creation

Before creating a neural network, the training dataset which the neural network will use to learn patterns, needs to be created. The three zone hybrid technique discussed only requires the position of $\vec{V_{ref}}$ in $\alpha - \beta$ plane to calculate switching timings as well as the dominant switching sequence. A phasor is fully defined by its magnitude and phase. Hence these two quantities are the only independent quantities required to predict the outputs. The output values would be the three switch timings plus the switching sequence.

Hence the training dataset had 2+4=6 columns and 100,000 observations were taken. Also, the sector information was saved in the dataset although not required, for testing purposes. The first few rows of the dataset looked like the following:

m	angle	sector	T1/Ts	T2/Ts	T0/Ts	switchseq
0.671694	0.083924	1	0.803905	0.2524	0.196095	1
1.077585	2.917395	3	0	1	0.767325	3
0.751648	3.234408	4	0.158512	0.771824	0.841488	1
0.529469	3.387929	4	0.245376	0.625512	0.754624	1
0.28878	1.850297	2	0.431006	0.638787	0.361213	1
0.286047	1.083467	2	0.616001	0.626374	0.373626	1
0.541815	5.798631	6	0.770701	0.229299	0.481684	1
1.107074	1.367706	2	1	1	0	2
0.495469	5.949344	6	0.743288	0.256712	0.419065	1
1.061187	2.670635	3	0	1	0.545631	3

Figure 3.1: Training Dataset

The above training dataset was created using a MATLAB function which had code to implement the logic for determining switching times and sequence. Note that T1, T2 and T0 are all stored normalized with respect to switching time period T_s. Also, the switching sequences, in order to be used by ML or NN models, were given numerical values (1-5 for switching sequences 0127, 0121, 7212, 1012, 2721 respectively).

3.2 Regression and Classification Models

For simplicity of ML model, it was first tested whether simple regression(for switching times) or classification(for switching sequence) algorithms could perform well. Fig 3.2 shows the different ML models which were trained on the earlier created dataset, along with their final performance.

Training Loss	**Loss**	**mse**
Regression		
Linear svm	0.1196	0.348
Gaussian SVM	0.0044	0.07
polynomial svm (order 3)	0.0238	0.1578
polynomial svm (order 5)	0.0122	0.1177
GPR(quasinewton optimizer)	**0.00071**	**0.0292**
GPR(lbfgs optimizer)	0.000827	0.0279
GPR(fminsearch optimizer)	0.000834	0.0316
Classification	**accuracy on train set**	**accuracy on test set**
linear svm	69.83%	68.55%
gaussian svm	79.15	79%
naïve bayes	71.07%	70%
naive bayes with hyperparameter optimization	**94.94%**	**94.65%**

Figure 3.2: Performance of Different ML Models

The best performing models are highlighted in the above figure. The accuracy and final loss of these classification and regression models were not too satisfactory plus in the future, the models would be more complex because of increase in number of zones and other factors too. A neural network would best suit for this purpose because of its multiple layer structure. Hence from hereon, a NN was developed and trained.

3.3 Neural Network Model

Neural networks had to be created in order to automatically calculate switching times as well as the switching sequence. In this project two separate neural networks were created having same architecture, one for regression outputs (the three switch timings) and other for classification output (the switching sequence). The following subsections describe the neural network in more detail.

3.3.1 Neural Network Architecture

Various architectures were tried to make an accurate neural network. The different architectures tried are shown below:

				For Regression Outputs:		For Classification Outputs:	
Feature Expansion	No. of Hidden Layers	Size of hidden layers	Activation Function for Hidden Layers	Train Set MSE	Test Set MSE	Train Set accuracy	Test Set accuracy
NA	1	6	relu	0.1453	0.1462	69.74	70.9
NA	1	6	tanh	0.0862	0.084	72.64	72.7
NA	1	10	relu	0.124	0.1195	72.0875	72.1
NA	1	10	tanh	0.0751	0.0755	74.2125	72.55
NA	1	20	tanh	0.069	0.0689	72.825	72.75
NA	2	6, 6	tanh	0.067	0.0654	70.7625	69.7
NA	2	10, 6	tanh	0.0697	0.0693	75.7	74.8
NA	**2**	**6, 10**	**tanh**	**0.0585**	**0.0586**	76.6625	77
NA	2	10, 10	tanh	0.0612	0.0656	78.3125	76.75
upto 2nd order	1	6	tanh	0.0732	0.0758	73.625	74.35
upto 2nd order	1	10	tanh	0.0718	0.0749	80.3625	79.1
upto 2nd order	**1**	**20**	**tanh**	**0.0612**	**0.0616**	**83.9**	**85.05**
upto 2nd order	2	6, 6	tanh	0.0606	0.059	75.875	76
upto 2nd order	2	6, 10	tanh	0.0609	0.0627	80.45	80.35
upto 2nd order	2	10, 10	tanh	NA	NA	82.475	82.8
upto 3rd order	1	6	tanh	0.0883	0.0881	74.3125	73.3
upto 3rd order	1	10	tanh	0.0788	0.0781	78.8125	78.9
upto 3rd order	1	20	tanh	0.0682	0.0685	81.175	80.55
upto 3rd order	2	6, 6	tanh	0.081	0.0809	79.8375	78.9
upto 3rd order	2	6,10	tanh	0.0746	0.0748	84.4375	83.6
upto 3rd order	2	10, 10	tanh	NA	NA	82.8125	80.95
include sine terms	**1**	**10**	**tanh**	**0.055982**	**0.053531**	**94.1825**	**95.65**
include sine terms	**1**	**20**	**tanh**	**0.0536**	**0.054413**	**98.9125**	**99.15**

Figure 3.3: Performance of Different NN Architectures

It was observed that the best performant architecture was when sine of phase angle of $\vec{V_{ref}}$ was also given as input to the model. The error was further minimized by taking two subnetworks (parallel layers) and multiplying them at a later stage. This was done because in the training set creation it was observed that in calculation of switching times, sinusoidal terms (and their products) were present. The final architecture looked like the one shown in Fig 3.5. Some other architectures which were also tried are shown in Fig 3.4.

The hidden layers in each subnets had 20 nodes each. The input and output layers had size of 18(due to feature expansion) and 3 (or 1 for classification) respectively. The activation function used was $tanh$ and for output layer it was $sigmoid/softmax$.

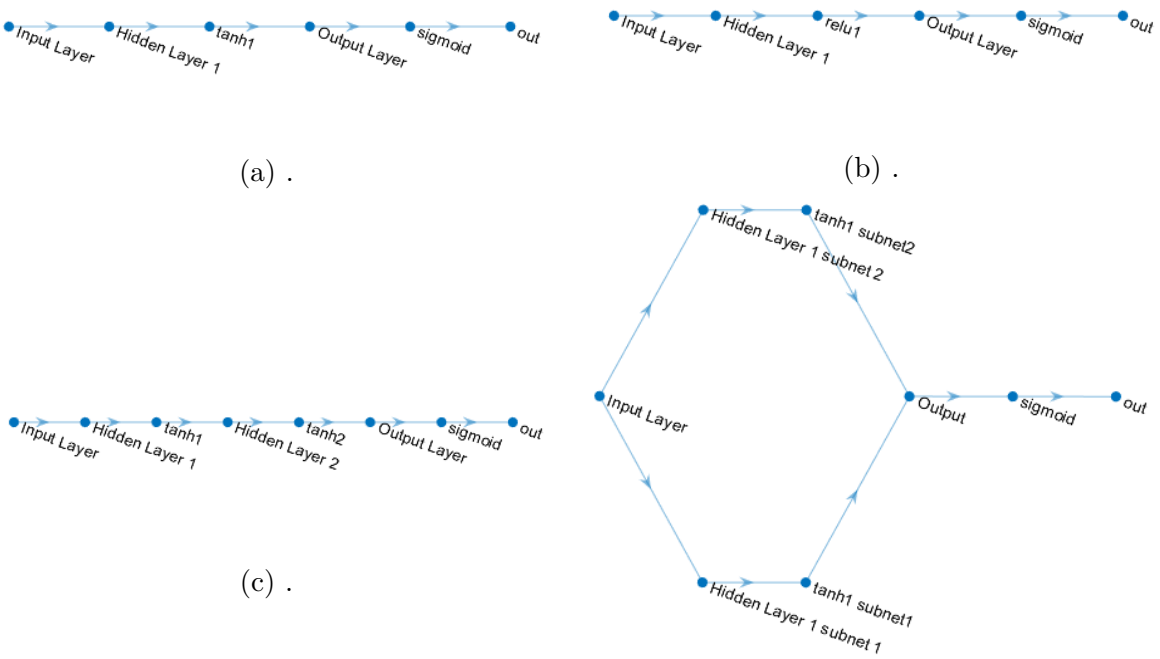


Figure 3.4: Some NN Architectures which were tried on the training data

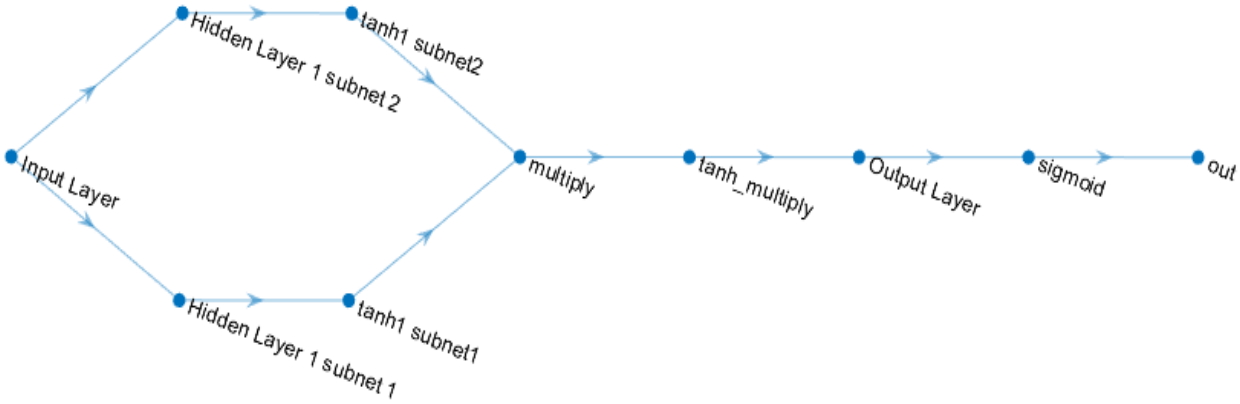


Figure 3.5: Final NN Architecture

3.3.2 Expanding Features for Training the Neural Network

Fig 3.6 depicts the input output relationship of the training data, i.e all the 4 variables (switch times and switching sequence) plotted individually against the two independent variables, m and $angle$.

It was seen that the input output relationships are not linear. Also, there are some non-differential lines in each of the plots which, in order to be recreated using the NN model, needs theoretically infinite number of sinusoidal harmonics. Practically, this is not possible, so harmonics upto a certain order were included to obtain maximum efficiency. It was also observed that odd harmonics only were present in the waveforms.

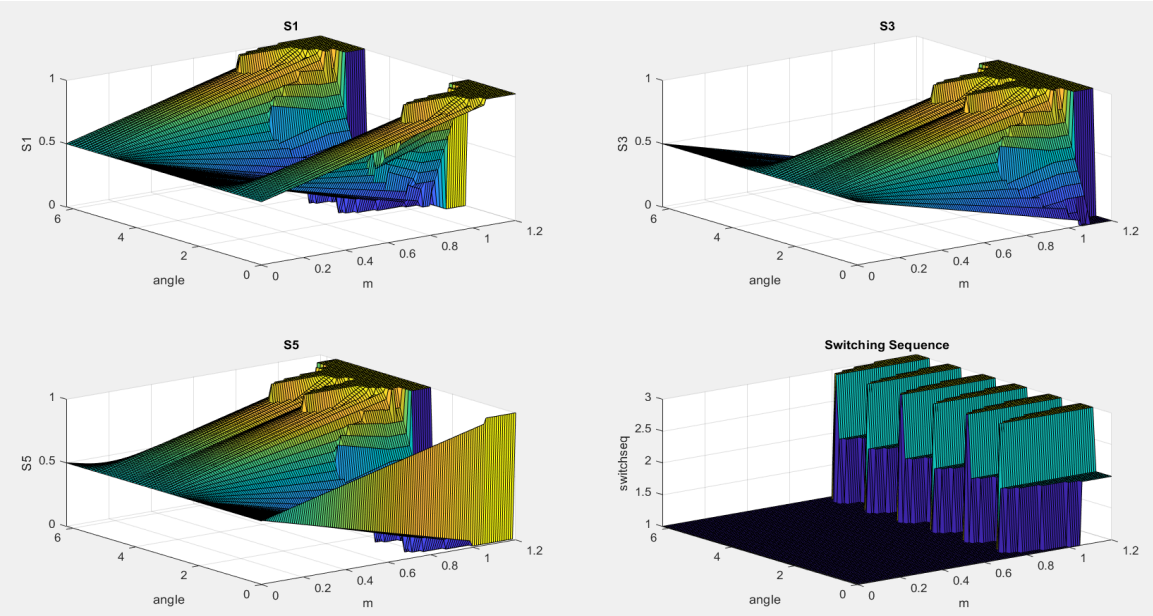


Figure 3.6: Input Output Relationships

These harmonics were provided as an input to the NN model and its performance was checked. The input size became 50 when 23 harmonics were included and this was used in training the NN.

3.3.3 Training the Neural Network

After defining the structure of neural network, it was trained on the training dataset created earlier. MATLAB's Deep Learning and Machine Learning Toolbox were used. For cross-validation purposes, 80% data was given to training and 20% was kept for validation. Different hyperparameters were fine tuned using some normal conventions typically followed. The final set values were as shown in Table 3.1.

Table 3.1: Final Hyperparameters Values for Training NN for SVM

Optimization Used	ADAM
Number of Epochs	4000
Initial Learning Rate	0.05
Learning Rate Decay	0.5 in every 200 iterations
Regularization Factor	0.0000001

For all the stated architectures in Fig 3.3, training was done and final accuracy/loss obtained are shown in the adjoining columns of the figure.

The finalized architecture model was then trained similarly including feature expansion as per section 3.3.2 and the final loss and accuracy of the model were **0.02336** and **99.65%** respectively. The training plots for both models are shown in Fig 3.7 and Fig 3.8.

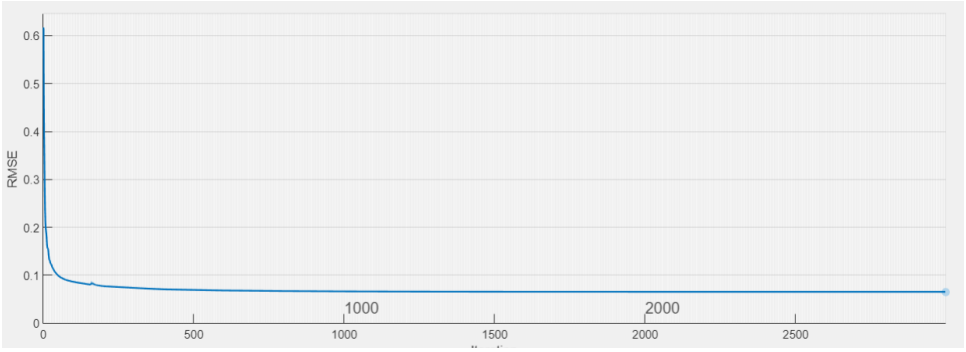


Figure 3.7: Training Progress Plot for Regression Outputs (the 3 Switch Timings)

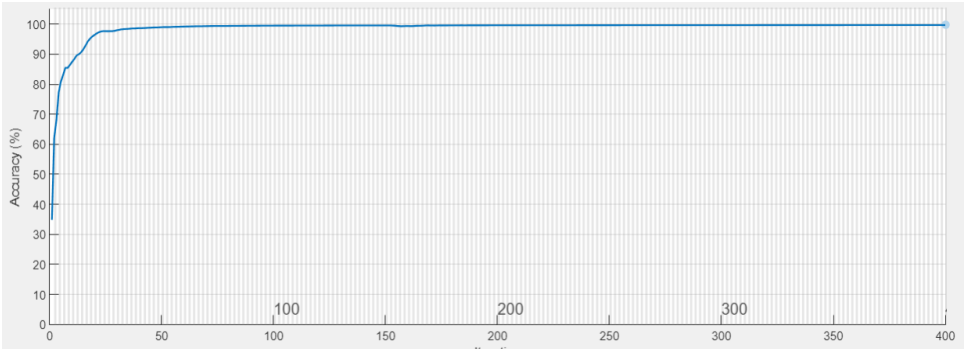


Figure 3.8: Training Progress Plot for Classification Output (Switching Sequence)

3.4 Resultant Waveforms

After NN training was done it was then deployed to the original Simulink model. The details of the Simulink model are given in the Appendix section. Various waveforms were observed to check whether the NN model performed well or not on the real time prediction of timings. Following were the observations made:

Ideal and Generated $\vec{V_{ref}}$

The generated $\vec{V_{ref}}$ was compared against the ideal input $\vec{V_{ref}}$ that was supposed to be generated. Fig 3.9 shows the plots obtained in $\alpha - \beta$ domain.

As can be seen from the figure, the NN models the ideal $\vec{V_{ref}}$ quite accurately. There are some noises somewhere which are expected since the NN cannot be 100%

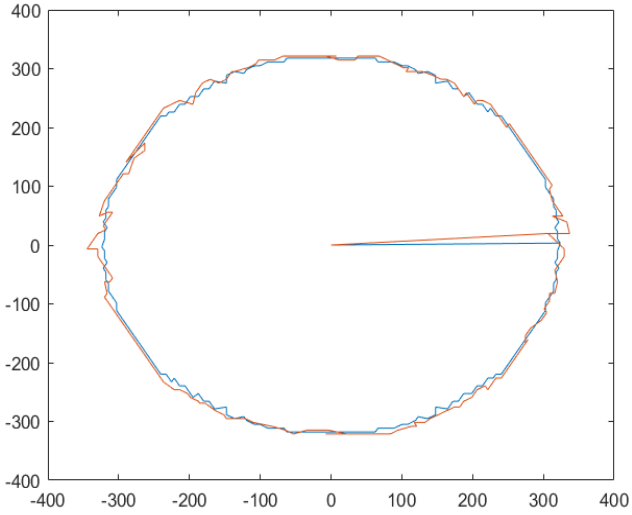
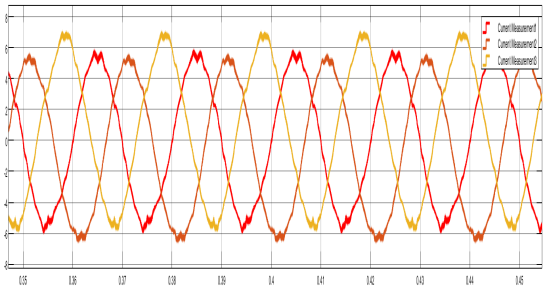


Figure 3.9: Ideal and Generated $\vec{V_{ref}}$ in $\alpha - \beta$ Domain

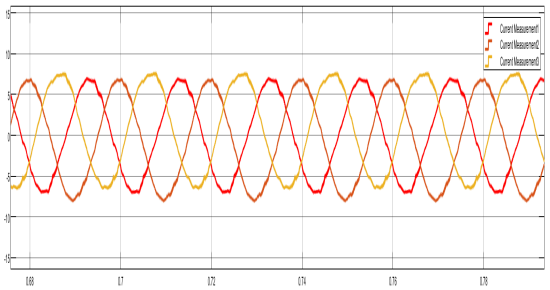
accurate. But the magnitude of these noises are less which is a good indicator of model's performance.

Motor Input Current

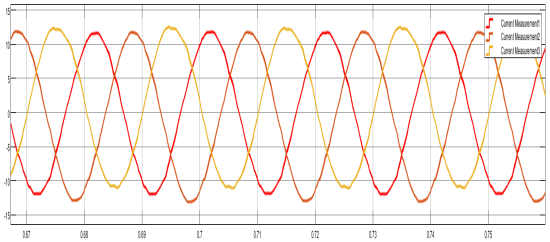
The three phase induction motor's input current THD is a good indicator of how well the NN is performing. The following current waveforms were observed for different loading conditions.



(a) Under No Load



(b) For Load Torque = 0.4*(Rated Torque)



(c) Under Full Load

Figure 3.10: Motor Input Currents for Different Loading Conditions

It can be seen from the above figure that the input currents are almost perfectly sinusoidal, they are having very less THD values under all loading conditions. The THDs were found to be 10.1% under no load, 6% under 0.4×(Rated Torque load) and 4.12% under full load. Also there is a presence of a DC component in a phase, this is because of NN lack of symmetry for different phases.

RMS $|\vec{V_{ref}}|$ Generated

It was checked whether the NN based VSI was able to generate the required rms voltage (which can be completely determined from the value of modulation index m) at the output or not. For example, for $m = 0.97$, the following rms voltage was observed:

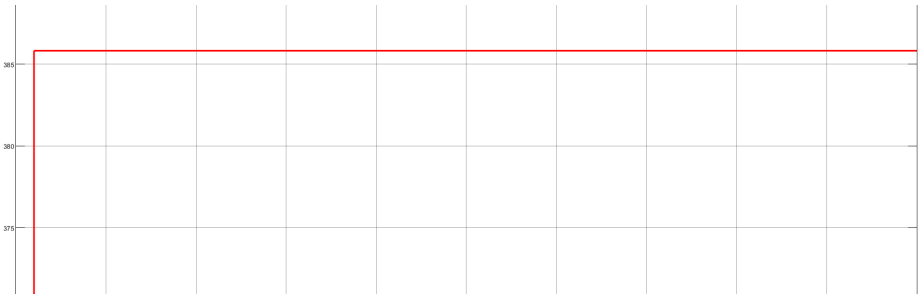


Figure 3.11: RMS Line Voltage Generated

Some important conclusions can be made from the rms voltage plot. Firstly the rms value is constant which means that the NN operated VSI is properly generating voltage. The magnitude of the constant rms was about 385.8V, which ideally (using m) should be 388V. The error in voltage is about 0.56% which is well within the acceptable bounds. The rms values were also verified for other values of m.

Rotor Speed, Torque and Motor Currents

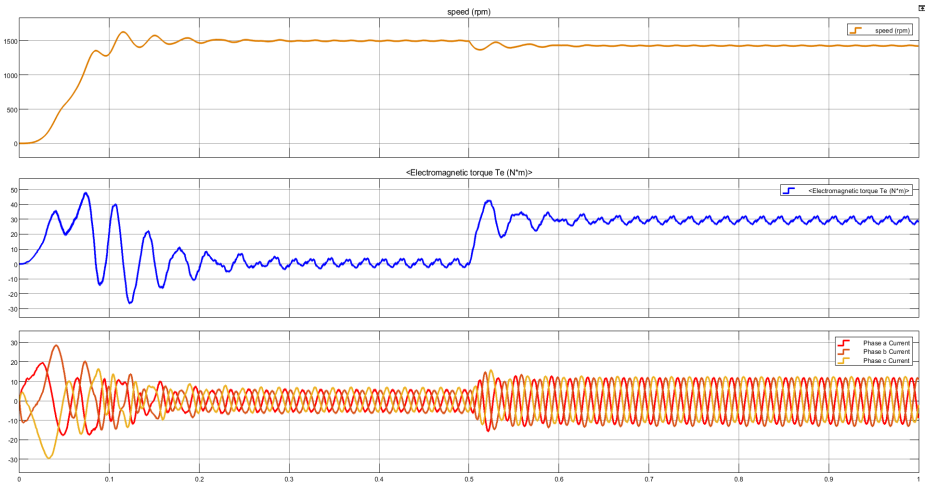


Figure 3.12: Motor Speed Waveform

The rotor speed should ideally become constant after some time, and the constant's value depends on loading. Under rated conditions, motor should operate at rated speed. The motor speed waveform was observed as shown in Fig 3.12.

The motor speed was found to be 1500RPM under no load and about 1430RPM under full load torque (applied at 0.5s time), which is also the rated speed. This suggests that the VSI is operating correctly and is providing sinusoidal input to the motor. The motor speed also had some very small ripples as is evident from the above figure, which had the same frequency as the input voltage vector $\vec{V_{ref}}$. The reason for this is that the generated $\vec{V_{ref}}$ is not exactly same as the input $\vec{V_{ref}}$ which is sinusoidal. And this error is not similar over all the sectors hence the motor speed is not uniform but a bit oscillating.

Overall it can be noted that the model is really performing well during deployment as well. The errors, though not negligible, are very less. Hence we can proceed to the control part of induction machines where this SVM block can be used.

Chapter 4

Indirect Vector Control of Induction Machines

4.1 Introduction

Now that the NN based SVM inverter was seen to be performing well enough, the indirect vector control scheme as discussed in Chapter 1 using 4 PI controllers was implemented and various waveforms and results were analysed.

A Simulink model was made to implement the control technique. The details of the same are given in Appendix section. Various waveforms were observed in order to check whether the control scheme works properly or not. Unless stated otherwise, for all the controllers the sampling time is set to $T_s = 10^{-4}$.

4.2 Resultant Waveforms

D Axis Stator Current and Reference

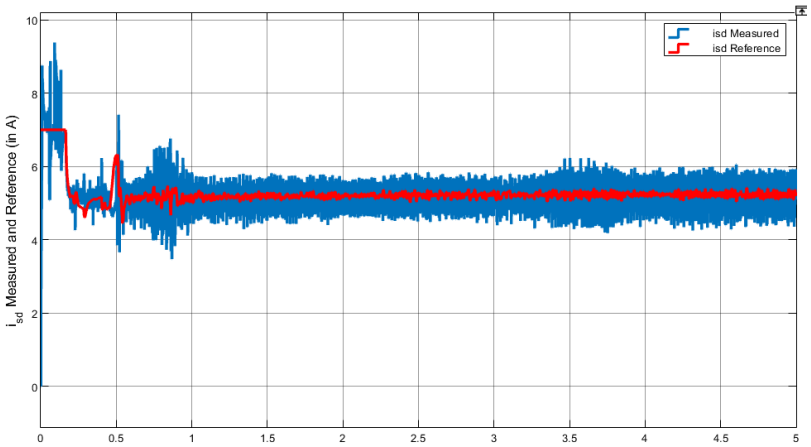


Figure 4.1: D Axis Stator Current and its Reference

The controller performance for d axis stator current was observed and its waveform observed is shown in Fig 4.1. It can be observed that the controller is able to give desired i_{sd} value as set by the outer flux controller. This implies that this controller is working properly. Though there is some ripple in the current, but still the controller is working quite properly to keep average i_{sd} to the desired reference value.

Q Axis Stator Current and Reference

The controller performance for q axis stator current was observed and its waveform observed is shown in Fig 4.2. The controller is able to track the desired reference value with very less ripple.

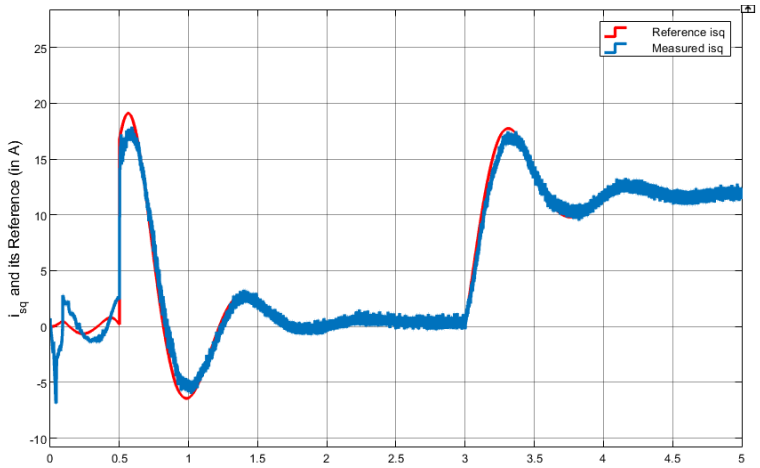


Figure 4.2: Q Axis Stator Current and its Reference

Flux and its Reference

The controller performance for stator flux was observed by measuring the magnetising current i_{mr} and its waveform observed is shown in Fig 4.3.

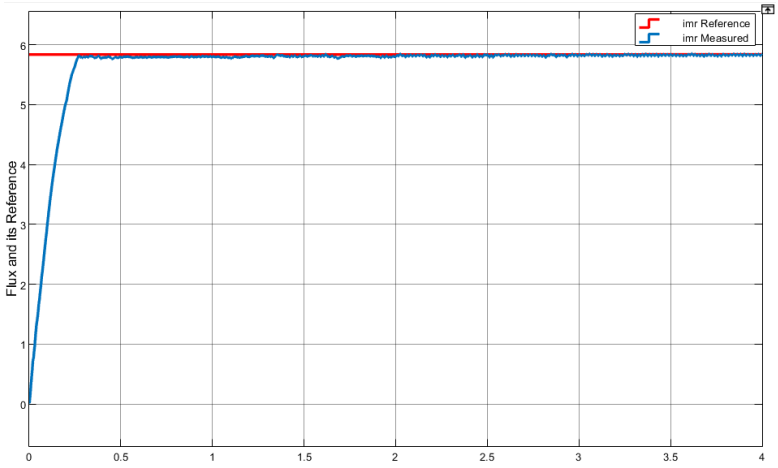


Figure 4.3: Flux and its Reference

Rotor Speed and its Reference

The controller performance for rotor speed was observed and its waveform observed is shown in Fig 4.4.

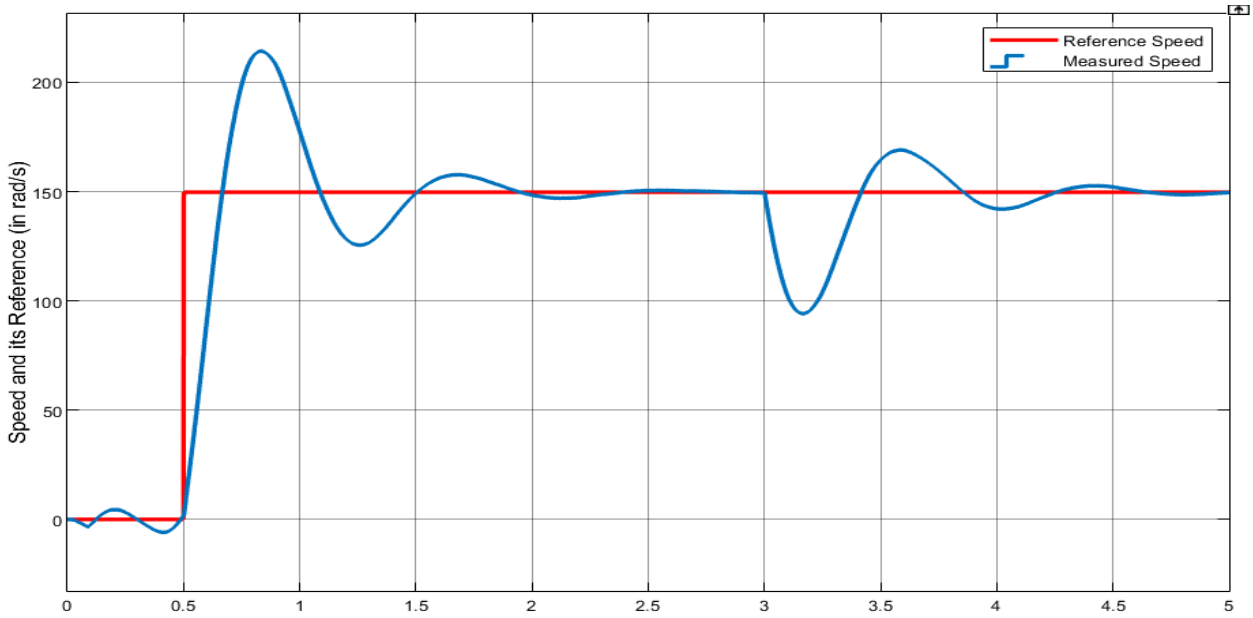


Figure 4.4: Speed and its Reference

Load Torque and Machine Electromagnetic Torque

The machine load and electromagnetic waveform is as shown in Fig 4.5.

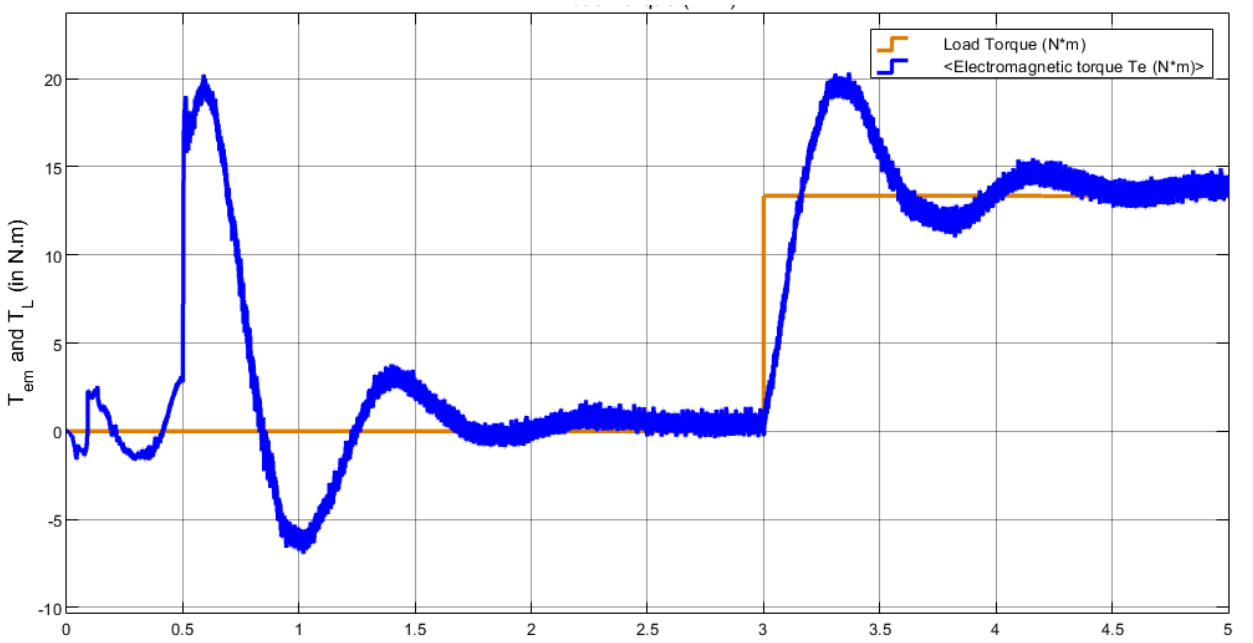


Figure 4.5: Load and Machine Electromagnetic Torque

Machine Input Currents Along with Sector and Switching Sequence

The machine input currents which are also the stator currents were observed to check if the NN based SVM inverter block is performing good enough with external controllers. The observed waveform is shown in Fig 4.6. The zoomed version of same under steady state is shown in Fig 4.7. It was observed that the NN based SVM inverter performs well with the external PI controls.

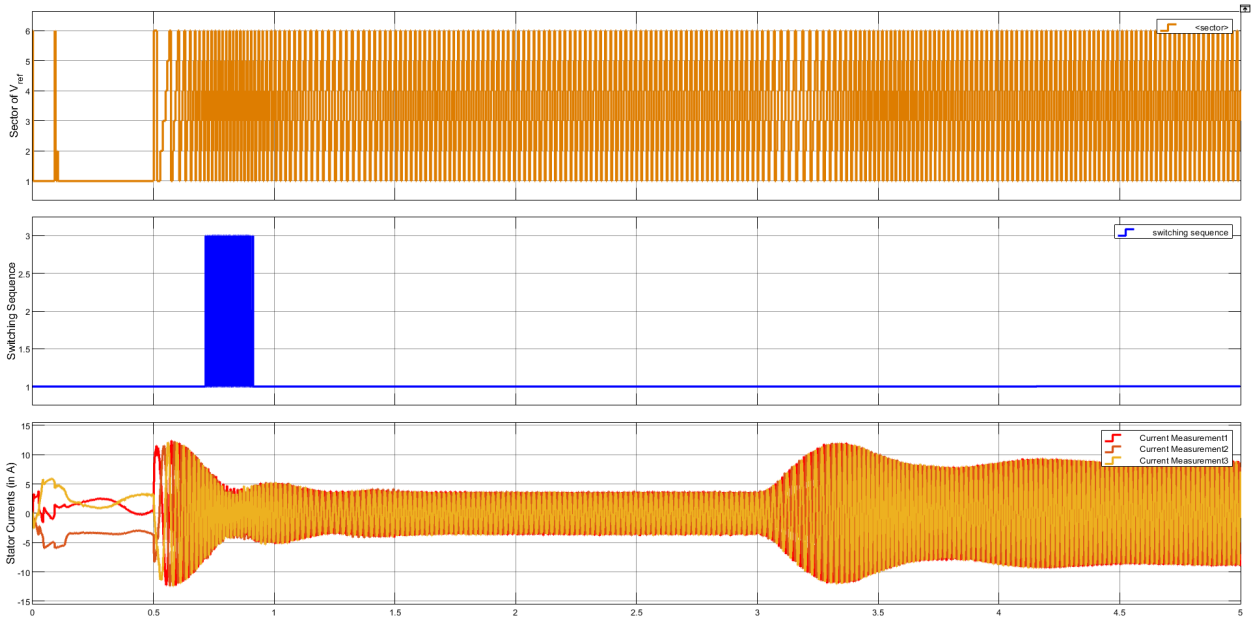


Figure 4.6: Sector, Switching Sequence and Three Phase Machine Input Currents

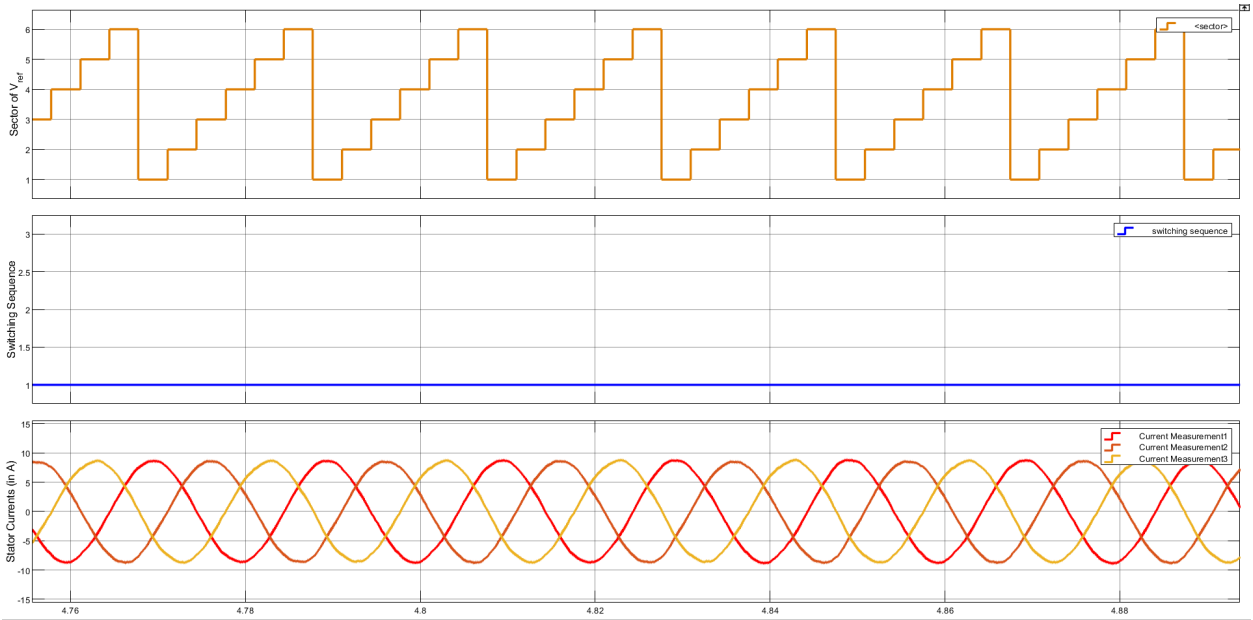


Figure 4.7: Sector, Switching Sequence and Three Phase Machine Input Currents in Steady State

Synchronous Speed Determined in Flux Estimation

The flux estimation block inherently also calculates the synchronous speed ω_s which is also the speed at which the dq frame rotates. Its waveform as observed is shown in Fig 4.8.

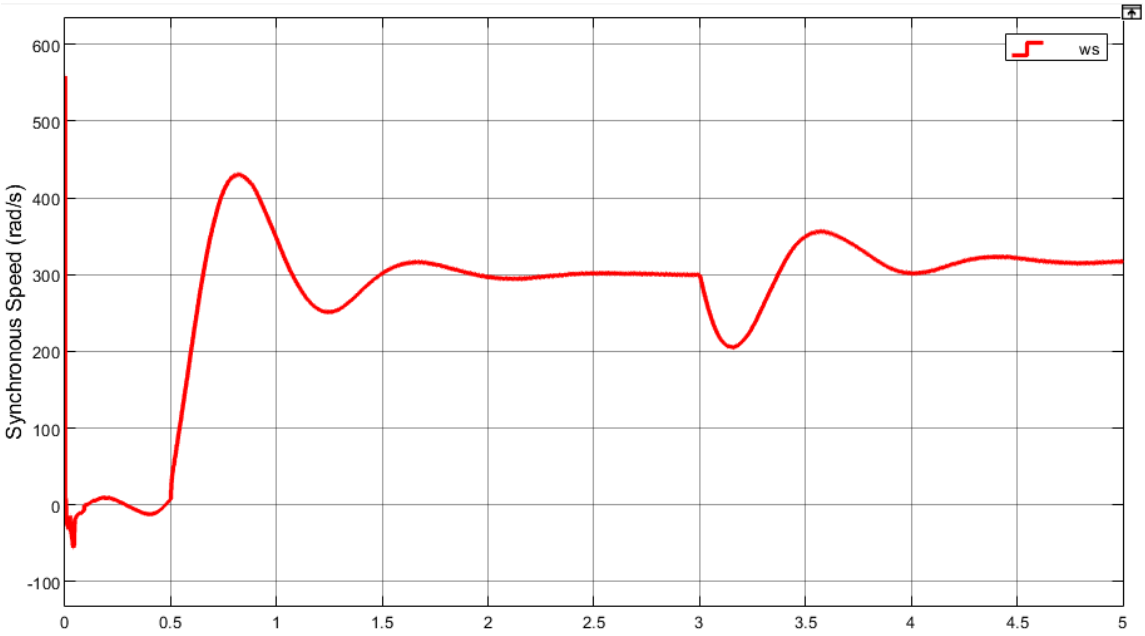


Figure 4.8: Synchronous Speed

From above waveforms, it can be seen that both the current controllers are working quite good and are able to track the currents to their reference values in very less time. Their transient characteristics is also very good which is also evident from the stator currents waveforms as the currents smoothly vary during transient state as well. The machine is also able to generate the desired torqueas per load requirement as is visible from torque waveform.The speed of the motor is also remaining the same during steady states of no load and full load which is one of the main purpose of using control techniques. Overall, it can be concluded that the NN based SVM inverter is working well enough with the indirect vector control scheme and is providing good enough results.

4.3 Comparison of Performance of Control Scheme with Different Sampling Frequencies

The performance of PI controllers was also seen with higher sampling frequency of $T_s = 5 \times 10^{-4}$. The different controller waveforms with both the sampling frequencies were observed and their performance were compared for both the sampling frequencies. The reference values were also plotted to observe their errors from their desired references.

D Axis Stator Current and Reference

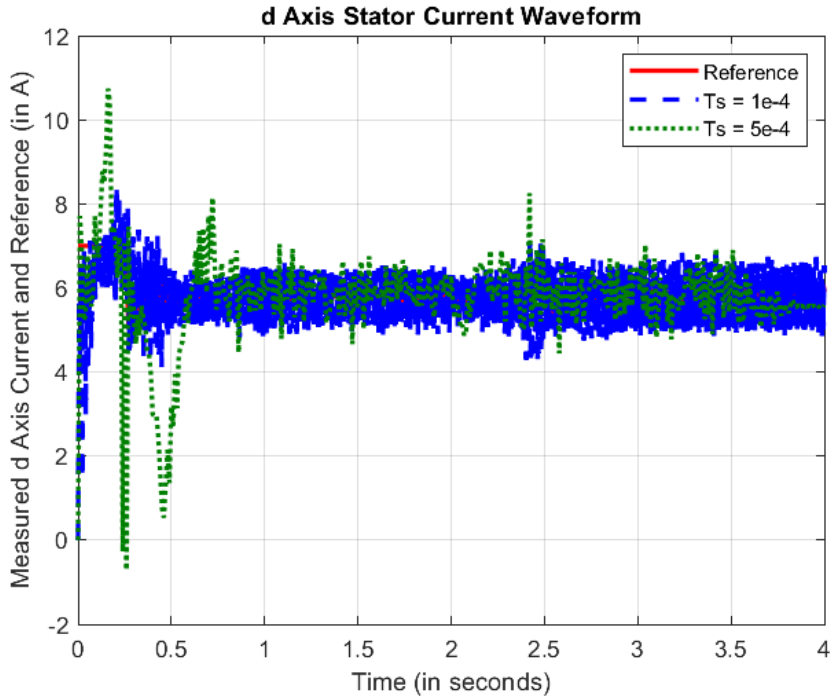


Figure 4.9: D Axis Stator Current and its Reference with Different T_s

Q Axis Stator Current and Reference

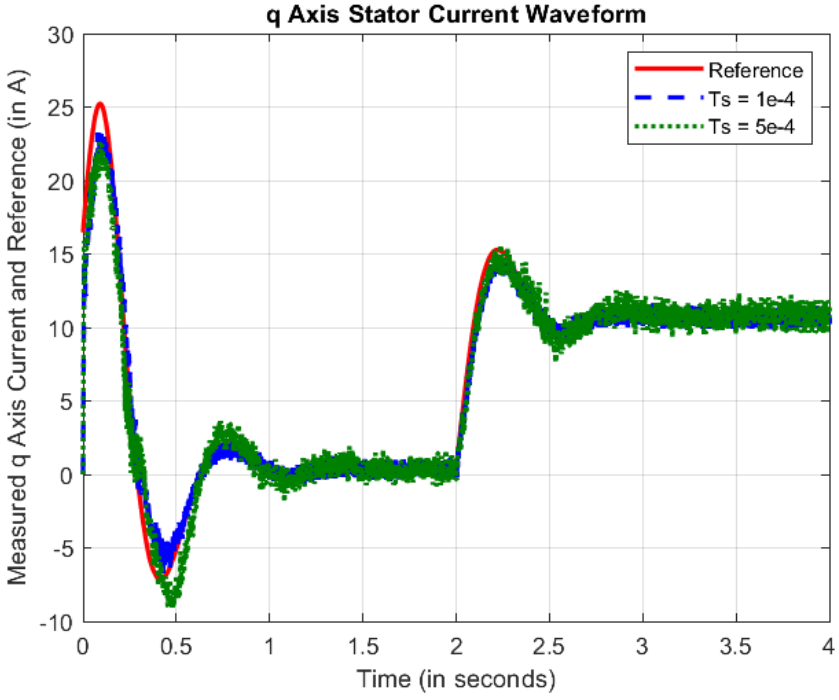


Figure 4.10: Q Axis Stator Current and its Reference with Different T_s

Flux and its Reference

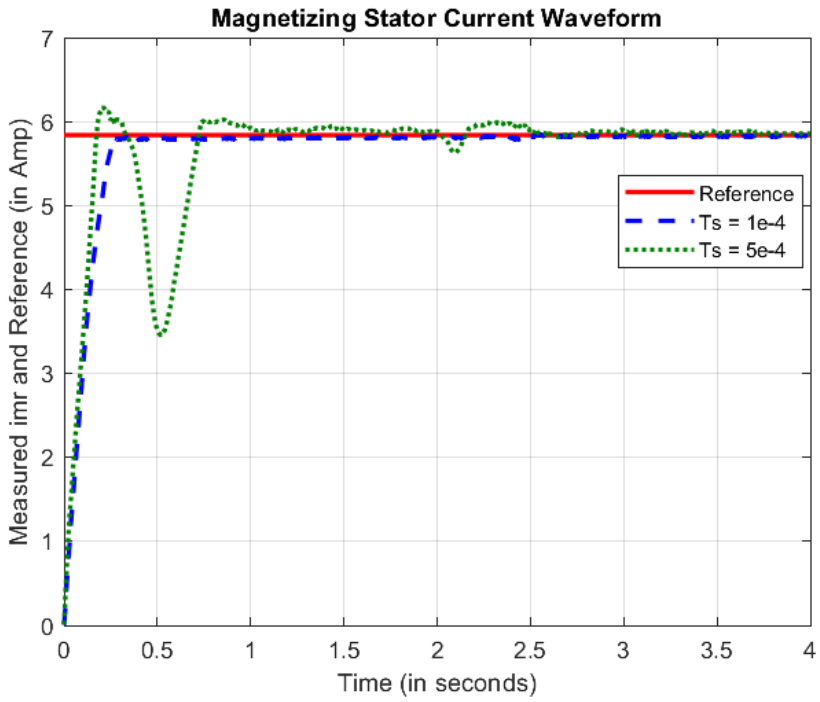


Figure 4.11: Flux and its Reference with Different T_s

Rotor Speed and its Reference

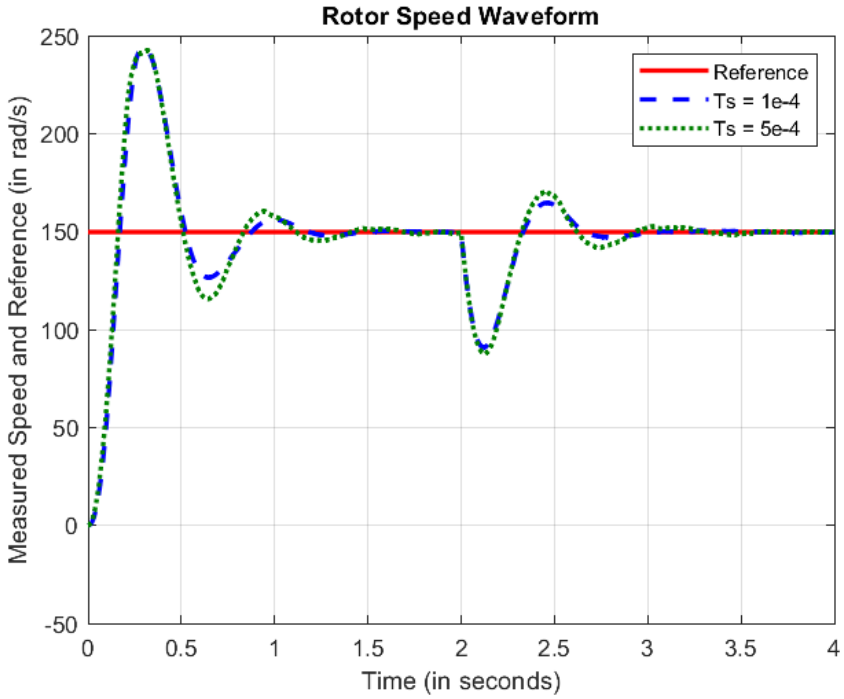


Figure 4.12: Speed and its Reference with Different T_s

From above figures, it can clearly be seen that with higher sampling frequency ($T_s = 10^{-4}$), the control scheme works fairly good but with lower sampling frequency ($T_s = 5 \times 10^{-4}$), the control action is not as good.

In order to perform well enough for other sampling frequencies, we can incorporate sampling delay into consideration in equivalent transfer function model but then for different sampling times, the PI controllers parameters need to be changed accordingly. Moreover, even if PI controller parameters are varied accordingly to maintain system performance, it has been shown in past works that the current loops become slower as the sampling frequency is reduced. However, a neural network has a scope for better performance with different sampling frequencies since it can use the T_s information and give output accordingly with other factors, and maintain the speed of control action as well. In the next chapter, a neural network is trained and tested for replacing the inner two loops in control scheme, and its performance with different sampling frequencies is seen.

Chapter 5

NN Based Control of Induction Machines

As seen in previous chapter, the PI controller based indirect vector control technique faces issue of its performance dependency on sampling time T_s. In this chapter, the inner two current loops in vector control scheme along with decoupling block is to be replaced with a neural network and its performance is to be analyzed.

5.1 Training Dataset Creation

The training dataset was created by assuming the overall closed current loops to be of first order nature with time constant τ. The overall transfer function of plant for current loop is

$$G_p(s) = \frac{1}{s\sigma L_s + R_s} \qquad \ldots \quad (5.1)$$

This transfer function function along with NN under closed loop operation should give first order response whose transfer function given by:

$$G_{loop}(s) = \frac{1}{1 + s\tau} \qquad \ldots \quad (5.2)$$

The output of the NN is supplied as input to the plant (decoupling terms cancel each other). Let this output of NN be x. Then,

$$\frac{i_{sd}(s)}{x(s)} = G_p(s) = \frac{1}{s\sigma L_s + R_s} \qquad \ldots \quad (5.3)$$

The above equation is in Laplace domain. Converting it into time domain using inverse Laplace transform we get,

$$i_{sd}(n)\left[1 + \frac{T_s}{\sigma L_s}\right] = i_{sd}(n-1) + \frac{T_s}{\sigma L_s}x(n) \qquad \ldots \quad (5.4)$$

Now, the first order response of closed loop system in time domain is given as,

$$i_{sd}(n) = i^*_{sd}\left(1 - e^{-nT_s/\tau}\right) \qquad \ldots \quad (5.5)$$

The error $e(n)$ is the difference between reference and actual i_{sd} value. From above equation,

$$e(n) = i^*_{sd}\, e^{-nT_s/\tau} \qquad \ldots \quad (5.6)$$

Rewriting desired response equation using above equation,

$$i_{sd}(n) = i^*_{sd} - e(n)e^{-nT_s/\tau} \qquad \ldots \quad (5.7)$$

Replacing $i_{sd}(n)$ in original equation of $x(n)$ with above equation we obtain,

$$\frac{T_s}{\sigma L_s}x(n) = \left(i^*_{sd} - e(n)e^{-nT_s/\tau}\right)\left[1 + \frac{T_s}{\sigma L_s}\right] - i_{sd}(n-1) \qquad \ldots \quad (5.8)$$

The above equation relates the input to NN controller namely reference and measured i_{sd} values, to the output $x(n)$ desired from NN in order to achieve first order time response of current control loop. The equation for i_{sq} is also exactly the same with i_{sd} values replaced by corresponding i_{sq} values.

The above derived equation was used to create training set. The training set was created for various reference values by recursively using above equations for determining new i_{sd} and i_{sq} values. About a total of 100,000 values were taken with 159 values for each reference pair. A sample of training set is shown in Figure 5.1. The additional values of flux and speed were also stored which are used during decoupling. For debugging purposes, the voltage values before and after decoupling are both stored.

isd_ref	isq_ref	isd	isq	w	phi	d_phi/dt	Vsd'	Vsq'	Vsd	Vsq
5.267905	24.5814	0	0	99.54363	1.275022	20.45619	165.2652	771.1698	185.0505	893.9276
5.267905	24.5814	1.421391	6.632576	89.55214	0.121449	98.47163	122.6702	572.4109	205.4761	594.2509
5.267905	24.5814	2.459261	11.47554	175.9972	1.323267	67.73746	91.56829	427.2812	132.343	672.8124
5.267905	24.5814	3.217092	15.01178	2.629181	1.403172	16.58611	68.85831	321.3106	81.95805	345.0997
5.267905	24.5814	3.770444	17.59386	91.00139	0.642526	42.46347	52.27595	243.9331	67.48971	328.9874
5.267905	24.5814	4.17449	19.47925	158.1098	0.376028	87.66862	40.16786	187.4336	73.94566	281.2908
5.267905	24.5814	4.469517	20.85592	121.5029	0.121478	60.4861	31.32679	146.1789	5.22858	205.802
5.267905	24.5814	4.684939	21.86113	15.33581	1.408263	58.90841	24.87122	116.0555	72.73725	167.4252
5.267905	24.5814	4.842235	22.59512	184.3724	0.320254	35.15218	20.1575	94.06009	-18.4017	196.2057
5.267905	24.5814	4.95709	23.13106	93.22666	0.952925	20.86514	16.71564	77.99948	3.42476	201.2831
5.267905	24.5814	5.040954	23.52239	55.5798	0.646828	40.14149	14.20247	66.27236	24.75308	137.7999
5.267905	24.5814	5.102191	23.80814	189.2105	0.109338	45.58415	12.3674	57.70947	-75.6327	137.0957
5.267905	24.5814	5.146904	24.01678	139.3014	0.166325	17.56022	11.02747	51.45702	-64.164	124.962
5.267905	24.5814	5.179553	24.16913	99.00543	0.851559	31.6948	10.04908	46.89161	2.58733	168.1449
5.267905	24.5814	5.203392	24.28037	143.1858	1.258608	49.41744	9.334684	43.55804	9.938024	259.6621
5.267905	24.5814	5.220799	24.36159	161.7685	0.523366	83.4099	8.813046	41.12394	26.64518	168.2706
5.267905	24.5814	5.233509	24.4209	11.60339	0.949254	81.10229	8.432156	39.34661	73.88282	84.6534
5.267905	24.5814	5.24279	24.46421	117.2212	0.064196	0.91302	8.154039	38.04885	-168.393	115.2769

Figure 5.1: Training Set for NN Based Control of IM

5.2 Neural Network Model for Control Scheme

Once training set was created, the neural network architecture was decided and trained. Here directly a neural network was taken because simple architecture of NN provided good results. Some important concepts like feature expand were also implemented according to the requirement.

5.2.1 Neural Network Architecture

The neural network with 1 hidden layers provided good enough results in this case and so without much design considerations, its structure was finalized. The input layer (without feature expand) had 7 independent inputs (2 pairs of measured and reference current values, as well as speed and flux values) and the output was the dq voltage command to the NN based SVM inverter block. The hidden layer had 20 nodes and the overall architecture is shown in Figure 5.2.

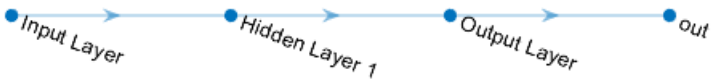


Figure 5.2: NN Architecture for Current Control

5.2.2 Feature Expansion

As mentioned in Chapter 3, feature expansion is required to properly include multiplication of some independent input terms which would otherwise be impossible for NN to realise. In this case, mainly the feature expansion is required to realise the terms of decoupling since they include terms like multiplication of flux and currents, etc. These were included accordingly before processing the input further.

5.2.3 Neural Network Training

Once the designed was finalised, the neural network was trained again and again and fine-tuning of hyperparameters were done to get better results. The final set hyperparameters values are shown in Table 5.1.

80% of the training set was used for training and 20% was reserved for testing. The model was trained on a CPU and the final loss values were found to be 0.0004131 and 0.0004453 on training and test sets respectively. The training plot is is given in Fig 5.3.

The training was done in the same manner as it is done for a feedforward neural network, although when it will actually be used then it would actually become a recurrent

Table 5.1: Final Hyperparameters Values for Training NN for Control

Optimization Used	ADAM
Number of Epochs	20000
Initial Learning Rate	0.02
Learning Rate Decay	0.8 in every 500 iterations
Regularization Factor	0.0001

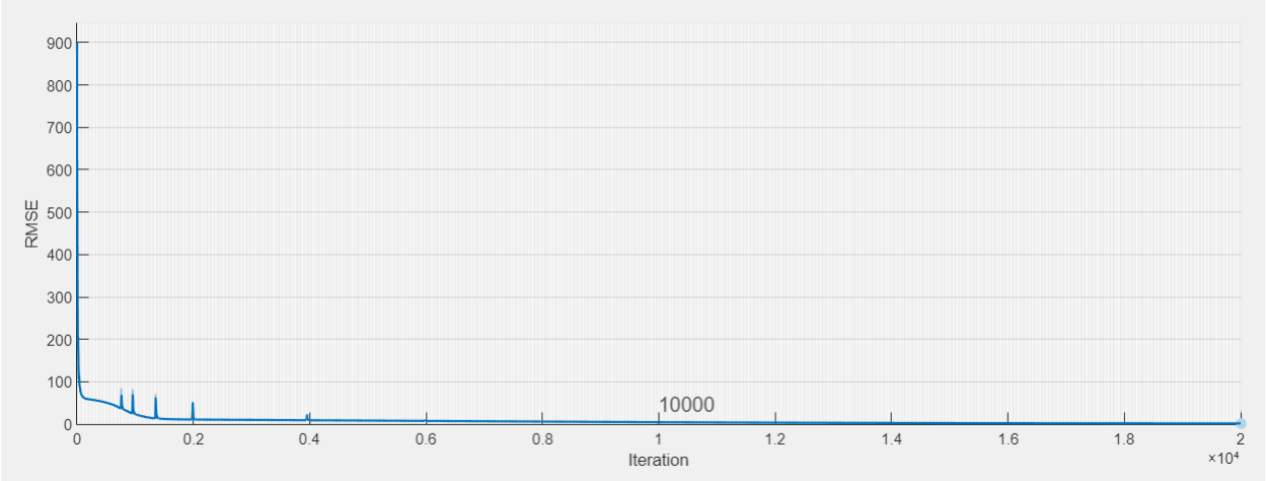


Figure 5.3: Training Progress Plot for NN Controller

neural network (RNN) since feedbacks would be provided in terms of measured currents. But for training, we can assume feedback also to be a variable and train our model without knowing that feedback is a result of past inputs.

5.3 Resultant Waveforms

After training the NN, it was then deployed to the original Simulink model. The details of the same are given in Appendix section. The following waveforms were observed. For following waveforms, initially motor is provided with just flux reference at start and speed and load commands are given at time $t = 0.5$ and $t = 3$ seconds respectively.

D Axis Stator Current and Reference

The controller performance for d axis stator current was observed and its waveform observed is shown in Fig 5.4.

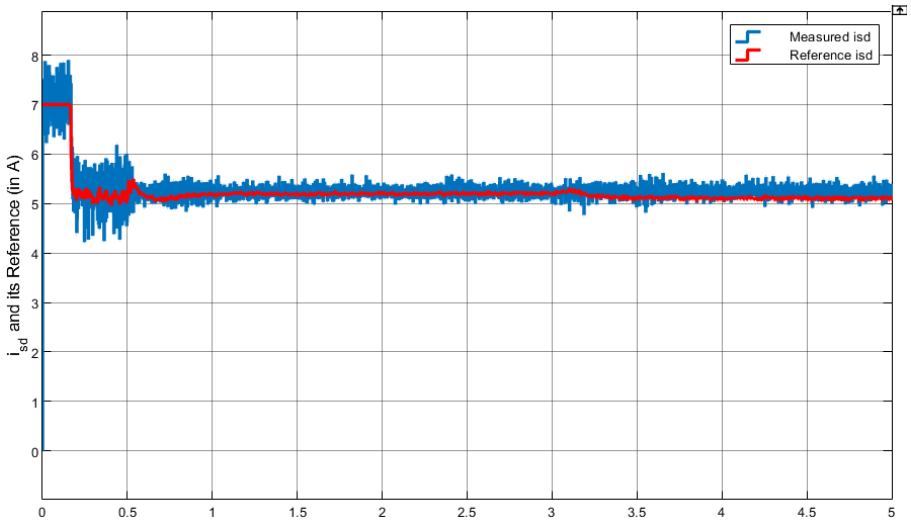


Figure 5.4: D Axis Stator Current and its Reference with NN Control

It can be observed that the controller is able to give desired i_{sd} value as set by the outer flux controller. This implies that this controller is working properly. The ripple is less compared to the PI controller case as observed in Chapter 4 which implies NN is working properly for d axis current control.

Q Axis Stator Current and Reference

The controller performance for q axis stator current was observed and its waveform observed is shown in Fig 5.5. The NN controller is able to properly control the q axis current.

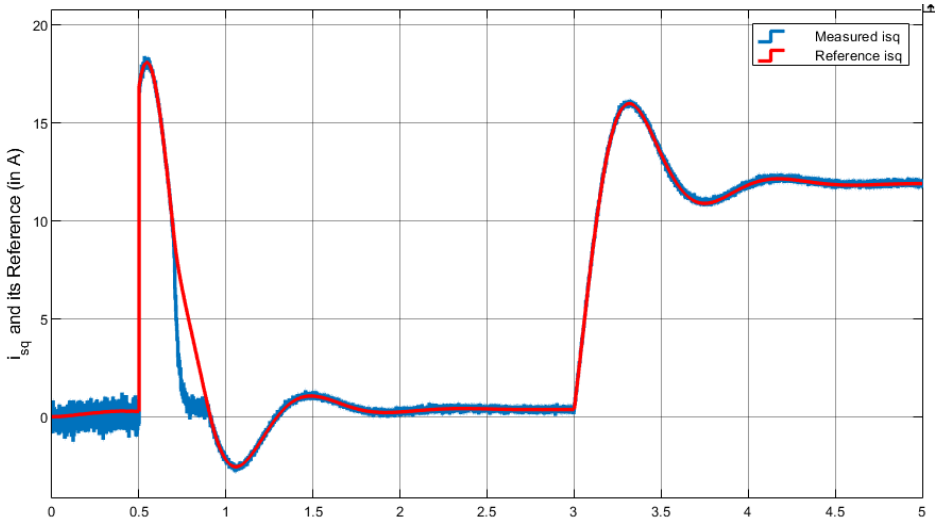


Figure 5.5: Q Axis Stator Current and its Reference with NN Control

Flux and its Reference

The controller performance for stator flux was observed by measuring the magnetising current i_{mr} and its waveform observed is shown in Fig 5.6.

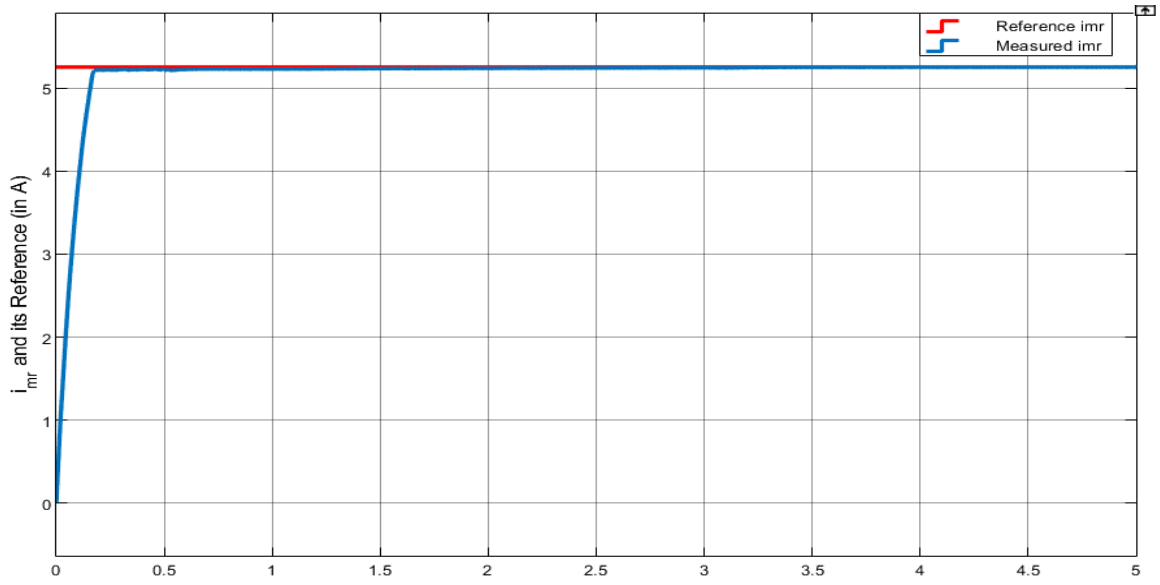


Figure 5.6: Flux and its Reference with NN Control

Rotor Speed and its Reference

The controller performance for rotor speed was observed and its waveform observed is shown in Fig 5.7.

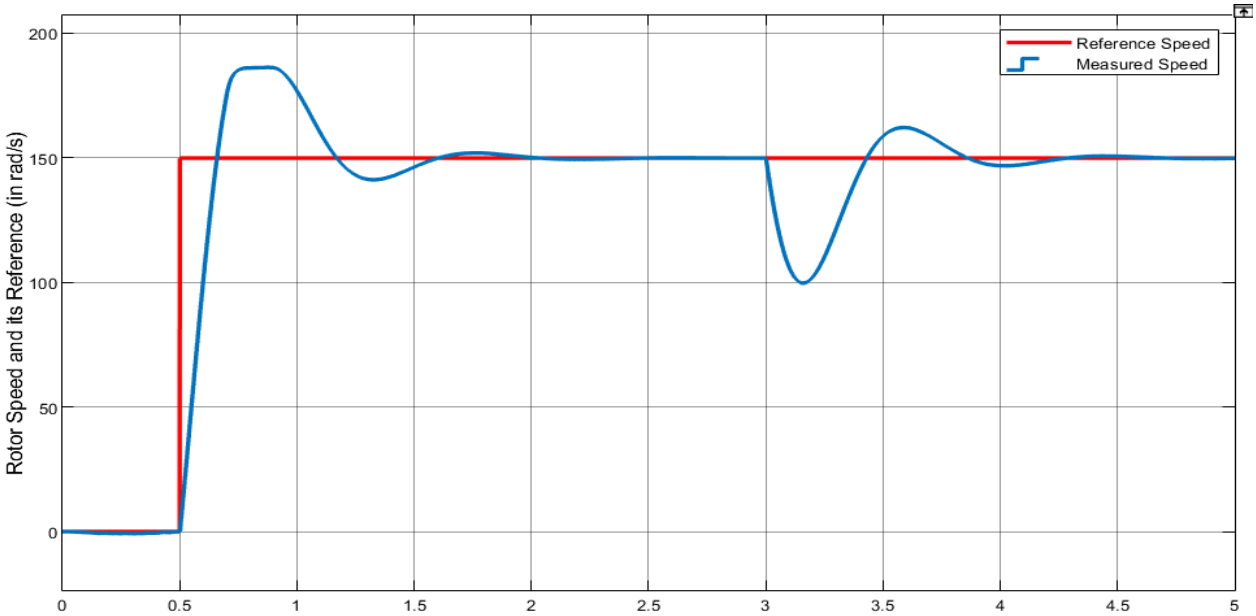


Figure 5.7: Speed and its Reference with NN Control

Load Torque and Machine Electromagnetic Torque

The machine load and electromagnetic torque waveform is as shown in Fig 5.8.

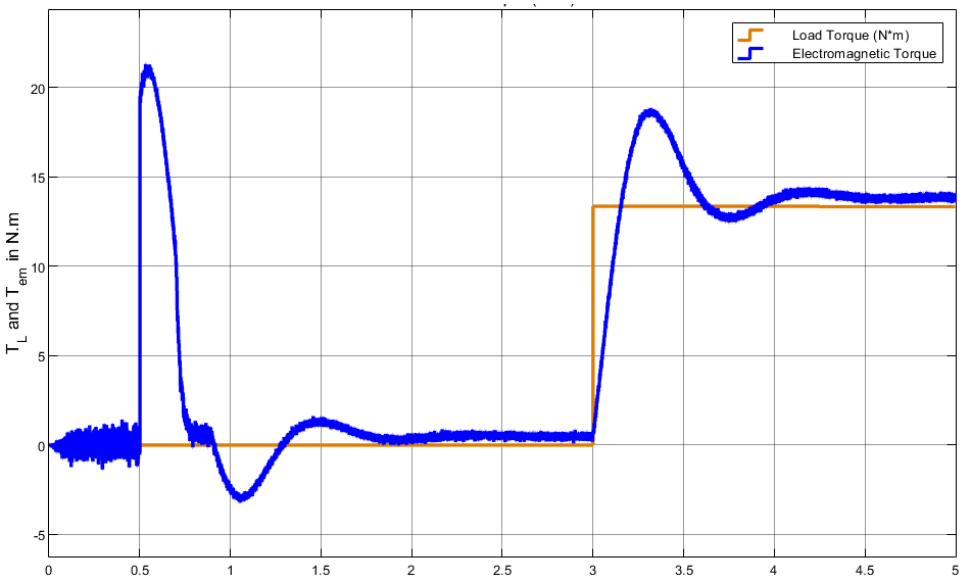


Figure 5.8: Load and Machine Electromagnetic Torque for NN Control

Synchronous Speed Determined in Flux Estimation

The flux estimation block inherently also calculates the synchronous speed ω_s which is also the speed at which the dq frame rotates. Its waveform as observed is shown in Fig 5.9.

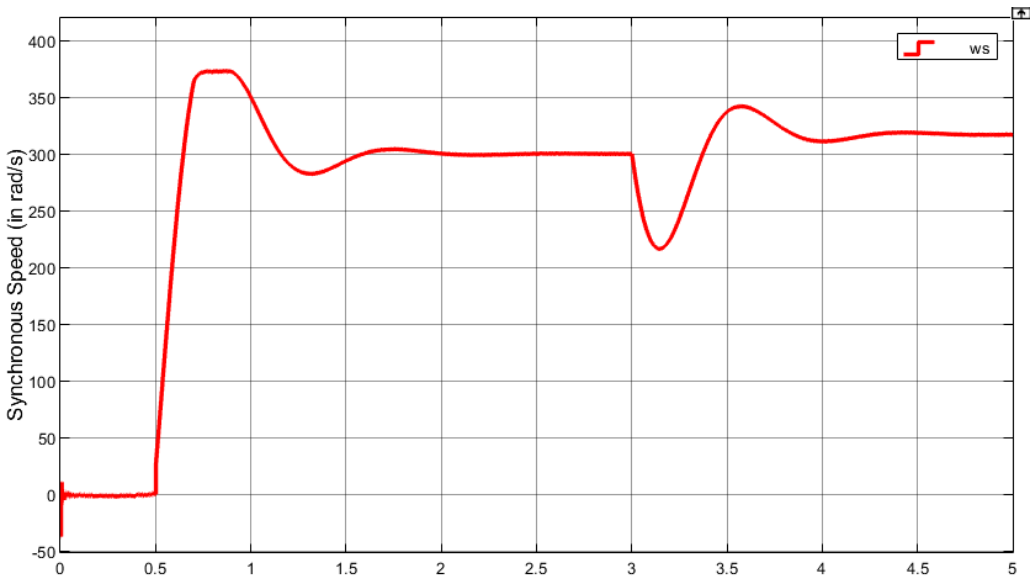


Figure 5.9: Synchronous Speed with NN Control

Machine Input Currents Along with Sector and Switching Sequence

The machine input currents which are also the stator currents were observed to check if the NN based SVM inverter block is performing good enough with external NN current controller. The observed waveform is shown in Fig 5.10. The zoomed version of same under steady state is shown in Fig 5.11. It was observed that the NN based SVM inverter performs well with the external NN control as well.

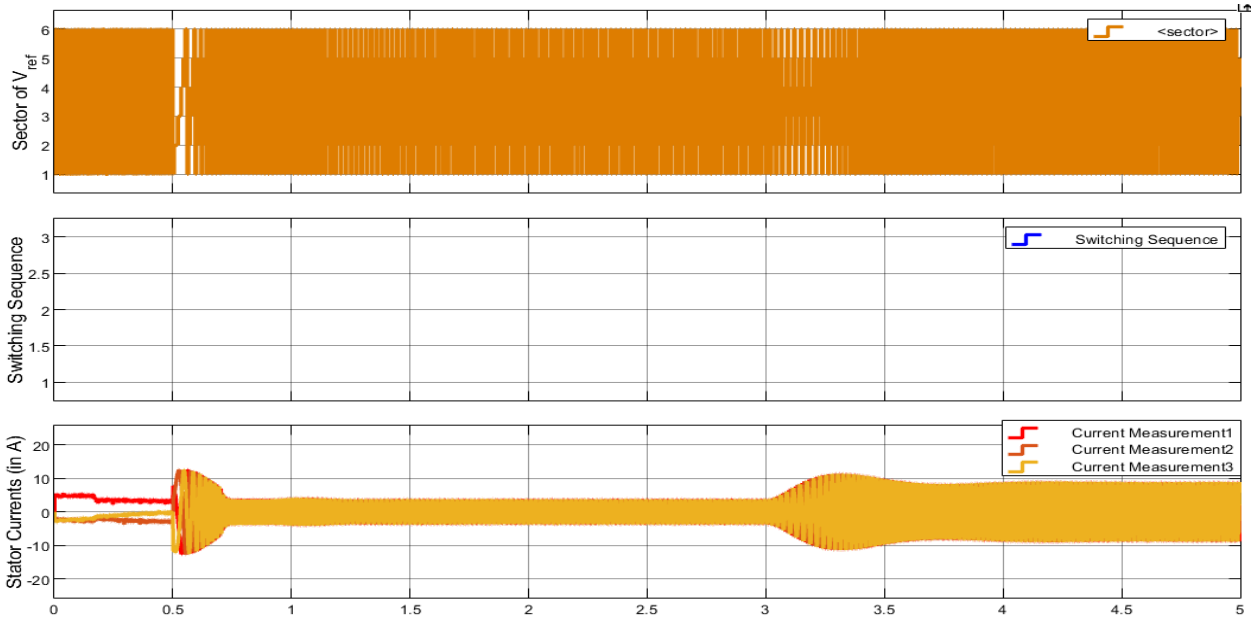


Figure 5.10: Sector, Switching Sequence and Three Phase Machine Input Currents with NN Control

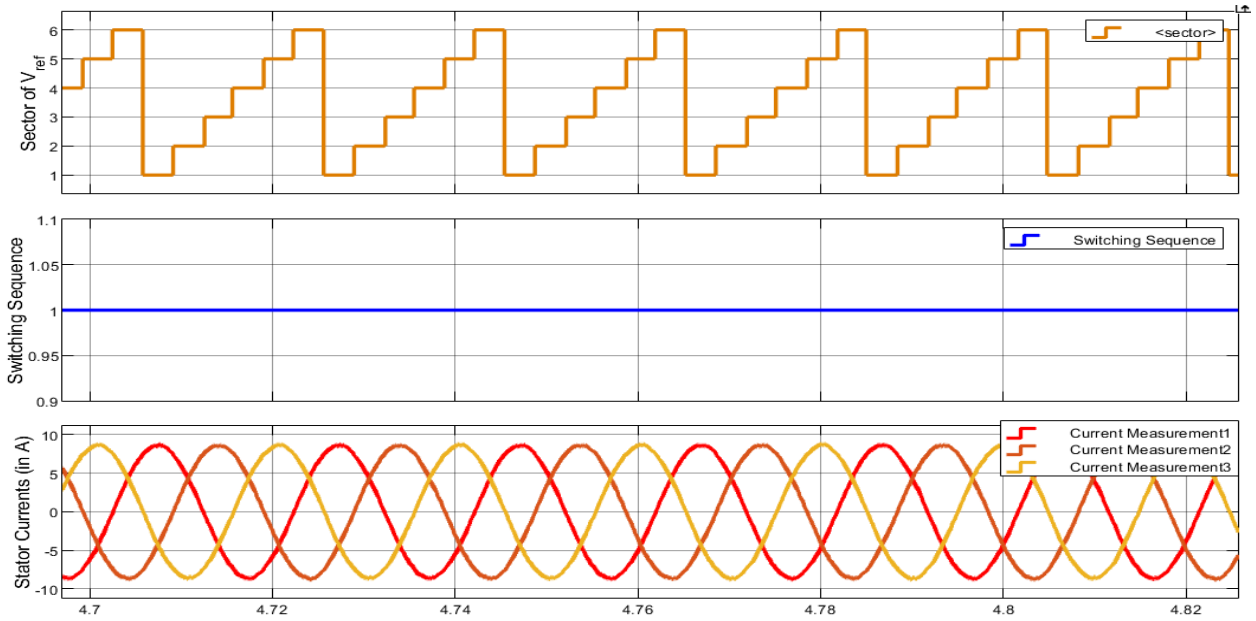


Figure 5.11: Sector, Switching Sequence and Three Phase Machine Input Currents in Steady State with NN Control

From the waveforms above, it can be seen that the NN current controller is able to track the desired reference values as set by the outer PI controllers and both the NNs are able to work well enough with each other and provide good results. The machine input currents are also smoothly varying implying that the NN is able to control the system well. Also, from the torque waveform it can be seen that the machine is able to generate the required torque with very less ripple.

5.4 Comparison of Performance of NN Based Control Scheme with Different Sampling Frequencies

Just like in the case of PI controllers, the performance of NN based controllers was also seen with higher sampling frequency of $T_s = 5 \times 10^{-4}$. The controller waveforms with both the sampling frequencies were observed. A comparison of the below waveforms with that in Chapter 4 is presented towards the end of this Chapter to compare the performance of NN control with PI control.

D Axis Stator Current and Reference

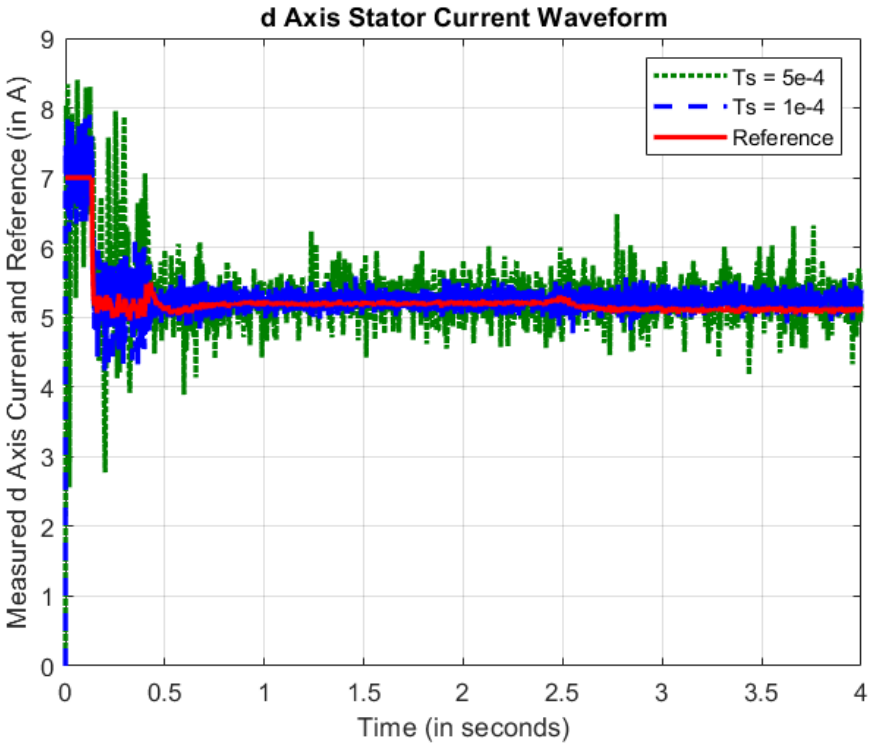


Figure 5.12: D Axis Stator Current and its Reference with Different T_s with NN Based Control

Q Axis Stator Current and Reference

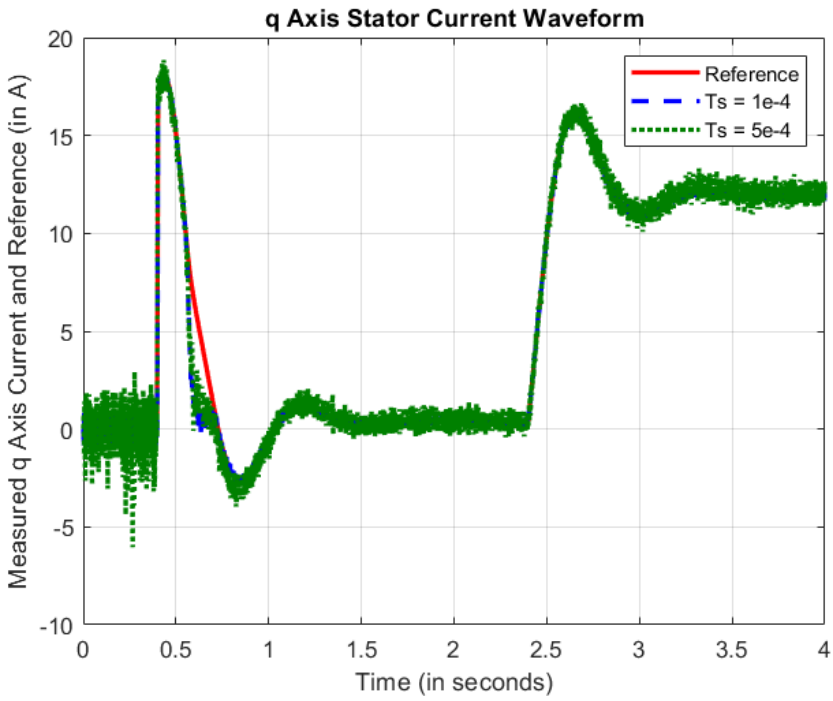


Figure 5.13: Q Axis Stator Current, Reference with Different T_s with NN Based Control

Flux and its Reference

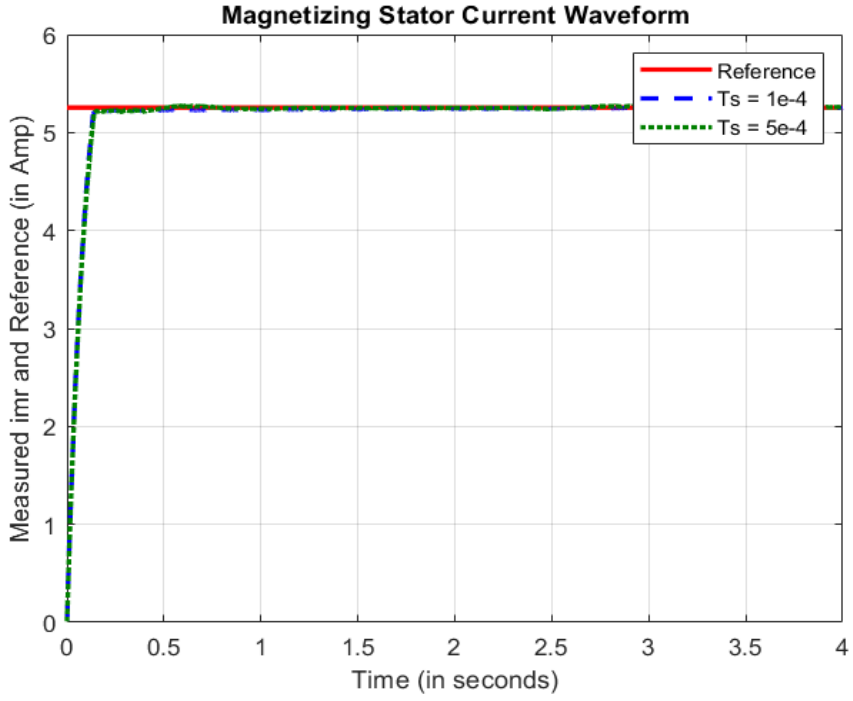


Figure 5.14: Flux and its Reference with Different T_s with NN Based Control

Rotor Speed and its Reference

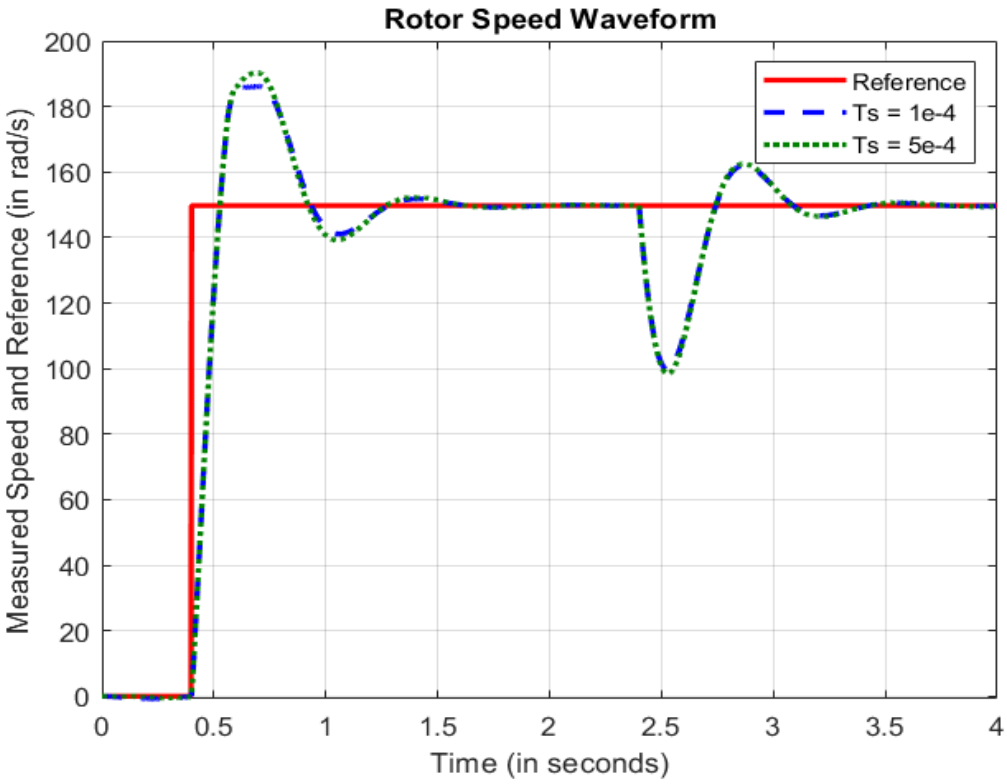


Figure 5.15: Speed and its Reference with Different T_s with NN Based Control

From the above figures, it can be seen that this control scheme works well with both higher and lower sampling frequencies, and there is hardly any difference between the controller waveforms under both sampling frequencies, which was not the case with only PI controllers based control scheme.

Hence we can say that the NN based control scheme outperforms the usual PI controller based indirect vector control scheme in case of lower sampling frequencies. The performance of NN controller degrades very less with decreasing sampling frequency as compared to the only PI controller based control scheme. This is an important result because with independency from sampling time, the technique now becomes more universal and applicable to a large amount of applications.

Chapter 6

Conclusions and Future Works

Conclusions

SVM is one of the most effective method for performing PWM. Although very effective, but it is still not able to generate perfect sinusoidal outputs. Hence calculation of errors is necessary to effectively operate with the SVM technique and distinguish between two SVM alternatives. Stator flux error was used in this project to compare between various switching sequence alternatives.

The three zone hybrid technique was discussed implemented using Simulink. The Simulink block for SVM logic was itself used to generate training dataset for NN. Different neural networks architectures were tried and the best performing one was chosen to perform SVM logic generation. The trained model was then deployed to original Simulink model and its performance was found to be very good.

Once the SVM logic was replaced by NN, the indirect vector control scheme was implemented and its performance was measured. It was found that the indirect vector control performs well with higher sampling frequencies but its performance degrades considerably with lower sampling frequencies.

A RNN, which was trained like a ANN, was trained to replace the inner two loops along with decoupling block of the control scheme. The performance of the RNN was seen against different sampling frequencies and it was found to perform well even with lower sampling frequencies.

Overall, the two NN were found to work well with each other and provide good results. It was an important thing because now the two NN can be combined into one single NN and it will itself generate gate timings for switches depending on the reference and measured current values.

Future Works

Future work include the following:

1. Extend the NN model to Five Zone Hybrid SVM technique and finally to Seven Zone Hybrid SVM technique. The same architecture with very minimum difference should perform well.

2. Taking into account machine parameters also for NN based control to make the NN performance independent of machine parameters.

3. Combining the two NNs into one single NN, which can possibly result in less total number of layers, and boost the computation time further.

4. Hardware implementation of the both the NN based SVM method and NN based control method individually and together to check their performance on hardware setup.

Bibliography

[1] A.Bakhshai, Praveen K. Jain and Hua Jin, Incorporating the Overmodulation Range in Space Vector Pattern Generators Using a Classification Algorithm", IEEE, 2000

[2] G. Narayanan, Di Zhao, Harish K. Krishnamurthy, Rajapandian Ayyanar and V.T. Ranganathan, "Space Vector Based Hybrid PWM Techniques for Reduced Current Ripple", IEEE Transactions on Industrial Electronics, April 2008.

[3] Vishnu V Bhandankar and Anant J Naik, "Artificial Neural Network based implementation of Space Vector Modulation for Three Phase VSI", IEEE, April 2016.

[4] Joao O.P. Pinto, Bimal K. Bose, Luiz Eduardo Borges da Silva and Marian P. Kazmierkowski, "A Neural Network Based Space Vector PWM Controller for Voltage-Fed Inverter Induction Motor Drive", IEEE Transactions on Industrial Electronics, November/December 2000.

[5] Joao O.P. Pinto, Bimal K. Bose and Luiz Eduardo Borges da Silva, "A Stator Flux Oriented Vector Controlled Induction Motor Drive With Space Vector PWM and Flux Vector Synthesis by Neural Networks", IEEE Transactions on Industry Applications, September/October 2001.

[6] Xingang Fu and Shuhui Li, "A Novel Neural Network Vector Control Technique for Induction Motor Drive", IEEE Transactions on Energy Conversion, December 2015.

[7] Yang Xiao-ping, Zhong Yan-ru and Wang Yan, "A Novel Control Method for DSTATCOM Using Artificial Neural Networks", IEEE Transactions on Industry Applications, November/December 2006.

[8] Subrata K. Mondal, Joao O.P. Pinto and Bimal K. Bose, "A Neural Network Based Space Vector PWM Controller for a Three-Level Voltage-Fed Inverter Induction Motor Drive", IEEE Transactions on Industry Applications, May/June 2002.

[9] Mustafa Mohamadian, Ed Nowicki, Farhad Ashrafzadeh, Alfred Chu, Rishi Sachdeva and Ed Evanik, "A Novel Neural Network Controller and Its Efficient DSP Implementation for Vector-Controlled Induction Motor Drives", IEEE Transactions on Industry Applications, November/December 2003.

[10] Russel J. Kerkman, Timothy M. Rowan, David Leggate and Brian J. Siebel, "Control of PWM Voltage Inverters in the Pulse Dropping Region", IEEE Industry Application Magazine, September/October 1996, pp. 25-28.

[11] A.Bakhshai, J.Espinoza, G. Joos and H.Jin, "A Combined Neural Network and DSP Approach to the Implementation of Space Vector Modulation Techniques", IEEE Explore, 1996, pp. 934-940

[12] Shun JIN and Yan-ru ZHONG, "Limit-Trajectory Single- and Two-Mode Overmodulation Technology", IPEMC, 2006

[13] S. Vukosavic and M. Stojic, "Reduction of parasitic spectral components of digital space vector modulation by real-time numerical methods,"IEEE Trans. Power Electron., vol. 10, no. 1, pp. 94–102, 1995

Appendix

Simulink Model for Hybrid SVM Techniques

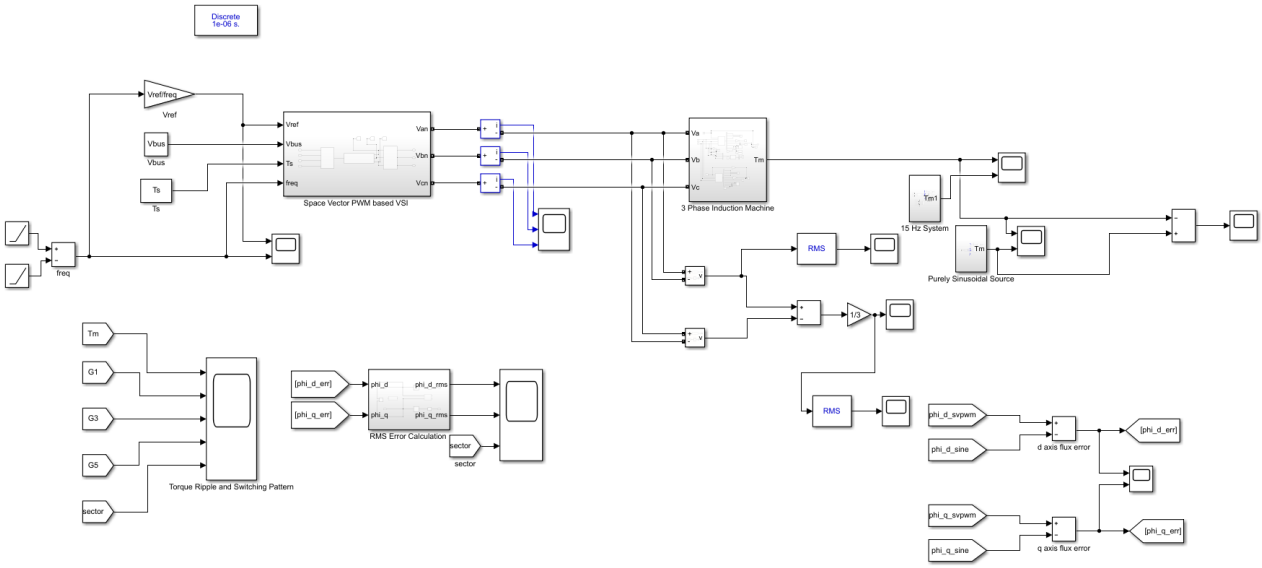


Figure 6.1: Simulink Model for Implementing Hybrid Zone Techniques

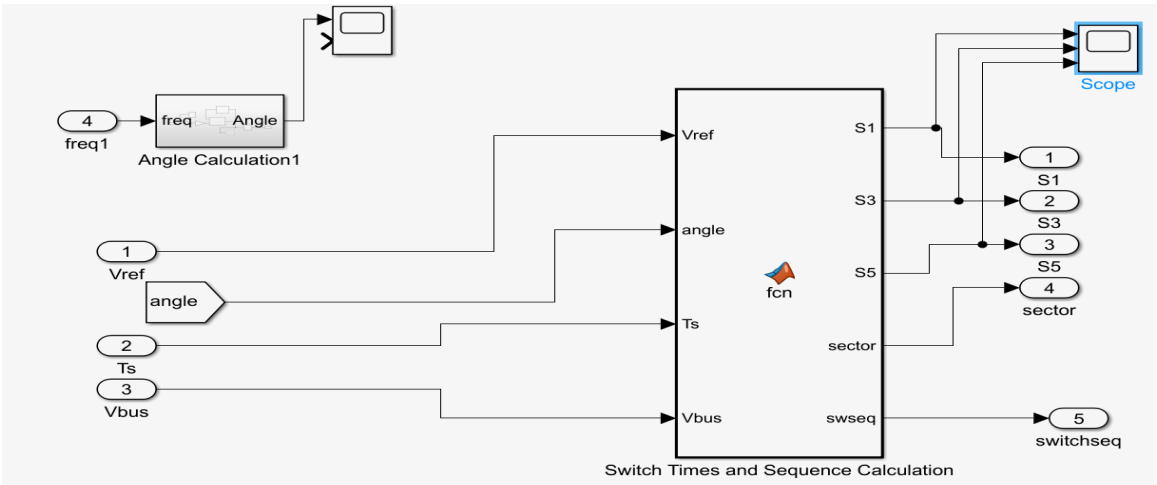


Figure 6.2: "Space Vector PWM Based VSI" Block

A simulink model was built to observe various waveforms and draw important conclusions by implementing the hybrid methods as discussed in Chapter 2. The Simulink

model is as shown in Fig 6.1. The"Space Vector PWM Based VSI" block of the model as shown in Fig 6.2 had the code and logic for implementing different hybrid techniques.

The dc bus voltage V_{bus} was fixed at about 650V (to give $m = 1$ at rated voltage). The sampling frequency ($f_s = 1/T_s$) of the $\vec{V_{ref}}$ vector was set to 10kHz and the system sampling frequency ($f_{sys} = 1/T_{sys}$) was set to 1MHz.

The load used was preset model of a three phase, 4KW, 400V, 50Hz, 1430RPM, 4 pole (2 pole pairs) induction machine having stator and rotor resistances as 1.405Ω and 1.395Ω and inductances as $0.005839H$ each respectively. The mutual inductance value was $0.1722H$ and friction factor was $0.002985Nms$.

Deploying Neural Network for SVM Logic to Simulink Model

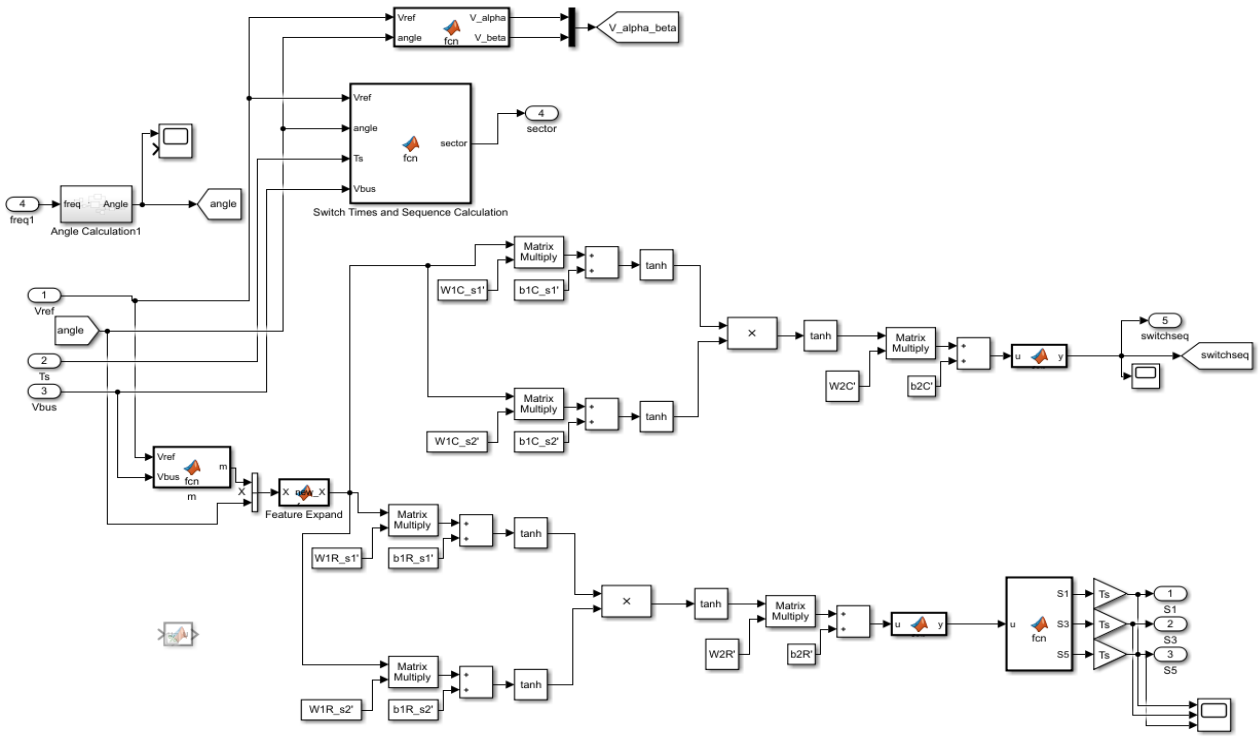


Figure 6.3: "Space Vector PWM Based VSI" block of Simulink Model

After training the neural network for implememting , it's weight matrices were saved locally and loaded into the original MATLAB Simulink model shown in Fig 6.1. After loading, these were then used to predict switching times and sequences for implementing SVM. The "Space Vector PWM Based VSI" block in Fig 6.1 looked as as shown in Fig 6.3 after deploying the NN.

Simulink Model for Indirect Vector Control Technique

Fig 6.4 shows the Simulink model for indirect vector controlled IM drive.

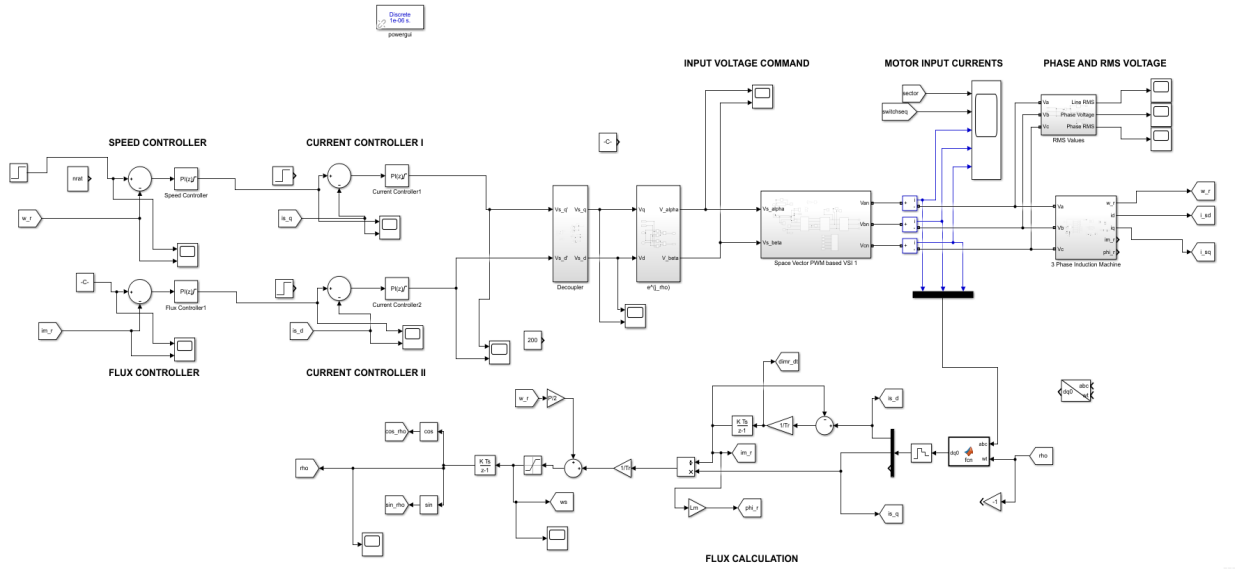


Figure 6.4: Simulink Model for Indirect Vector Controlled IM Drive

As can be seen from Fig 6.4, there were 4 PI controllers in total namely flux, d axis current, speed and q axis current controllers connected in two parallel branches and the current controllers forming the two inner loops. A ZOH was used in measurement of each feedback quantity to incorporate the effect of different sampling times. The input to the SVM block is the voltage command in $\alpha - \beta$ domain, which can be easily obtained by dq to $\alpha - \beta$ conversion, where the dq voltage command is obtained by the two current controller output followed by the decoupling block.

The SVM based VSI block is the same NN based SVM VSI block as described in previous sections. The output of the inverter is supplied to the 3 phase induction machine. The three phase input currents to the induction machine are measured and converted to dq frame in order to perform flux estimation.

Deploying Neural Network to Replace Current Loops

Once the NN for control was trained and the final loss came under acceptable bounds, the neural network was saved in terms of its weights and biases, and then the inner two control loops in control scheme along with the decoupling block were replaced by this NN. The final Simulink model is shown in Fig 6.5.

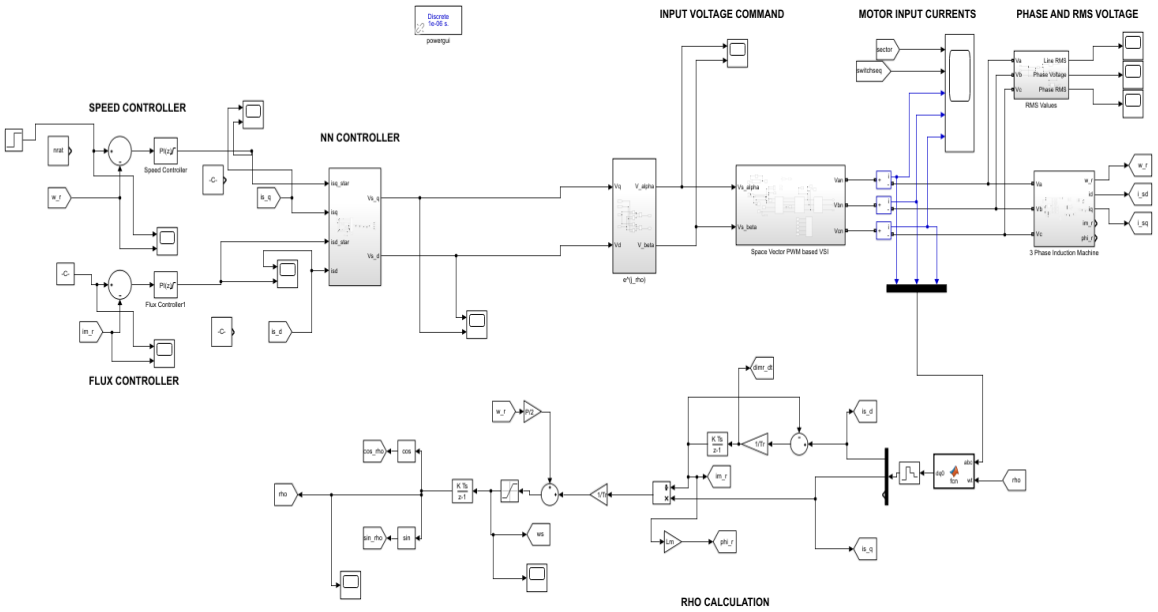


Figure 6.5: Simulink Model for NN Based Controller

MATLAB Codes

To Create Training Data for SVM

```
function train_set =  training_set_creator()
    Vbus = 800;
    train_set_size = 100000;

    %Generate random numbers inside hexagon to get random values of Vref
    %and angle
    Vref_angle = get_random_Vref_angle(train_set_size, Vbus);

    train_set = zeros(train_set_size, 7);
    for i=1:train_set_size
        train_set(i, 1) = (Vref_angle(i, 1)/Vbus) * (2/sqrt(3));
        train_set(i, 2) = Vref_angle(i, 2);
        [train_set(i,3), train_set(i, 4), train_set(i, 5), ...
            train_set(i, 6), train_set(i, 7)] = ...
            create_training_set_row( train_set(i, 1), Vref_angle(i, ...
            2) );
    end
    size(train_set)

    xlswrite("training_set_4.xlsx", train_set);
end
```

Helper Functions for Creating Training Data

```
function [sector, S1, S3, S5, swseq] = create_training_set_row(m, angle)
    %     Method 2
    Ts = 1;
    theta = rem(angle, (pi/3));
    Vbus = 800;
    Vref = m*Vbus*sqrt(3)/2;
    flag = 0;

    numzones = 3;

    %     Method 2

    if numzones == 1
        flag = 1;
    end

    %Switching times calculation of different vectors
    if m ≤ 1                % Linear modulation (Overmodulation mode 0)
        n2 = Vref*cos( theta );
        n1 = Vref*cos( (pi/3) - theta );

        T1 = (2*Ts/(3*Vbus)) * ((-n1) + (2*n2));
        T2 = (2*Ts/(3*Vbus)) * ((2*n1) - n2);
        T0 = Ts-(T1+T2);
        flag = 1;

    elseif m≤1.05       % Overmodulation mode I
        alpha_t = (pi/6) - acos(1/m);
        alpha_t = alpha_t/2;
        if theta > alpha_t && theta < (pi/3) - alpha_t
            T1 = (sin( (pi/3) - theta ) / ( sin( (pi/3) - theta ) + ...
                sin( theta ) ) ) * Ts;
            T2 = Ts-T1;
            T0=0;
        else
            n2 = Vref*cos( theta );
            n1 = Vref*cos( (pi/3) - theta );
            T1 = (2*Ts/(3*Vbus)) * ((-n1) + (2*n2));
            T2 = (2*Ts/(3*Vbus)) * ((2*n1) - n2);
            T0 = Ts-(T1+T2);
            flag = 1;
        end

    elseif m ≤ 1.10     % Overmodulation mode II
        alpha_h = (pi/6) - acos(1/m);
```

```
45          if theta ≤ alpha_h
46              T1 = Ts;
47              T2=0;
48              T0=0;
49
50          elseif theta > alpha_h && theta < (pi/3) - alpha_h
51              T1 = (sin( (pi/3) - theta ) / ( sin( (pi/3) - theta ) + ...
                    sin( theta ) ) ) * Ts;
52              T2 = Ts-T1;
53              T0=0;
54
55          else
56              T1 = 0;
57              T2 = Ts;
58              T0 = 0;
59          end
60
61      else        % Six Step Mode
62          if theta ≤ (pi/6)
63              T1 = Ts;
64              T2 = 0;
65              T0 = 0;
66          else
67              T1 = 0;
68              T2 = Ts;
69              T0 = 0;
70          end
71
72      end
73
74      if flag == 1
75          % Switching Sequence Determination
76          Q1 = (cos(theta) - (Vref/Vbus)) * T1;
77          Q2 = (cos( (pi/3) - theta) - (Vref/Vbus)) * T2;
78          Q0 = - (Vref/Vbus)*T0;
79          D = sin(theta) * T1;
80
81          %Error Functions (Squared) for Different Switching Sequences
82          F_0127 = ( (1/3) * ( (0.5*Q0).^2 ) * T0 / (2*Ts) ) + ( (1/3) ...
                * ( ( (0.5*Q0).^2 ) + ( 0.5*Q0* ( (0.5*Q0) + Q1 ) ) + ( ( ...
                1* ( (0.5*Q0) + Q1 ) ).^2) ) * (T1/Ts) ) + ( (1/3) * ( ( ...
                ( (0.5*Q0) + Q1).^2) - ( ( (0.5*Q0) + Q1) * (0.5*Q0) ) + ...
                ( (0.5*Q0).^2 ) ) * (T2/Ts) ) + ( (1/3) * ( (0.5*Q0).^2 ) ...
                * ( T0 / (2*Ts) ) ) + ( (1/3) * (D.^2) * ( (T1+T2) / Ts ) );
83
84          F_012 = ( (4/27) * (Q0.^2) * (T0/Ts) ) + ( (4/27) * ( (Q0.^2) ...
                + ( Q0*(Q0+Q1) ) + ( (Q0+Q1).^2 ) ) * (T1/Ts) ) + ( ...
                (4/27) * ( (Q0+Q1).^2 ) * (T2/Ts) ) + ( (4/27) * (D.^2) * ...
```

```
        ( (T1+T2) / Ts) );
85
86      F_721 = ( (4/27) * (Q0.^2) * (T0/Ts) ) + ( (4/27) * ( (Q0.^2) ...
            + ( Q0*(Q0+Q2) ) + ( (Q0+Q2).^2 ) ) * (T2/Ts) ) + ( ...
            (4/27) * ( (Q0+Q2).^2 ) * (T1/Ts) ) + ( (4/27) * (D.^2) * ...
            ( (T1+T2) / Ts) );
87
88      F_0121 = ( (1/3) * (Q0.^2) * (T0/Ts) ) + ( (1/3) * ( (Q0.^2) ...
            + ( Q0*( Q0 + (0.5*Q1) ) ) + ( ( Q0 + (0.5*Q1) ).^2 ) ) * ...
            ( T1 / (2*Ts) ) ) + ( (1/3) * ( ( ( Q0 + (0.5*Q1) ).^2 ) ...
            - ( ( Q0 + (0.5*Q1) ) * (0.5*Q1) ) + ( (0.5*Q1).^2 ) ) * ...
            (T2/Ts) ) + ( (1/3) * ( (0.5*Q1).^2 ) * (T1 / (2*Ts) ) ) ...
            + ( (1/3) * ( (0.5*D).^2 ) * ( (T1+T2) / Ts ) );
89
90      F_7212 = ( (1/3) * (Q0.^2) * (T0/Ts) ) + ( (1/3) * ( (Q0.^2) ...
            + ( Q0*( Q0 + (0.5*Q2) ) ) + ( ( Q0 + (0.5*Q2) ).^2 ) ) * ...
            ( T2 / (2*Ts) ) ) + ( (1/3) * ( ( ( Q0 + (0.5*Q2) ).^2 ) ...
            - ( ( Q0 + (0.5*Q2) ) * (0.5*Q2) ) + ( (0.5*Q2).^2 ) ) * ...
            (T1/Ts) ) + ( (1/3) * ( (0.5*Q2).^2 ) * (T2 / (2*Ts) ) ) ...
            + ( (1/3) * ( (0.5*D).^2 ) * ( (T1+T2) / Ts ) );
91
92      F_1012 = ( (1/3) * ( ( 0.5*Q1).^2 ) * (T1 / (2*Ts) ) ) + ( ...
            (1/3) * ( ( (0.5*Q1).^2 ) + ( 0.5*Q1*( (0.5*Q1) + Q0 ) ) ...
            + ( ( (0.5*Q1) + Q0 ).^2 ) ) * (T0/Ts) ) + ( (1/3) * ( ( ...
            ( (0.5*Q1) + Q0 ).^2 ) - ( ( (0.5*Q1) + Q0 )*Q2 ) + ...
            (Q2.^2) ) * ( T1 / (2*Ts) ) ) + ( (1/3) * (Q2.^2) * ...
            (T2/Ts) ) + ( (1/3) * ( (0.5*D).^2 ) * ( T1 / (2*Ts) ) ) ...
            + ( ( (0.5*D).^2 ) * (T0/Ts) ) + ( (1/3) * ( ( (0.5*D).^2 ...
            ) + ( (0.5*D) * D ) + (D.^2) ) * ( T1 / (2*Ts) ) ) + ( ...
            (1/3) * (D.^2) * (T2/Ts) );
93
94      F_2721 = ( (1/3) * ( ( 0.5*Q2).^2 ) * (T2 / (2*Ts) ) ) + ( ...
            (1/3) * ( ( (0.5*Q2).^2 ) + ( 0.5*Q2*( (0.5*Q2) + Q0 ) ) ...
            + ( ( (0.5*Q2) + Q0 ).^2 ) ) * (T0/Ts) ) + ( (1/3) * ( ( ...
            ( (0.5*Q2) + Q0 ).^2 ) - ( ( (0.5*Q2) + Q0 )*Q1 ) + ...
            (Q1.^2) ) * ( T2 / (2*Ts) ) ) + ( (1/3) * (Q1.^2) * ...
            (T1/Ts) ) + ( (1/3) * ( (0.5*D).^2 ) * ( T2 / (2*Ts) ) ) ...
            + ( ( (0.5*D).^2 ) * (T0/Ts) ) + ( (1/3) * ( ( (0.5*D).^2 ...
            ) + ( (0.5*D) * D ) + (D.^2) ) * ( T2 / (2*Ts) ) ) + ( ...
            (1/3) * (D.^2) * (T1/Ts) );
95
96      if numzones == 1
97          F = [F_0127];
98
99      elseif numzones == 3
100         %For Three Zone Hybrid PWM
101         F = [F_0127, F_0121, F_7212];
102
```

```
103         elseif numzones == 5
104             %For Five Zone Hybrid PWM
105             F = [F_0127, F_0121, F_7212, F_1012, F_2721];
106 
107         elseif numzones == 7
108             %For Seven Zone Hybrid PWM
109             F = [F_0127, F_0121, F_7212, F_1012, F_2721, F_012, F_721];
110 
111         end
112 
113         [¬, idx] = min(F);
114 
115         if idx == 1
116             switchseq = "0127";
117         elseif idx == 2
118             switchseq = "0121";
119         elseif idx == 3
120             switchseq = "7212";
121         elseif idx == 4
122             switchseq = "1012";
123         elseif idx == 5
124             switchseq = "2721";
125         elseif idx == 6
126             switchseq = "012";
127         elseif idx == 7
128             switchseq = "721";
129         else
130             switchseq = "0127";
131         end
132 
133     else
134         if theta < (pi/6)
135             switchseq = "0121";
136         else
137             switchseq = "7212";
138         end
139     end
140 
141     % Switching Times calculation of switches
142     Vsamp = Vref*exp(1i*angle);
143 
144     V1 = Vbus*exp(1i*0);
145     V2 = Vbus*exp(1i*(pi/3));
146     V3 = Vbus*exp(1i*(2*pi/3));
147     V4 = Vbus*exp(1i*(3*pi/3));
148     V5 = Vbus*exp(1i*(4*pi/3));
149     V6 = Vbus*exp(1i*(5*pi/3));
150 
```

```
151     V = [V1; V2; V3; V4; V5; V6];
152     V = V - Vsamp;
153     [¬, minVidx] = mink(V, 2);
154 
155 %         minVidx = minVidx(:, 1);
156 
157     if max(minVidx) == 6 && min(minVidx) == 1
158         minVidx = 6;
159     else
160         minVidx = min(minVidx);
161     end
162 
163     Gmax = 0;
164     Gmed_T1 = 0;
165     Gmed_T2 = 0;
166     Glow = 0;
167     if switchseq == "0127"
168         Gmax = T1+T2+(T0/2);
169         Gmed_T1 = T1 + (T0/2);
170         Gmed_T2 = T2+(T0/2);
171         Glow = (T0/2);
172 
173     elseif switchseq == "0121" || switchseq == "1012" || switchseq == ...
            "012"
174         if rem(minVidx, 2) == 1
175             Gmax = T1+T2;
176             Gmed_T1 = T1;
177             Gmed_T2 = T2;
178             Glow = 0;
179         else
180             Gmax = T1+T2+T0;
181             Gmed_T1 = T1+T0;
182             Gmed_T2 = T2+T0;
183             Glow = T0;
184         end
185 
186     elseif switchseq == "7212" || switchseq == "2721" || switchseq == ...
            "721"
187         if rem(minVidx, 2) == 1
188             Gmax = Ts;
189             Gmed_T1 = T1+T0;
190             Gmed_T2 = T2+T0;
191             Glow = T0;
192         else
193             Gmax = Ts-T0;
194             Gmed_T1 = T1;
195             Gmed_T2 = T2;
196             Glow = 0;
```

```
197         end
198
199     end
200
201 %     timings = zeros(6, 3, int32(Ts/Tsys));
202     timings = zeros(6, 3);
203     timings(1, :) = [Gmax        ;     Gmed_T2 ;     Glow         ];
204     timings(2, :) = [Gmed_T1 ;     Gmax        ;     Glow         ];
205     timings(3, :) = [Glow         ;     Gmax        ;     Gmed_T2 ];
206     timings(4, :) = [Glow         ;     Gmed_T1 ;     Gmax        ];
207     timings(5, :) = [Gmed_T2 ;     Glow         ;     Gmax        ];
208     timings(6, :) = [Gmax        ;     Glow         ;     Gmed_T1 ];
209
210     answ = squeeze(real(timings(minVidx, :)));
211 %     size(answ)
212
213     S1 = answ(1);
214 %     S1 = S1(:);
215     S3 = answ(2);
216 %     S3 = S3(:);
217     S5 = answ(3);
218 %     S5 = S5(:);
219     sector = minVidx;
220
221     if switchseq == "0127"
222         swseq = 1;
223     elseif switchseq == "0121"
224         swseq = 2;
225     elseif switchseq == "7212"
226         swseq = 3;
227     elseif switchseq == "1012"
228         swseq = 4;
229     elseif switchseq == "2721"
230         swseq = 5;
231     elseif switchseq == "012"
232         swseq = 6;
233     elseif switchseq == "721"
234         swseq = 7;
235     else
236         swseq = 1;
237     end
238 end
```

```
1 function Vref_angle = get_random_Vref_angle(train_set_size, Vbus)
2     %Generate random numbers inside hexagon to get random values of Vref
3     %and angle
4     xy = generate_random_xy_in_circle(train_set_size, Vbus);
```

```
    [angle, Vref] = cart2pol(xy(:, 1), xy(:, 2));
    angle(angle<0) = angle(angle<0) + (2*pi);
    Vref_angle = [Vref, angle];
end
```

```
function xy = generate_random_xy_in_circle(requiredPoints, radius)
        numPointsIn = 1;
       viscircles([[0 0]], radius);
       hold on;
       xy = zeros(requiredPoints, 2);

        while numPointsIn ≤ requiredPoints
              testx = 2 * radius * rand(1) - radius;
              testy = 2 * radius * rand(1) - radius;
              if ((testx.^2) + (testy.^2)) ≤ (radius.^2)
                    xy(numPointsIn, 1) = testx;
                    xy(numPointsIn, 2) = testy;
                    numPointsIn = numPointsIn + 1;
              end
        end
        plot(xy(:, 1), xy(:, 2), '.', 'MarkerSize', 20);
end
```

For Training Neural Network for SVM

Load and split data to send it for training

```
function [nn, XTest, YTest, XTrain, YTrain] = model_trainer()
    filename = "training_set_4.xlsx";

    file = readmatrix(filename);

    [m,n] = size(file);
    P = 0.80;
    idx = randperm(m);
   fileTrain = file(idx(1:round(P*m)), :);
   fileTest = file(idx(round(P*m)+1:end), :);

    XTrain = fileTrain(:, 1:2);
    YTrain = fileTrain(:, 4:6);
%     YTrain = categorical(fileTrain(:, 7));

    XTest = fileTest(:, 1:2);
    YTest = fileTest(:, 4:6);
%     YTest = categorical(fileTest(:, 7));
```

```
19
20      %Regression Models
21  %     linmdl = fitrsvm(XTrain, YTrain, 'Standardize', true);
22  %     polymdl = fitrsvm(XTrain, YTrain, 'Standardize', true, ...
        'KernelFunction', 'polynomial', 'PolynomialOrder', 3);
23  %     gaumdl = fitrsvm(XTrain, YTrain, 'Standardize', true, ...
        'KernelFunction', 'gaussian');
24  %     gprmdl = fitrgp(XTrain, YTrain);
25
26
27      %Classification Models
28  %     t = templateSVM('KernelFunction','gaussian');
29  %     svmmdl = fitcecoc(XTrain, YTrain, 'Learners', t);
30  %     cnbmdl = fitcnb(XTrain, YTrain, 'OptimizeHyperparameters','auto');
31  %     test_on_NN(dlnet, XTest, YTest);
32
33      %NN Models
34      nn = train_on_NN_2(XTrain, YTrain);
35
36  end
```

Define and Train NN

```
1   function net = train_on_NN_2(XTrain, YTrain)
2       XTrain = featureExpand(XTrain);
3
4      numFeatures = size(XTrain,2);
5      numOutputs = size(YTrain, 2);
6
7      learn_rate = 0.05;
8      mini_batch_size = size(XTrain, 1);
9      learn_rate_drop_factor = 0.5;
10     learn_rate_drop_period = 200;
11     l2_regularization_factor =1e-7;
12     max_epochs = 4000;
13
14     %Defining NN Architcture
15
16      %For multiplication layer
17      lgraph = layerGraph();
18      lgraph = addLayers(lgraph, featureInputLayer(numFeatures, ...
           'Normalization',  'none', 'Name', 'Input Layer'));
19
20      lgraph = addLayers(lgraph, fullyConnectedLayer(20, 'Name', ...
           'Hidden Layer 1 subnet 1'));
21      lgraph = addLayers(lgraph, tanhLayer('Name', 'tanh1 subnet1'));
```

```
22
23      lgraph = addLayers(lgraph, fullyConnectedLayer(20, 'Name', ...
            'Hidden Layer 1 subnet 2'));
24      lgraph = addLayers(lgraph, tanhLayer('Name', 'tanh1 subnet2'));
25
26      lgraph = addLayers(lgraph, multiplicationLayer(2,'Name','multiply'));
27      lgraph = addLayers(lgraph, tanhLayer('Name', 'tanh_multiply'));
28      lgraph = addLayers(lgraph, fullyConnectedLayer(3, 'Name', 'Output ...
            Layer'));
29      lgraph = addLayers(lgraph, sigmoidLayer('Name', 'sigmoid'));
30      lgraph = addLayers(lgraph, regressionLayer('Name', 'out'));
31
32      %Connect layers to form NN
33      lgraph = connectLayers(lgraph,'Input Layer','Hidden Layer 1 ...
            subnet 1');
34      lgraph = connectLayers(lgraph,'Input Layer','Hidden Layer 1 ...
            subnet 2');
35
36      lgraph = connectLayers(lgraph,'Hidden Layer 1 subnet 1', 'tanh1 ...
            subnet1');
37      lgraph = connectLayers(lgraph,'Hidden Layer 1 subnet 2', 'tanh1 ...
            subnet2');
38
39      lgraph = connectLayers(lgraph, 'tanh1 subnet1', 'multiply/in1');
40      lgraph = connectLayers(lgraph, 'tanh1 subnet2', 'multiply/in2');
41
42      lgraph = connectLayers(lgraph, 'multiply', 'tanh_multiply');
43      lgraph = connectLayers(lgraph, 'tanh_multiply', 'Output Layer');
44      lgraph = connectLayers(lgraph, 'Output Layer', 'sigmoid');
45
46      lgraph = connectLayers(lgraph, 'sigmoid', 'out');
47
48      options = trainingOptions('adam', ...
49              'Plots','training-progress', ...
50              'InitialLearnRate', learn_rate, ...
51              'shuffle', 'every-epoch', ...
52              'verbose', false, ...
53              'MiniBatchSize', mini_batch_size, ...
54              'MaxEpochs', max_epochs, ...
55              'LearnRateSchedule', 'piecewise', ...
56              'LearnRateDropPeriod', learn_rate_drop_period, ...
57              'LearnRateDropFactor', learn_rate_drop_factor, ...
58              'L2Regularization', l2_regularization_factor);
59
60       figure
61       plot(lgraph);
62
63      %Train the NN
```

```
64      net = trainNetwork(XTrain, YTrain, lgraph, options);
65  end
```

Feature Expansion

```
1   function new_X = featureExpand(X)
2       new_X = X;
3
4       %Include sinusoidal terms (1st order)
5       new_X(:, 3) = sin(X(:, 1));
6       new_X(:, 4) = sin(X(:, 2));
7       new_X(:, 5) = cos(X(:, 1));
8       new_X(:, 6) = cos(X(:, 2));
9
10      %Include 3rd harmonic sinusoidal terms
11      new_X(:, 7) = sin(3*X(:, 1));
12      new_X(:, 8) = sin(3*X(:, 2));
13      new_X(:, 9) = cos(3*X(:, 1));
14      new_X(:, 10) = cos(3*X(:, 2));
15
16  %Include 5th harmonic sinusoidal terms
17      new_X(:, 11) = sin(5*X(:, 1));
18      new_X(:, 12) = sin(5*X(:, 2));
19      new_X(:, 13) = cos(5*X(:, 1));
20      new_X(:, 14) = cos(5*X(:, 2));
21  %Include 7th harmonic sinusoidal terms
22      new_X(:, 15) = sin(7*X(:, 1));
23      new_X(:, 16) = sin(7*X(:, 2));
24      new_X(:, 17) = cos(7*X(:, 1));
25      new_X(:, 18) = cos(7*X(:, 2));
26  %Include 9th harmonic sinusoidal terms
27      new_X(:, 19) = sin(9*X(:, 1));
28      new_X(:, 20) = sin(9*X(:, 2));
29      new_X(:, 21) = cos(9*X(:, 1));
30      new_X(:, 22) = cos(9*X(:, 2));
31  %Include 11th harmonic sinusoidal terms
32      new_X(:, 23) = sin(11*X(:, 1));
33      new_X(:, 24) = sin(11*X(:, 2));
34      new_X(:, 25) = cos(11*X(:, 1));
35      new_X(:, 26) = cos(11*X(:, 2));
36
37  %Include 13th harmonic sinusoidal terms
38      new_X(:, 27) = sin(13*X(:, 1));
39      new_X(:, 28) = sin(13*X(:, 2));
40      new_X(:, 29) = cos(13*X(:, 1));
41      new_X(:, 30) = cos(13*X(:, 2));
```

```
42
43 %Include 15th harmonic sinusoidal terms
44     new_X(:, 31) = sin(15*X(:, 1));
45     new_X(:, 32) = sin(15*X(:, 2));
46     new_X(:, 33) = cos(15*X(:, 1));
47     new_X(:, 34) = cos(15*X(:, 2));
48
49 %Include 17th harmonic sinusoidal terms
50     new_X(:, 35) = sin(17*X(:, 1));
51     new_X(:, 36) = sin(17*X(:, 2));
52     new_X(:, 37) = cos(17*X(:, 1));
53     new_X(:, 38) = cos(17*X(:, 2));
54
55  %Include 19th harmonic sinusoidal terms
56     new_X(:, 39) = sin(19*X(:, 1));
57     new_X(:, 40) = sin(19*X(:, 2));
58     new_X(:, 41) = cos(19*X(:, 1));
59     new_X(:, 42) = cos(19*X(:, 2));
60
61 %Include 21th harmonic sinusoidal terms
62     new_X(:, 43) = sin(21*X(:, 1));
63     new_X(:, 44) = sin(21*X(:, 2));
64     new_X(:, 45) = cos(21*X(:, 1));
65     new_X(:, 46) = cos(21*X(:, 2));
66
67 %Include 23th harmonic sinusoidal terms
68     new_X(:, 47) = sin(23*X(:, 1));
69     new_X(:, 48) = sin(23*X(:, 2));
70     new_X(:, 49) = cos(23*X(:, 1));
71     new_X(:, 50) = cos(23*X(:, 2));
72 end
```

Plotting Output vs Input

```
1  function plotfigs()
2      x = 0:0.01:1.2;
3      y = 0:0.1:(2*pi);
4
5      [xx, yy] = meshgrid(x,y);
6
7      subplot(2,2,1);
8      [¬, z, ¬, ¬, ¬] = arrayfun(@create_training_set_row, xx, yy);
9      surf(xx, yy, z);
10     xlabel('m');
11     ylabel('angle');
12     zlabel('S1');
```

```
    title('S1');

    subplot(2,2,2);
    [¬, ¬, z, ¬, ¬] = arrayfun(@create_training_set_row, xx, yy);
    surf(xx, yy, z);
    xlabel('m');
    ylabel('angle');
    zlabel('S3');
    title('S3');

    subplot(2,2,3);
    [¬, ¬, ¬, z, ¬] = arrayfun(@create_training_set_row, xx, yy);
    surf(xx, yy, z);
    xlabel('m');
    ylabel('angle');
    zlabel('S5');
    title('S5');

    subplot(2,2,4);
    [¬, ¬, ¬, ¬, z] = arrayfun(@create_training_set_row, xx, yy);
    surf(xx, yy, z);
    xlabel('m');
    ylabel('angle');
    zlabel('switchseq');
    title('Switching Sequence');
end
```

Plotting Dominant Zones Inside a Sector

```
function plothexagon()
Vbus = 800;
x =linspace(0, Vbus, 1000);
y = linspace(0, Vbus, 1000);

xyidx=0;

for i=x
    for j=y
        if ((j-(sqrt(3)*i)) ≤ 0) && (j-(sqrt(3)*(Vbus-i)) ≤ 0)
            xyidx = xyidx+1;
            xy(xyidx, :) = [i, j];
        end
    end
end

%1 for 0127, 2 for 0121 and 3 for 7212
```

```
pointsize = 10;
type = zeros(size(xy, 1), 1);
for i = 1:size(xy,1)
    [theta, Vref] = cart2pol(xy(i, 1), xy(i,2));
    type(i) = switchtimes(Vref, theta, Vbus);
end

colorID = zeros(length(xy),3);
colorID(type == 1, 1) = 1*0.5;
colorID(type == 2, 2) = 1*0.5;
colorID(type == 3, 3) = 1*0.5;

scatter(xy(:,1), xy(:, 2), pointsize, colorID, 'filled');

colormap([1 0 0;0 1 0;0 0 1]*0.5)
chb = colorbar();
caxis([0,1])
set(chb,'Ticks',0.33:0.33:1,'TickLabels',{'0127', '0121', '7212'})
end
```

```
function [swseq] = switchtimes(Vref, angle, Vbus)
     numzones = 3;
    %     Method 2
    Ts = 1e-4;
    m = (2/sqrt(3))*(Vref/Vbus);
    theta = rem(angle, (pi/3));



        n2 = Vref*cos( theta );
        n1 = Vref*cos( (pi/3) - theta );

        T1 = (2*Ts/(3*Vbus)) * ((-n1) + (2*n2));
        T2 = (2*Ts/(3*Vbus)) * ((2*n1) - n2);
        T0 = Ts-(T1+T2);

    % Switching Sequence Determination
    Q1 = (cos(theta) - (Vref/Vbus)) * T1;
    Q2 = (cos( (pi/3) - theta) - (Vref/Vbus)) * T2;
    Q0 = - (Vref/Vbus)*T0;
    D = sin(theta) * T1;

    %Error Functions (Squared) for Different Switching Sequences
    F_0127 = ( (1/3) * ( (0.5*Q0).^2 ) * T0 / (2*Ts) ) + ( (1/3) * ( ...
        ( (0.5*Q0).^2 ) + ( 0.5*Q0* ( (0.5*Q0) + Q1 ) ) + ( ( 1* ( ...
        (0.5*Q0) + Q1 ) ).^2) ) * (T1/Ts) ) + ( (1/3) * ( ( ( ...
        (0.5*Q0) + Q1).^2) - ( ( (0.5*Q0) + Q1) * (0.5*Q0) ) + ( ...
```

```
        (0.5*Q0).^2 ) ) * (T2/Ts) ) + ( (1/3) * ( (0.5*Q0).^2 ) * ( ...
        T0 / (2*Ts) ) ) + ( (1/3) * (D.^2) * ( (T1+T2) / Ts ) );
25
26      F_012 = ( (4/27) * (Q0.^2) * (T0/Ts) ) + ( (4/27) * ( (Q0.^2) + ( ...
        Q0*(Q0+Q1) ) + ( (Q0+Q1).^2 ) ) * (T1/Ts) ) + ( (4/27) * ( ...
        (Q0+Q1).^2 ) * (T2/Ts) ) + ( (4/27) * (D.^2) * ( (T1+T2) / ...
        Ts) );
27
28      F_721 = ( (4/27) * (Q0.^2) * (T0/Ts) ) + ( (4/27) * ( (Q0.^2) + ( ...
        Q0*(Q0+Q2) ) + ( (Q0+Q2).^2 ) ) * (T2/Ts) ) + ( (4/27) * ( ...
        (Q0+Q2).^2 ) * (T1/Ts) ) + ( (4/27) * (D.^2) * ( (T1+T2) / ...
        Ts) );
29
30      F_0121 = ( (1/3) * (Q0.^2) * (T0/Ts) ) + ( (1/3) * ( (Q0.^2) + ( ...
        Q0*( Q0 + (0.5*Q1) ) + ( ( Q0 + (0.5*Q1) ).^2 ) ) * ( T1 / ...
        (2*Ts) ) ) + ( (1/3) * ( ( ( Q0 + (0.5*Q1) ).^2 ) - ( ( Q0 + ...
        (0.5*Q1) ) * (0.5*Q1) ) + ( (0.5*Q1).^2 ) ) * (T2/Ts) ) + ( ...
        (1/3) * ( (0.5*Q1).^2 ) * (T1 / (2*Ts) ) ) + ( (1/3) * ( ...
        (0.5*D).^2 ) * ( (T1+T2) / Ts ) );
31
32      F_7212 = ( (1/3) * (Q0.^2) * (T0/Ts) ) + ( (1/3) * ( (Q0.^2) + ( ...
        Q0*( Q0 + (0.5*Q2) ) + ( ( Q0 + (0.5*Q2) ).^2 ) ) * ( T2 / ...
        (2*Ts) ) ) + ( (1/3) * ( ( ( Q0 + (0.5*Q2) ).^2 ) - ( ( Q0 + ...
        (0.5*Q2) ) * (0.5*Q2) ) + ( (0.5*Q2).^2 ) ) * (T1/Ts) ) + ( ...
        (1/3) * ( (0.5*Q2).^2 ) * (T2 / (2*Ts) ) ) + ( (1/3) * ( ...
        (0.5*D).^2 ) * ( (T1+T2) / Ts ) );
33
34      F_1012 = ( (1/3) * ( ( 0.5*Q1).^2 ) * (T1 / (2*Ts) ) ) + ( (1/3) ...
        * ( ( (0.5*Q1).^2 ) + ( 0.5*Q1*( (0.5*Q1) + Q0 ) ) + ( ( ...
        (0.5*Q1) + Q0 ).^2 ) ) * (T0/Ts) ) + ( (1/3) * ( ( ( (0.5*Q1) ...
        + Q0 ).^2 ) - ( ( (0.5*Q1) + Q0 )*Q2 ) + (Q2.^2) ) * ( T1 / ...
        (2*Ts) ) ) + ( (1/3) * (Q2.^2) * (T2/Ts) ) + ( (1/3) * ( ...
        (0.5*D).^2 ) * ( T1 / (2*Ts) ) ) + ( ( (0.5*D).^2 ) * (T0/Ts) ...
        ) + ( (1/3) * ( ( (0.5*D).^2 ) + ( (0.5*D) * D ) + (D.^2) ) * ...
        ( T1 / (2*Ts) ) ) + ( (1/3) * (D.^2) * (T2/Ts) );
35
36      F_2721 = ( (1/3) * ( ( 0.5*Q2).^2 ) * (T2 / (2*Ts) ) ) + ( (1/3) ...
        * ( ( (0.5*Q2).^2 ) + ( 0.5*Q2*( (0.5*Q2) + Q0 ) ) + ( ( ...
        (0.5*Q2) + Q0 ).^2 ) ) * (T0/Ts) ) + ( (1/3) * ( ( ( (0.5*Q2) ...
        + Q0 ).^2 ) - ( ( (0.5*Q2) + Q0 )*Q1 ) + (Q1.^2) ) * ( T2 / ...
        (2*Ts) ) ) + ( (1/3) * (Q1.^2) * (T1/Ts) ) + ( (1/3) * ( ...
        (0.5*D).^2 ) * ( T2 / (2*Ts) ) ) + ( ( (0.5*D).^2 ) * (T0/Ts) ...
        ) + ( (1/3) * ( ( (0.5*D).^2 ) + ( (0.5*D) * D ) + (D.^2) ) * ...
        ( T2 / (2*Ts) ) ) + ( (1/3) * (D.^2) * (T1/Ts) );
37
38        if numzones == 3
39              %For Three Zone Hybrid PWM
40              F = [F_0127, F_0121, F_7212];
```

```
41
42          elseif numzones == 5
43              %For Five Zone Hybrid PWM
44              F = [F_0127, F_0121, F_7212, F_1012, F_2721];
45
46          elseif numzones == 7
47              %For Seven Zone Hybrid PWM
48              F = [F_0127, F_0121, F_7212, F_1012, F_2721, F_012, F_721];
49
50          end
51
52          [¬, idx] = min(F);
53
54          if idx == 1
55              switchseq = "0127";
56          elseif idx == 2
57              switchseq = "0121";
58          elseif idx == 3
59              switchseq = "7212";
60          elseif idx == 4
61              switchseq = "1012";
62          elseif idx == 5
63              switchseq = "2721";
64          elseif idx == 6
65              switchseq = "012";
66          elseif idx == 7
67              switchseq = "721";
68          end
69
70
71      if switchseq == "0127"
72          swseq = 1;
73      elseif switchseq == "0121"
74          swseq = 2;
75      elseif switchseq == "7212"
76          swseq = 3;
77      elseif switchseq == "1012"
78          swseq = 4;
79      elseif switchseq == "2721"
80          swseq = 5;
81      elseif switchseq == "012"
82          swseq = 6;
83      elseif switchseq == "721"
84          swseq = 7;
85      end
86  end
```

To Create Training Data for NN Control

```
function [Vsd_dash, Vsq_dash, Vsd, Vsq, isd_new, isq_new] = ...
    controller_training_set_row_creater(isd_star, isq_star,  isd, ...
    isq, wm, phi, phi_diff, Tsys)
    tau = 0.318e-3;

    %Motor Parameters
    r1 = 1.405;
    l1  = 0.005839;
    r2 = 1.395;
    l2 = 0.005839;
    Lm = 0.1722;
    Rs = r1;

    M = Lm;
    Lr = l1+Lm;
    Ls = l2+Lm;

    sigma = 1 - ((M*M) / (Lr*Ls));
    sigma_r = (Lr/M) - 1;

    G = 1;

    %Set Controller Parameters
    K2 = exp(-r1*Tsys/(sigma*Ls));
    K = Tsys/(sigma*Ls);
    K1 = K*Rs;
    K3 = exp(-Tsys/tau);

    e1 = isd_star-isd;
    e2 = isq_star-isq;

    %Finding Vsd_dash and Vsq_dash
    ycd = (((isd_star - ((K3*e1))) * (1+K1)) - isd) / K;
    ycq = (((isq_star - ((K3*e2))) * (1+K1)) - isq) / K;

    Vsd_dash = ycd;
    Vsq_dash = ycq;

    imr = phi/Lm;
    Tr = Ls/r2;

    wmr = wm + (isq/(Tr*imr));
    Vsd = Vsd_dash + ((1/(G*(1+sigma_r)))*phi_diff) - ...
        (sigma*Ls*wmr*isq/G);
    Vsq = Vsq_dash + (phi*wmr/(G*(1+sigma_r))) + (sigma*Ls*wmr*isd/G);
```

```
43
44
45      %Determine new isd and isq
46      isd_new = isd_star - e1*K3;
47      isq_new = isq_star - e2*K3;
48
49  end
```

```
1   function controller_training_set_creater2()
2       Vbus = 568;
3       train_set_size = 100000;
4       Tsys = 100e-6;
5
6       isd_lim = 7;
7       isq_lim = 23;
8       phi_lim = 1.2;
9
10      %Number of Samples for Each Reference Pair
11      tau = 0.318e-3;
12      num_time_const = 5*10;
13      sample_per_ref = int32(num_time_const * tau / Tsys) %200 %Number ...
            of samples for each pair of references %%%%
14
15      num_sample_ref = int32(train_set_size / sample_per_ref )%Number ...
            of reference pairs taken
16
17      train_set = zeros(train_set_size, 11);
18
19      for i=1:num_sample_ref
20          %Generate valid random values for references
21          ref = rand(1,2);
22          ref(1,1) = ref(1,1)*isd_lim*1.3;
23          ref(1,2) = ref(1,2)*isq_lim*1.3;
24
25  %         Tsys = rand(1,1)*1e-3;
26
27          isd_star = ref(1,1);
28          isq_star = ref(1,2);
29          isd = 0;
30          isq = 0;
31
32          %Run Sequentially for getting isd,isq
33          %Assuming random values of phi, w, d_phi/dt
34          for j=1:sample_per_ref
35              decoupling = rand(1,3);
36              decoupling(1,1) = decoupling(1,1)*200; %Speed Limit = 200 ...
                    rad/s
```

```
            decoupling(1,2) = decoupling(1,2)*phi_lim*1.2;
            decoupling(1,3) = decoupling(1,3)*100;

            idx = ((i-1)*sample_per_ref) + j;

            train_set(idx, 1:4) = [isd_star, isq_star, isd, isq];
            train_set(idx, 5:7) = decoupling;
%             train_set(idx, 8) = Tsys;

            [train_set(idx, 8), train_set(idx, 9), train_set(idx, ...
                10), train_set(idx,11), isd, isq] = ...
                controller_training_set_row_creater( train_set(idx, ...
                1), train_set(idx, 2), train_set(idx, 3), ...
                train_set(idx, 4), train_set(idx, 5), train_set(idx, ...
                6), train_set(idx, 7), Tsys);

        end

end

    size(train_set)

    xlswrite("controller_training_set_27.xlsx", train_set);
end
```

To Train NN for Control

```
function [nn, XTest, YTest, XTrain, YTrain] = model_trainer_controller()
    filename = "controller_training_set_27.xlsx";

    file = readmatrix(filename);

    [m,n] = size(file);
    P = 0.80;
    idx = randperm(m);
   fileTrain = file(idx(1:round(P*m)), :);
   fileTest = file(idx(round(P*m)+1:end), :);

    XTrain = fileTrain(:, 1:7);
    YTrain = fileTrain(:, 10:11);

    XTest = fileTest(:, 1:7);
    YTest = fileTest(:, 10:11);

    %NN Models
    nn = train_on_NN_for_controller(XTrain, YTrain);
```

```
20
21 end
```

```
1  function net = train_on_NN_for_controller(XTrain, YTrain)
2      XTrain = featureExpandForController(XTrain);
3
4     numFeatures = size(XTrain,2);
5     numOutputs = size(YTrain, 2);
6
7     learn_rate = 0.02;
8     mini_batch_size = size(XTrain, 1);
9     learn_rate_drop_factor = 0.8;
10    learn_rate_drop_period = 500;
11    l2_regularization_factor =0.0001;
12    max_epochs = 20000;
13
14    %Defining NN Architcture
15    layers = [
16              featureInputLayer(numFeatures, 'Normalization',  'none', ...
                    'Name', 'Input Layer')
17              fullyConnectedLayer(20, 'Name', 'Hidden Layer ...
                    1','BiasInitializer','zeros', 'BiasLearnRateFactor', 0)
18              fullyConnectedLayer(numOutputs, 'Name', 'Output ...
                    Layer','BiasInitializer','zeros', ...
                    'BiasLearnRateFactor', 0)
19              regressionLayer('Name', 'out')
20              ];
21
22          lgraph = layerGraph(layers);
23
24     options = trainingOptions('adam', ...
25              'Plots','training-progress', ...
26              'InitialLearnRate', learn_rate, ...
27              'shuffle', 'every-epoch', ...
28              'verbose', false, ...
29              'MiniBatchSize', mini_batch_size, ...
30              'MaxEpochs', max_epochs, ...
31              'LearnRateSchedule', 'piecewise', ...
32              'LearnRateDropPeriod', learn_rate_drop_period, ...
33              'LearnRateDropFactor', learn_rate_drop_factor, ...
34              'L2Regularization', l2_regularization_factor);
35
36       figure
37       plot(lgraph);
38
39      %Train the NN
40      net = trainNetwork(XTrain, YTrain, lgraph, options);
```

```
end
```

```
function new_X = featureExpandForController(X)
    new_X = X;

    %Add wmr*isd, wmr*isq, wmr*phi
    new_X(:,8) = X(:,5).*X(:,3);
    new_X(:,9) = X(:,5).*X(:,4);
    new_X(:,10) = X(:,5).*X(:,6);

    new_X(:,11) = 0.01*X(:,3).*(X(:,4)./X(:,6));
    new_X(:,12) = 0.01*X(:,4).*(X(:,4)./X(:,6));
    new_X(:,13) = 0.01*X(:,6).*(X(:,4)./X(:,6));
end
```

```
XTrain = featureExpandForController(XTrain);
y = predict(nn, XTrain);
disp(['Train Set MSE: ', num2str(mean(sqrt((mean((y-YTrain).^2))))) ]);

XTest = featureExpandForController(XTest);
y = predict(nn, XTest);
disp(['Test Set MSE: ', num2str(mean(sqrt((mean((y-YTest).^2))))) ]);
```

```
W1 = nn.Layers(2).Weights;
b1 = nn.Layers(2).Bias;
W2 = nn.Layers(3).Weights;
b2 = nn.Layers(3).Bias;

xlswrite("Controller NN Weights\W1.xlsx", W1);
xlswrite("Controller NN Weights\b1.xlsx", b1);

xlswrite("Controller NN Weights\W2.xlsx", W2);
xlswrite("Controller NN Weights\b2.xlsx", b2);
```

Helper Functions for Plotting

```
type = 2;

if type == 1
    ref_d1 = d1.signals(1).values;
    ref_q1 = q1.signals(1).values;
    ref_f1 = f1.signals(1).values;
```

```
    ref_S1 = S1.signals(1).values;

    meas_d1 = d1.signals(2).values;
    meas_q1 = q1.signals(2).values;
    meas_f1 = f1.signals(2).values;
    meas_S1 = S1.signals(2).values;

else
    ref_d2 = d1.signals(1).values;
    ref_q2 = q1.signals(1).values;
    ref_f2 = f1.signals(1).values;
    ref_S2 = S1.signals(1).values;

    meas_d2 = d1.signals(2).values;
    meas_q2 = q1.signals(2).values;
    meas_f2 = f1.signals(2).values;
    meas_S2 = S1.signals(2).values;
end
```

```
figure;
tim = linspace(0,4,size(ref_S1,1));
plot(tim, ref_S1, 'color', 'red','LineWidth', 2)
hold on;
plot(tim, meas_S1, '--', 'color', 'blue', 'LineWidth', 2);
hold on;
plot(tim, meas_S2, ':', 'color', [0 0.5 0], 'LineWidth', 2);
xlabel('Time (in seconds)');
ylabel('Measured Speed and Reference (in rad/s)');
legend('Reference', 'Ts = 1e-4', 'Ts = 5e-4');
title('Rotor Speed Waveform');
grid on;
hold off;

figure;
tim = linspace(0,4,size(ref_f1,1));
plot(tim, ref_f1, 'color', 'red','LineWidth', 2)
hold on;
plot(tim, meas_f1, '--', 'color', 'blue', 'LineWidth', 2);
hold on;
tim = linspace(0,4,size(ref_f2,1));
plot(tim, meas_f2, ':', 'color', [0 0.5 0], 'LineWidth', 2);
xlabel('Time (in seconds)');
ylabel('Measured imr and Reference (in Amp)');
legend('Reference', 'Ts = 1e-4', 'Ts = 5e-4');
title('Magnetizing Stator Current Waveform');
grid on;
hold off;
```

```
29
30 figure;
31 tim = linspace(0,4,size(ref_d1,1));
32 plot(tim, ref_d1, 'color', 'red','LineWidth', 2)
33 hold on;
34 plot(tim, meas_d1, '--', 'color', 'blue', 'LineWidth', 2);
35 hold on;
36 tim = linspace(0,4,size(ref_d2,1));
37 plot(tim, meas_d2, ':', 'color', [0 0.5 0], 'LineWidth', 2);
38 xlabel('Time (in seconds)');
39 ylabel('Measured d Axis Current and Reference (in A)');
40 legend('Reference', 'Ts = 1e-4', 'Ts = 5e-4');
41 title('d Axis Stator Current Waveform');
42 grid on;
43 hold off;
44
45 figure;
46 tim = linspace(0,4,size(ref_q1,1));
47 plot(tim, ref_q1, 'color', 'red','LineWidth', 2)
48 hold on;
49 plot(tim, meas_q1, '--', 'color', 'blue', 'LineWidth', 2);
50 hold on;
51 plot(tim, meas_q2, ':', 'color', [0 0.5 0], 'LineWidth', 2);
52 xlabel('Time (in seconds)');
53 ylabel('Measured q Axis Current and Reference (in A)');
54 legend('Reference', 'Ts = 1e-4', 'Ts = 5e-4');
55 title('q Axis Stator Current Waveform');
56 grid on;
57 hold off;
```