

# Harmonic Effects of Space Vector Modulation on Induction Motor Performance

Jennifer Vining

Dept. of Electrical and Computer Engineering  
1415 Engineering Drive  
Madison, WI 53706

**Abstract** – This project explores the impact of applying a variety of space vector modulated voltage waveforms – a type of pulse width modulation – to the stator of a squirrel-cage induction machine with a focus on the frequency spectra of the resulting voltage, current, and torque waveforms. Four different space vector modulation (SVM) algorithms are investigated: conventional continuous SVM with the active vectors centered in each half-carrier cycle, and 30°, 60°, and 120° discontinuous SVM. The effects on the squirrel-cage machine are studied for both rated operation and overmodulation.

## I. INTRODUCTION

The problem posed is to find the ripple and harmonic frequency content of squirrel-cage induction machine characteristics (i.e. current, torque, rotor speed, and voltage) when various space vector modulation (SVM) algorithms are applied to the stator voltage. The 30°, 60°, and 120° discontinuous modulation techniques as well as continuous SVM with active pulses centered in each half-carrier cycle were implemented using basic SVM theory from [1]. The theory is described in Section II. Matlab and Simulink provided the programming environment to develop these four SVM algorithms, and all the documented Matlab code and Simulink simulation material may be found in the appendix.

## II. SPACE VECTOR MODULATION IMPLEMENTATION

This project instigates four space vector modulation algorithms – conventional with active vectors placed in the middle of the half-cycle of the carrier and the 30°, 60°, and 120° discontinuous modulation algorithms. Theory tells us to expect the conventional SVM to outperform the discontinuous modulation algorithms with respect to unwanted harmonic content and ripple. One may question the use of discontinuous modulation when faced with this fact. The reason to use discontinuous modulation is to decrease the switching losses through the transistors by periodically clamping one of the three phases to a rail to produce a zero vector. The decrease in switching losses associated with discontinuous modulation allows the system to utilize a higher carrier / switching frequency. However, this analysis only uses one carrier frequency,  $f_c = 15$  kHz. The carrier frequency governs the period in which modulation / switching of the inverter gates occurs.

We want the amplitude of our stator voltage,  $V_s$ , to be  $460 \cdot \sqrt{2/3} V_{in}$  on each phase for rated operation. This means by virtue of the inverter circuit in Fig. 1 and space vector modulation theory that the DC input to the inverter must be

$$V_{in} = V_s \frac{3}{2} \frac{1}{\sin(\pi/3)}, \quad (1)$$

assuming that the magnitude of each space vector is  $(2/3)V_{in}$ . This relation is due to the limit of averaging the two nearest space vectors at phases

$$\theta_k = \sum_{k=1}^6 \frac{\pi}{6} + \frac{\pi}{3} k. \quad (2)$$

With the input voltage as defined in (1), the peak inverter output voltage at each phase  $\theta_k$  cannot exceed  $460 \cdot \sqrt{2/3}$ .

The voltage source inverter is modeled with ideal switches (e.g.  $T_{a1}, T_{a2}, T_{b1}, T_{b2}, T_{c1}, T_{c2}$ ) and an infinitely stiff voltage source at the inverter input terminals. Ideal switch operation assumes no conduction or switching losses in the transistors, and the stiff input voltage requires the capacitance,  $C_{in}$ , across the inverter input terminals to be infinite. The inverter is pictured in Fig. 1 below.

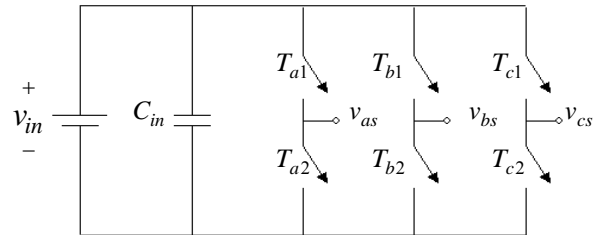


Fig. 1. Ideal three phase inverter

The underlying theory behind space vector modulation is to apply space vectors as illustrated in Fig. 2 for varying time periods in a pattern based on the SVM algorithm. Six space vectors can be obtained in a three phase system through six different combinations of open and closed switches in the inverter shown in Fig. 1. Only one switch may be closed per phase leg in order to prevent a short circuit. The space vectors represent the complex d-q voltage applied to the stator.

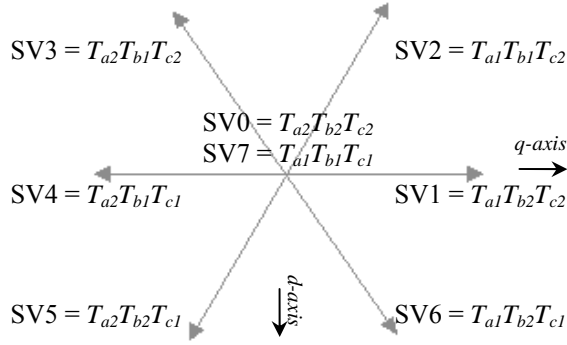


Fig.2. Space vector d,q-axis locations and their corresponding closed switches

### A. Summary of how to apply active vectors

The method of choosing active vectors is the same regardless of which SVM algorithm is used. Basically, a command voltage with the desired magnitude and phase is compared to all the space vectors – in this project, the magnitude and phase of the command voltage were sampled at the beginning of each carrier half-cycle. Two space vectors with phases closest to the phase of the command voltage are chosen. Application of a combination of these space vectors and the zero vector represent the command voltage. The active vectors are applied for a prescribed time period within each half cycle of the carrier based on an averaging effect that effectively yields the correct voltage phase and magnitude. The active times for each space vector are derived in [1] using a geometric relation which places a limit on the output voltage. This limit is based on the fact that the combined active times of each space vector may not exceed half of the carrier period. Manipulation of this relation yields the following equations for the space vector active times within the carrier half-cycle with a command voltage  $V_{cmd} \angle \theta_{cmd}$ .

$$t_{SV(n)} = \frac{V_{cmd} \sin\left(\frac{\pi}{3} - \theta_{cmd}\right)}{2f_c V_{in} \sin\left(\frac{\pi}{3}\right)} \quad (3)$$

$$t_{SV(n+1)} = \frac{V_{cmd} \sin\left(\frac{\pi}{3}\right)}{2f_c V_{in} \sin\left(\frac{\pi}{3}\right)} \quad (4)$$

The first time,  $t_{SV(n)}$ , corresponds to the space vector whose phase angle is smaller.

Once the active times are identified, the only part left is to identify the placement of the zero vectors. The position of the zero vectors in the carrier half-cycle will influence the harmonic content of the voltage waveform which in turn will

affect machine performance. Each of the four SVM algorithms implemented uses a different zero vector arrangement.

### B. Summary of modulation techniques chosen

All of the discontinuous modulation strategies work by eliminating one of the zero vectors, causing the active space vectors in two successive half carrier intervals to join together. The 120° discontinuous modulation algorithm is the most basic of the discontinuous modulation strategies. This algorithm operates by sequentially clamping one phase leg to the upper or lower DC rail for one-third of the fundamental cycle. If the upper DC rail is chosen as the clamp, the SV0 vector is eliminated, but if the lower DC rail is used to clamp, the SV7 vector is eliminated from the SVM algorithm. Since the effect of either method is similar, this project only implements the 120° discontinuous modulation which clamps to the lower DC rail.

Both the 30° and 60° discontinuous modulation strategies work by alternately eliminating the zero space vectors SV0 and SV7. There is only one type of 30° discontinuous modulation which is essentially a variant of 60° discontinuous modulation. These methods clamp all the phase legs to the opposite DC rails in each 60° segment so as to switch between the SV0 and SV7 zero vectors in each 60° segment. Clamping occurs in the 60° intervals between voltage peaks.

There are three different ways of realizing 60° discontinuous modulation: 30° lagging clamp, 30° leading clamp, and 0° clamp. The clamp phase offset relates to where the non-switching periods of each phase leg are positioned relative to the fundamental (command) voltage waveform peaks. Each method calls for a successive inverter phase leg to be unmodulated for 60° of the fundamental, alternating between clamping to the lower (SV0) or upper (SV7) DC rail of the inverter. The 30° lagging, 60° discontinuous modulation strategy was chosen because it is best suited for a system with a lagging power factor of 0.866, which is close to that of our induction machine.

The pulse width modulation technique is regularly sampled since the voltages are implemented as discrete time waveforms. As mentioned previously, the command voltage magnitude and phase is sampled at the beginning of each half-carrier cycle. A look-up table is developed to store the active times for the two space vectors in any one of the six 60° segments. This is applied to each of the four space vector modulation algorithms which were implemented as discretely sampled waveforms using Matlab. The code to develop these waveforms is included in the appendix.

$$\begin{bmatrix} \dot{\Psi}_{ds} \\ \dot{\Psi}_{qs} \\ \dot{\Psi}_{dr} \\ \dot{\Psi}_{qr} \end{bmatrix} = \begin{bmatrix} \frac{\omega_B r_s}{X_{ls}} \left( \frac{X_m^*}{X_{ls}} - 1 \right) & \omega & \frac{\omega_B r_s X_m^*}{X_{ls} X_{lr}} & 0 \\ -\omega & \frac{\omega_B r_s}{X_{ls}} \left( \frac{X_m^*}{X_{ls}} - 1 \right) & 0 & \frac{\omega_B r_s X_m^*}{X_{ls} X_{lr}} \\ \frac{\omega_B r_r X_m^*}{X_{ls} X_{lr}} & 0 & \frac{\omega_B r_r}{X_{lr}} \left( \frac{X_m^*}{X_{lr}} - 1 \right) & (\omega - \omega_r) \\ 0 & \frac{\omega_B r_r X_m^*}{X_{ls} X_{lr}} & (\omega_r - \omega) & \frac{\omega_B r_r}{X_{lr}} \left( \frac{X_m^*}{X_{lr}} - 1 \right) \end{bmatrix} \begin{bmatrix} \Psi_{ds} \\ \Psi_{qs} \\ \Psi_{dr} \\ \Psi_{qr} \end{bmatrix} + \omega_B \begin{bmatrix} v_{ds} \\ v_{qs} \\ v_{dr} \\ v_{qr} \end{bmatrix} \quad (5)$$

$$\begin{bmatrix} \dot{i}_{ds} \\ \dot{i}_{qs} \\ \dot{i}_{dr} \\ \dot{i}_{qr} \end{bmatrix} = \begin{bmatrix} \frac{1}{X_{ls}} \left( 1 - \frac{X_m^*}{X_{ls}} \right) & 0 & -\frac{X_m^*}{X_{ls} X_{lr}} & 0 \\ 0 & \frac{1}{X_{ls}} \left( 1 - \frac{X_m^*}{X_{ls}} \right) & 0 & -\frac{X_m^*}{X_{ls} X_{lr}} \\ -\frac{X_m^*}{X_{ls} X_{lr}} & 0 & \frac{1}{X_{lr}} \left( 1 - \frac{X_m^*}{X_{lr}} \right) & 0 \\ 0 & -\frac{X_m^*}{X_{ls} X_{lr}} & 0 & \frac{1}{X_{lr}} \left( 1 - \frac{X_m^*}{X_{lr}} \right) \end{bmatrix} \begin{bmatrix} \Psi_{ds} \\ \Psi_{qs} \\ \Psi_{dr} \\ \Psi_{qr} \end{bmatrix} \quad (6)$$

$$T_e = \frac{3P}{4} (\lambda_{ds} i_{qs} - \lambda_{qs} i_{ds}) \quad (7)$$

$$\dot{\omega}_r = \frac{P}{2J} (T_e - T_L) \quad (8)$$

### III. INDUCTION MACHINE MODEL

A squirrel-cage induction motor whose equivalent circuit is pictured in Fig. 3 is used in this simulation. The state space model represented in (5) and the accompanying equations (6) – (8) were used to simulate the performance of such a motor. Table II defines the variables used in these equations. Values of the machine parameters can be found in the appendix along with a Simulink model of the induction machine. All calculations were carried out in the stator / stationary reference frame using flux voltages as the state variable.

The Simulink model imports the SVM voltage waveforms from Matlab's workspace since the voltage waveforms were created by the Matlab script listed in the appendix rather than in the Simulink model itself.

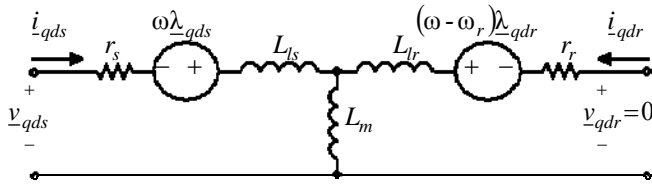


Fig. 3. The d,q equivalent three phase squirrel-cage induction machine circuit

TABLE I  
MACHINE VARIABLE DEFINITIONS

Machine Variables	
$X_{ls}, X_{lr}, X_m$	Stator, rotor, and magnetizing machine impedances ( $\Omega$ )
$X_m^* = (X_{ls} \  X_{lr} \  X_m)$	Parallel combination of impedances ( $\Omega$ )
$r_s, r_r$	Stator and rotor resistances ( $\Omega$ )
$T_e, T_L$	Electromechanical and load torque ( $N \cdot m$ )
$P$	Number of machine poles
$J$	Machine inertia ( $kg \cdot m^2 = N \cdot m \cdot s^2$ )
$\underline{\lambda}_{qds} = \lambda_{qs} - j\lambda_{ds}$	Stator d,q flux ( $V \cdot s$ )
$\underline{i}_{qds} = i_{qs} - ji_{ds}$	Stator d,q current (A)
$\underline{\Psi}_{qds} = \Psi_{qs} - j\Psi_{ds}$	Stator d,q flux voltage, $[\Psi = \omega_B \lambda]$ , (V)
$\underline{v}_{qds} = v_{qs} - jv_{ds}$	Stator d,q voltage (V)
$\omega_r, \omega_B, \omega$	Rotor, base (rated), and reference frame frequency (rad/s)

### IV. RESULTS

The following two sub-sections describe the ripple and harmonic effects of space vector modulation (SVM) on several machine parameters such as the torque, current, and voltage. The first sub-section deals with effects at rated operation while the second evaluates the effects of overmodulated operation.

### A. Rated operation

Table II below lists the steady-state torque, rotor speed, and stator phase a current and voltage in the synchronous reference frame. These values were calculated from the machine equations in Section II with a pure 60 Hz sinusoidal voltage applied to the stator. It is assumed that the stator voltage peak is aligned with the q-axis of the synchronous reference frame. The steady state solution in Table II occurs at rated operation and provides us with a basis for comparison between the SVM methods. The appendix contains documented Matlab code used to evaluate the steady state solution.

TABLE II  
RATED STEADY-STATE VALUES WITH PURE 60 Hz SINUSOIDAL EXCITATION

Steady State Value Stator Voltage	$T_e$ (pu)	$\omega_r$ (pu)	$i_{as}^e$ (pu)	$v_{as}^e$ (pu)
60 Hz Sinusoid	1.05	0.954	1.07	1.0

The following two figures illustrate continuous SVM on stator phase a in the stationary reference frame as compared to a purely sinusoidal voltage waveform at 60 Hz. Fig. 4 includes one full 60 Hz cycle whereas Fig. 5 is a close-up segment of Fig. 4 to illustrate the switching effects of continuous SVM. All of the graphs appearing in this paper were generated using the Matlab scripts in the appendix.

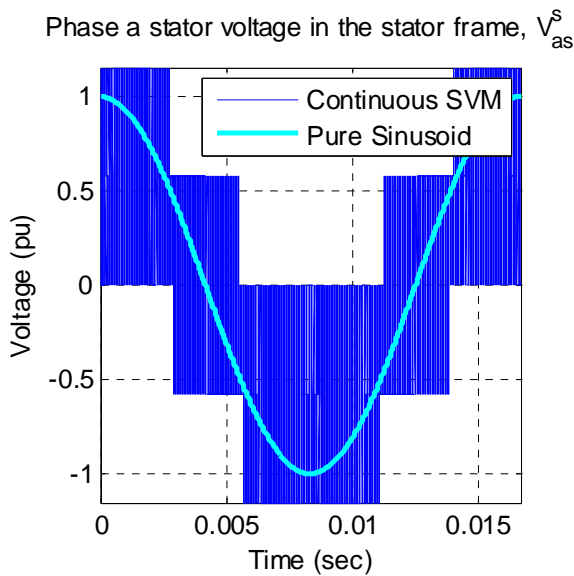


Fig. 4. One cycle of phase a stator voltage in the stationary reference frame

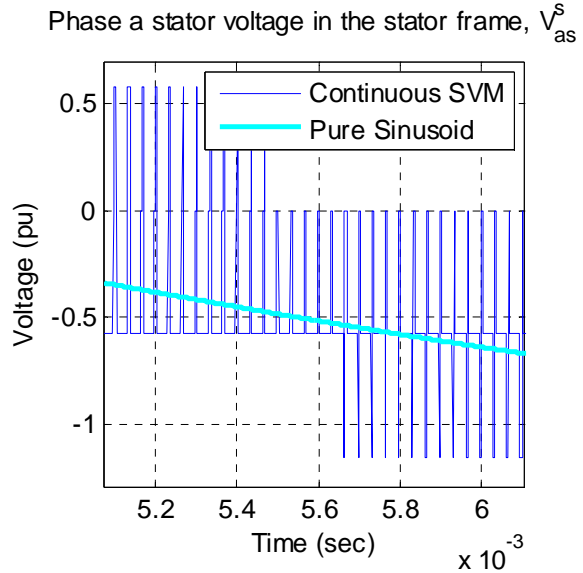


Fig. 5. Zoom-in view of phase a stator voltage

The subsequent figures depict the torque, rotor speed, and stator phase a current and voltage waveforms in the synchronous reference frame. Each of the graphs provides a comparison between operation with a 60 Hz sinusoid and the SVM voltages. Fig. 6 presents a key for each of the graphs.

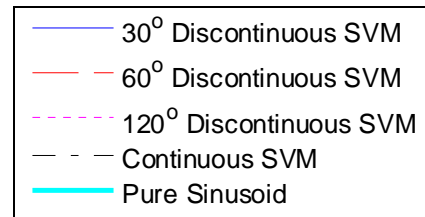


Fig. 6. Waveform key for Figs. 7 - 10 and 17 - 19

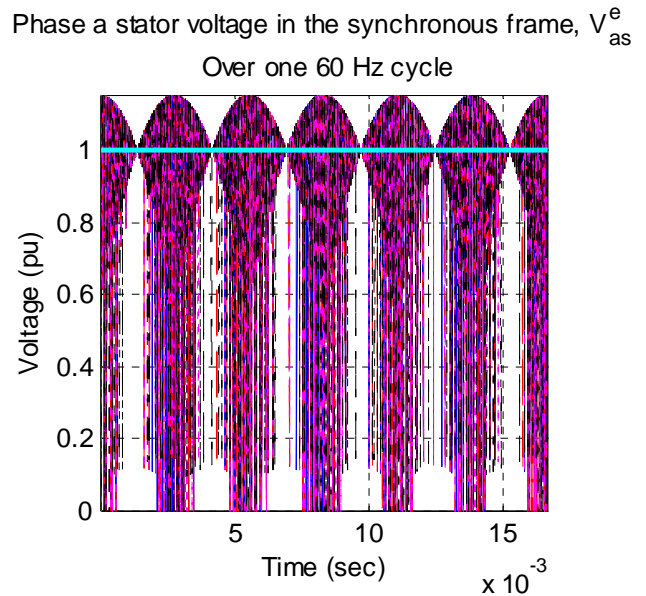


Fig. 7. Phase a stator voltage in the synchronous reference frame

Notice that the sinusoidal voltage in the synchronous reference frame is constant, but the SVM waveforms have large instantaneous voltage spikes. These spikes are what cause unwanted harmonics. The stator current pictured in Fig. 8 attenuates these spikes due to the nature of the induction machine's lowpass filtering properties (i.e.  $i = \int \frac{v}{L} dt$ ). Thanks to this filtering by the machine inductances, the rotor speed and torque of Figs. 9 and 10 are smoothed despite the voltage spikes. In the presence of harmonic losses within the machine, however, the rotor speed is lower for SVM operation than sinusoidal in order to supply the rated torque.

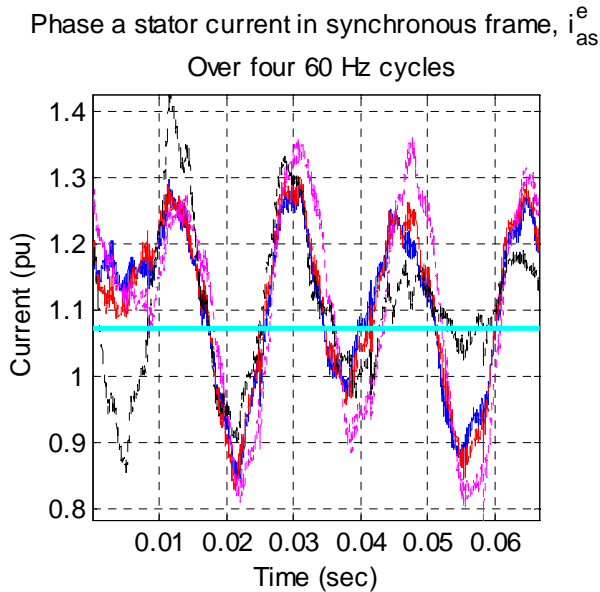


Fig. 8. Phase a stator current in the stationary reference frame

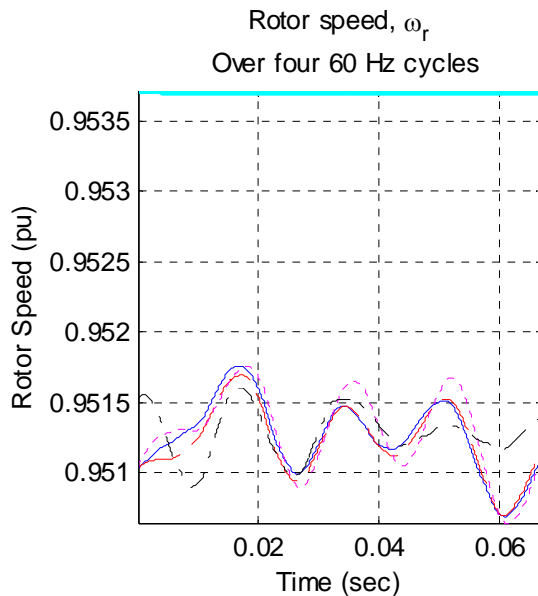


Fig. 9. Rotor speed

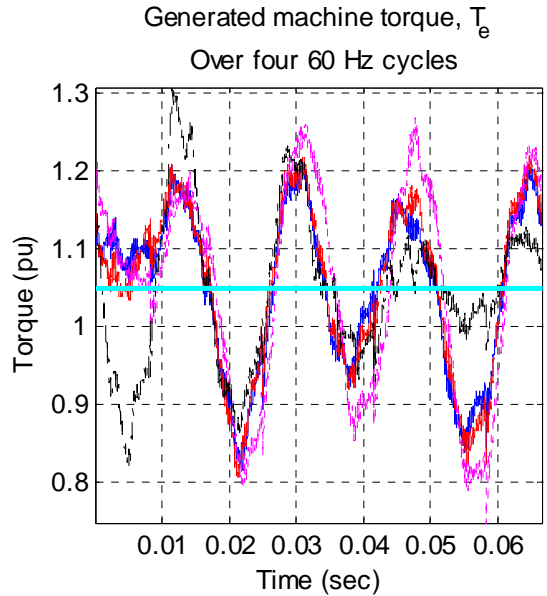


Fig. 10. Torque

The following table compares the ripple in the torque, rotor speed, stator phase a current and voltage waveforms in the synchronous reference frame for all SVM algorithms. A trivial but important result is that the voltage ripple for each algorithm is identical since every SVM scheme relies on switching between the same space vectors. Comparing the SVM performance depends on whether performance is rated in terms of torque ripple or rotor speed ripple. If we rate based on rotor speed ripple, continuous SVM outperforms the others as one would expect. But if we choose torque ripple, 60° discontinuous SVM performs the best contrary to what one would predict. This is due to the fact that the 60° discontinuous SVM method chosen is suited for an inductive load which decreases the current ripple and in turn decreases the torque ripple.

TABLE III  
RIPPLE VALUES FOR ALL SVM ALGORITHMS

Ripple Variable SVM Algorithm	$T_e$ (pu)	$\omega_r$ (rad/s)	$i_{as}^e$ (pu)	$v_{as}^e$ (pu)
30° discontinuous modulation	0.4626	0.4399	0.527	1.1547
60° discontinuous modulation	0.4229	0.4347	0.4886	1.1547
120° discontinuous modulation	0.5197	0.4369	0.5858	1.1547
Continuous modulation with centered active vectors	0.4877	0.2639	0.5752	1.1547

Table IV contains the value of ripple as a percentage of the sinusoidally-excited steady state value. This gives an idea of how much the modulation technique causes the machine characteristics to deviate from the rated value. Overall, the rotor speed does not deviate much, but the torque and current fluctuate substantially from the rated steady-state value.

TABLE IV  
RIPPLE AS A PERCENTAGE OF THE STEADY STATE VALUE

Ripple % Variable SVM Algorithm	$T_e$ (%)	$\omega_r$ (%)	$i_{as}^e$ (%)	$v_{as}^e$ (%)
30° discontinuous modulation	44.05	0.1223	49.26	1.1547
60° discontinuous modulation	40.28	0.1209	45.65	1.1547
120° discontinuous modulation	49.5	0.1215	54.74	1.1547
Continuous modulation with centered active vectors	46.45	0.0734	53.75	1.1547

The baseband, carrier, and sideband harmonics of the stator voltage, current, and machine torque were compared from each voltage excitation method. Equations (9) and (10) define these harmonic components, where  $f(t)$  is a waveform from which we are extracting the harmonic components and  $f_e$  is the electrical frequency whose period is  $T_e$ . All harmonic content was calculated using the Matlab code in the appendix. Since the sideband harmonics were negligible, they are not included in the analysis.

$$n^{th} \text{ Baseband Harmonic} = \frac{1}{T_e} \int_0^{T_e} f(t) e^{jn(2\pi f_e)t} dt \quad (9)$$

$$m^{th} \text{ Carrier Harmonic} = \frac{1}{T_c} \int_0^{T_c} f(t) e^{jm(2\pi f_c)t} dt \quad (10)$$

The ensuing three figures depict the torque and stator phase a current and voltage harmonics from the stationary reference frame. Each of the graphs provides a comparison between operation with a 60 Hz sinusoid and the SVM techniques. Fig. 11 acts as a key for the graphs.

Although the harmonic content of the voltage and current resulting from sinusoidal excitation only contains the fundamental component, it is included as a baseline comparison for the SVM techniques. Interestingly, the baseband harmonic content of every one of the space vector modulated stator voltages and currents, as seen in Figs. 12 and 13, is quite similar with different SVM algorithms

performing better at different harmonic frequencies. In all but the stator voltage, the carrier harmonics of the SVM algorithms follow the harmonics of the sinusoid, leading the observer to conclude that these are artifacts of digital processing. With a lower carrier frequency or higher electrical frequency, we might expect the differences in the harmonics to be more exaggerated, but with the carrier frequency at 15 kHz, the baseband harmonics of each of the algorithms do not distinguish any SVM method as superior over another.

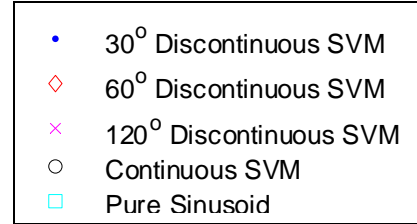


Fig. 11. Key for the graphs in Figs. 12 - 14 and 20 - 22

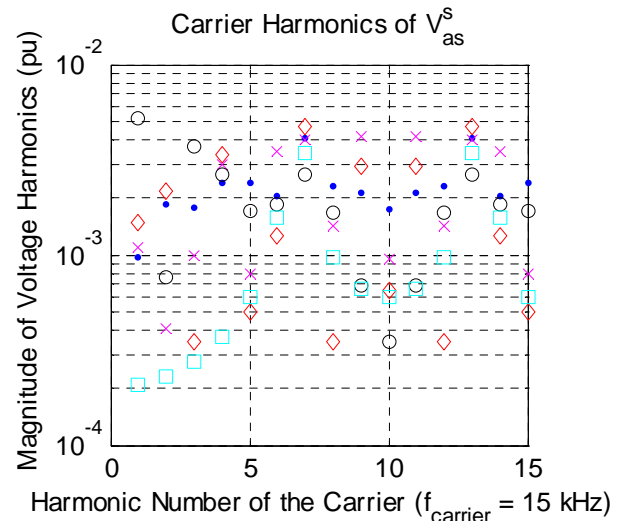
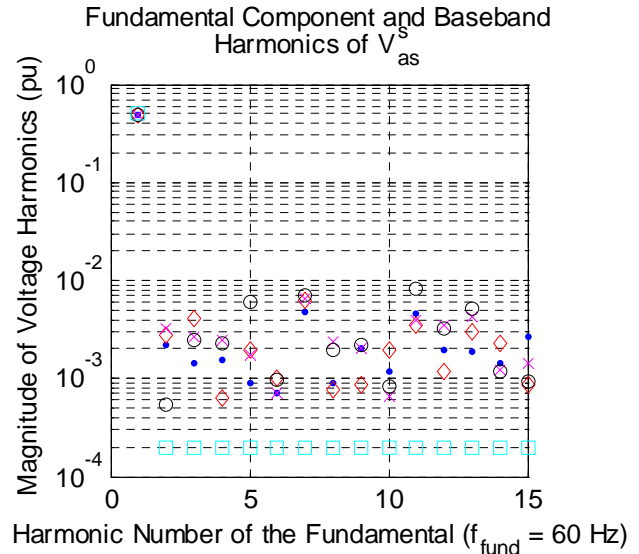


Fig. 12. Harmonics of the stator voltage in the stator reference frame

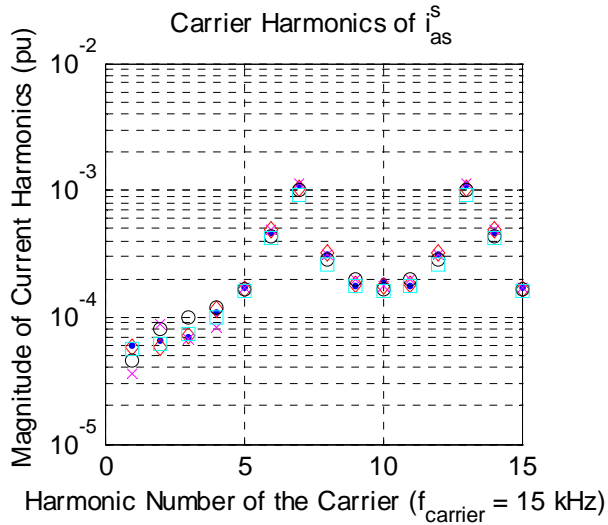
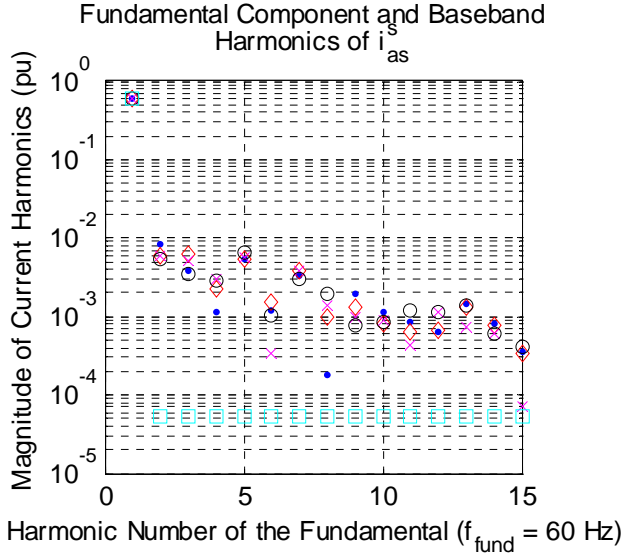


Fig. 13. Harmonics of the stator current in the stator reference frame

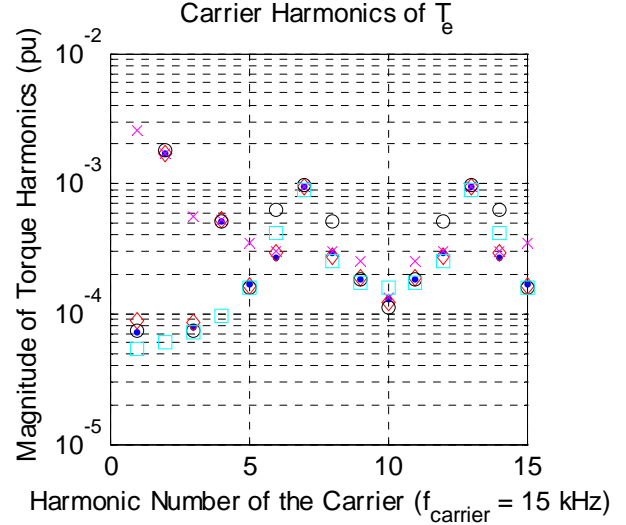
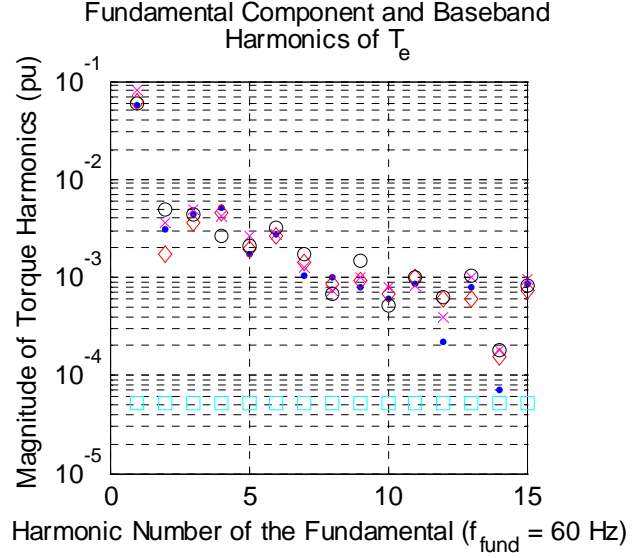


Fig. 14. Harmonics of the torque

### B. Overmodulation

Overmodulation causes the voltage amplitude on each phase to saturate at the maximum input voltage. This in turn causes the magnitude of the stator d-q voltage to vary over time rather than remain constant as is the case with regular inverter operation such as the rated operation just presented. The following questions arise when dealing with overmodulated operation:

- 1) Does the machine current and torque ripple increase relative to rated operation?
- 2) Does the harmonic response of the space vector modulation algorithms degrade substantially from rated operation?
- 3) Do any of the methods yield better overmodulation performance as compared with each other?

In the overmodulation regime, the inverter output voltage magnitude is limited by the input voltage, but the frequency may still increase beyond the rated value. Because of this,

constant V/Hz operation may not be achieved, and we expect the torque to decrease accordingly. A good visualization of this effect comes from the fact that the stator and rotor flux voltage amplitudes do not reach 1.0 p.u. at the higher electrical frequencies with limited voltage. Induction machine theory predicts that the steady state torque will be 0.7 p.u., according to the relation (11) where  $\omega_e$  is  $2\pi f_e$ .

$$T_e = \frac{T_{Rated}}{(\omega_e/\omega_B)} \quad (11)$$

For this analysis, both the excitation frequency and command voltage on the stator are increased by a factor of 1.5 from rated operation. The load torque is scaled to match (11), and the base electrical frequency is held at 60 Hz. The voltage amplitude on each phase is increased according to constant V/Hz operation but clipped at the maximum voltage value, as seen in Fig. 15, according to the limits of the inverter.



Phase a stator voltage in the stator frame,  $V_{as}^s$

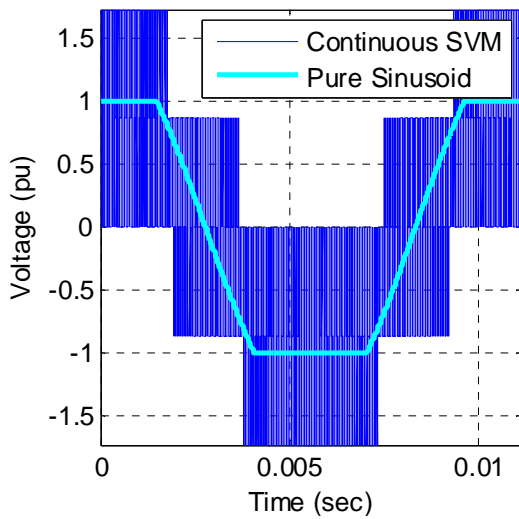


Fig. 15. One cycle of phase a stator voltage in the stationary reference frame

The subsequent figures depict the torque, rotor speed, and stator phase a current and voltage waveforms in the synchronous reference frame. Each of the graphs provides a comparison between operation with the sinusoid pictured in Fig. 15 and the SVM voltages. The same key from Fig. 6 in the previous section applies to Figs. 16 - 19.

Phase a stator voltage in the synchronous frame,  $V_{as}^e$

Over one 90 Hz cycle

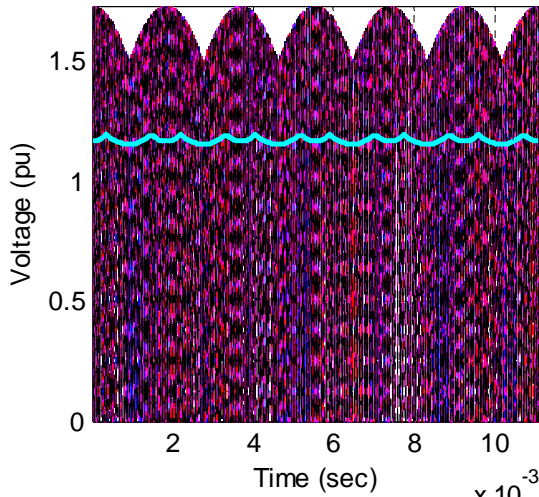


Fig. 16. Phase a stator voltage in the synchronous reference frame

The first thing that the reader should notice is that the stator voltage in the synchronous frame is no longer constant because the amplitude of each phase voltage is clipped, analogous to the command voltage for the SVM algorithms. The performance difference between each of the algorithms is more pronounced now, and we can easily see that the continuous SVM algorithm outperforms the others with respect to torque and rotor speed ripple. As before, the SVM

algorithms yield lower rotor speed than the sinusoidal input, and the torque and current ripple centers around the sinusoidal steady state value which now fluctuates slightly.

There are two resounding conclusions to be made from the graphs in Figs. 17 - 19. The first is that the ripple associated with the continuous SVM algorithm decreases from rated operation, but the most interesting result of all is that the 60° discontinuous SVM scheme performs the poorest of all even though it was one of the best for rated operation.

Phase a stator current in synchronous frame,  $i_{as}^e$

Over four 90 Hz cycles

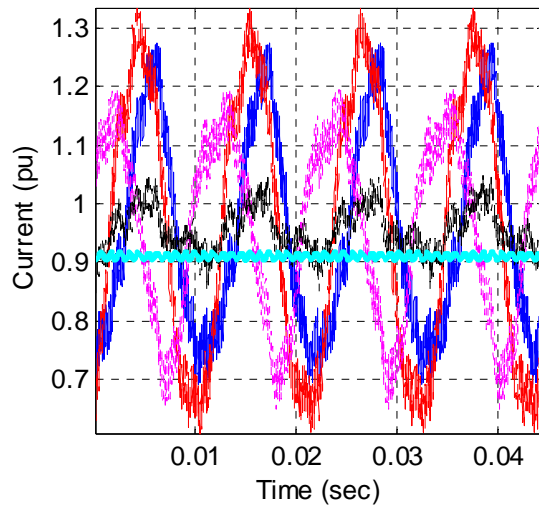


Fig. 17. Phase a stator current in the stationary reference frame

Rotor speed,  $\omega_r$

Over four 90 Hz cycles

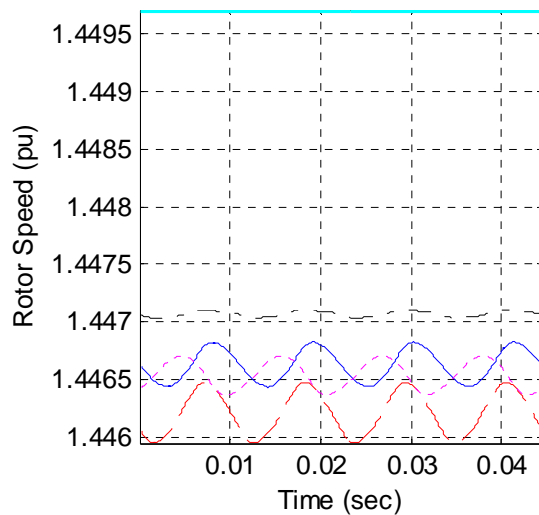


Fig. 18. Rotor speed



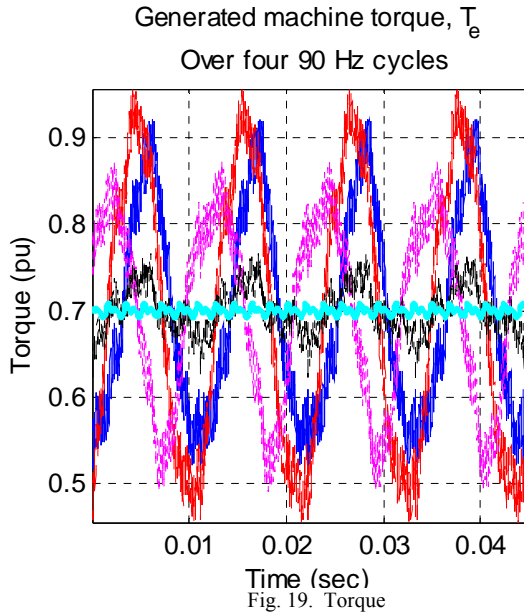


Fig. 19. Torque

Since all the voltage waveforms applied to the stator are distorted due to clipping of the voltage amplitude, the fast fourier transform was taken to compare against the harmonic content. As was expected, the only significant frequency response occurs at the baseband harmonics. Because of this, analysis of the carrier harmonics is omitted.

The harmonic response of the machine to the overmodulated stator voltage is pictured in Figs. 20 - 22. Fig. 11 in the previous section may be used as a key for these graphs. We can see that the baseband harmonics are two orders of magnitude smaller than the fundamental component at 90 Hz, but no SVM algorithm is clearly superior to another. In this case, the ripple parameters give a more apparent indication of which algorithm performs better.

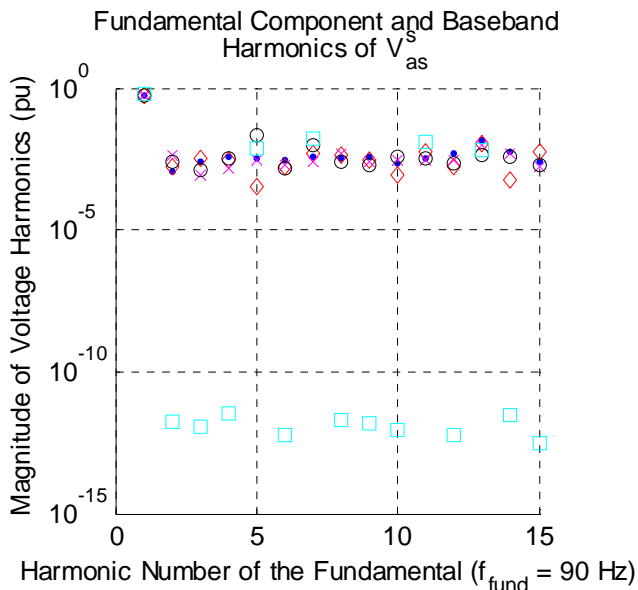


Fig. 20. Harmonics of the stator voltage in the stator reference frame

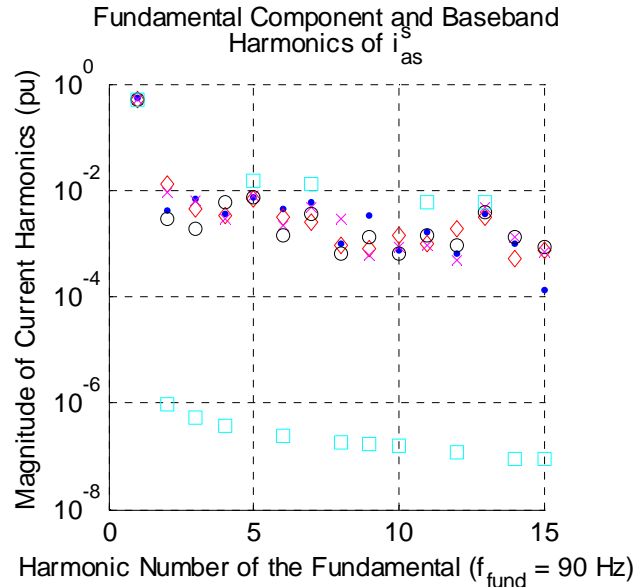


Fig. 21. Harmonics of the stator current in the stator reference frame

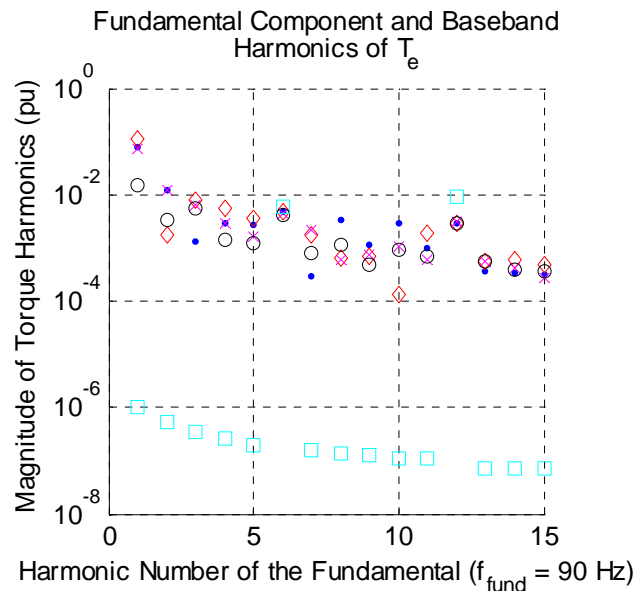


Fig. 22. Harmonics of the torque

#### IV. CONCLUSION

Four different space vector modulation (SVM) algorithms were investigated: conventional continuous SVM with the active vectors centered in each half-carrier cycle, and 30°, 60°, and 120° discontinuous SVM. The impact of these four types of SVM voltage waveforms on a squirrel-cage induction machine were compared with each other and with a sinusoid. The resulting voltage, current, and torque ripple and harmonic content were used for this comparison at both rated operation and overmodulation. It was found that both the 60° discontinuous SVM and continuous SVM methods yielded the best results at rated operation even though the 60° discontinuous SVM performed the worst in overmodulation.

For overmodulated operation, the continuous SVM algorithm offered the best performance. If any of the SVM algorithms were to be used in a real machine, lowpass filtering the SVM

voltage waveform would be necessary to decrease the large torque, current, and rotor speed ripple.

## APPENDIX

### Matlab code for space vector modulation generation and analysis

```

% Project
% ECE 711
%
%   Space Vector Modulation Algorithm
%

%% Induction Machine parameters %%
%
Poles      = 4;           % Number of poles in machine
Pbase      = 37.285e3;    % 3-phase Base power [W]
Vload_rms  = 460/sqrt(3); % Voltage across stator and load [V line-to-neutral]
Vbase      = Vload_rms*sqrt(2); % Base voltage is peak value [V line-to-neutral]
Ibase      = 66.2;       % Base current is peak value [A]
Tbase      = 197.9;     % Base torque [Nm]
sRated     = 0.0463;    % Rated slip of machine
fe         = 1.5 * 60;  % Electrical frequency [Hz]
we         = 2*pi*fe;   % Electrical speed [elec rad/s]
w_base     = 2*pi*60;   % Base electrical speed @ 60Hz [elec rad/s]
w_ref      = 0;        % Reference frame speed [rad/s] -- stator frame
Flux_base  = Vbase / w_base; % Base flux is peak value [V s]
%% assume pf is lagging b/c induction machine
pfRated    = 0.894;     % Rated power factor at stator (cos Theta_rated)
IsRated_pu = 1.2;      % Rated current [pu]
Zbase      = 5.67;     % Base impedance [ohms]
Rs         = 0.087;    % Stator resistance [ohms]
Rs_pu      = 0.0153;   % Stator resistance [pu]
Rr         = 0.228;    % Rotor resistance [ohms]
Rr_pu      = 0.0402;   % Rotor resistance [pu]
Xls        = 0.302;    % Stator leakage impedance [ohms]
Xls_pu     = 0.0532;   % Stator leakage impedance [pu]
Xlr        = Xls;      % Rotor leakage impedance [ohms]
Xlr_pu     = Xls_pu;   % Rotor leakage impedance [pu]
Xm         = 13.08;    % Magnetizing impedance [ohms]
Xm_pu      = 2.31;    % Magnetizing impedance [pu]
Xs         = Xls + Xm; % Stator impedance [ohms]
Xs_pu      = Xls_pu + Xm_pu; % Stator impedance [pu]
Xr         = Xlr + Xm; % Rotor impedance [ohms]
Xr_pu      = Xlr_pu + Xm_pu; % Rotor impedance [pu]
Lm         = Xm / w_base; % Magnetizing inductance [H]
Lr         = Xr / w_base; % Rotor inductance [H]
Ls         = Xs / w_base; % Stator inductance [H]
Llr        = Xlr / w_base; % Rotor leakage inductance [H]
Lls        = Xls / w_base; % Stator leakage inductance [H]
Lm_pu      = Lm * (w_base/Zbase); % Magnetizing inductance [pu]
Lr_pu      = Lr * (w_base/Zbase); % Rotor inductance [pu]
Ls_pu      = Ls * (w_base/Zbase); % Stator inductance [pu]
Llr_pu     = Llr * (w_base/Zbase); % Rotor leakage inductance [pu]
Lls_pu     = Lls * (w_base/Zbase); % Stator leakage inductance [pu]
Xm_star    = ((1/Xm)+(1/Xls)+(1/Xlr))^-1; % Shorted machine impedance [ohms]
Xm_star_pu = Xm_star / Zbase; % Shorted machine impedance [pu]
M          = 1.5;      % Inertial time constant [sec]

%% Calculate machine inertia %%
%
J          = M * (Poles/2)^2 * Pbase / (w_base^2); % [Ns]

%% Calculate the rated angle between stator voltage and current %%
%
pf_angleRated = acos(pfRated); % [rad]

```

```

%%% Calculate rated rotor speed %%%
%%%
wr      = -1 * [(s_rated * we) - we];      % [rad/s]

%%% Calculate load torque constant %%%
%%%
% Tload = k*wr
%
% use steady state: Te_ss - Tload_ss = 0
%      k = Tload_ss / wr_ss
Te_ss_pu = 1.0500612722482;      % [pu]
Te_ss    = Te_ss_pu * Tbase;      % [Nm]
k        = Te_ss / wr;           % [Nms]
k_pu     = Te_ss_pu / (wr/w_base); % [pu]

%%% PWM parameters %%%
%%%
Fc      = 15e3;                  % Carrier frequency is 15kHz [Hz]
Tc      = 1/Fc;                  % Period of carrier signal [sec]
t_Tc    = 0:(Tc/2):1/fc-(Tc/2); % Time vector of fundamental
                                           % with half cycle time steps [sec]

%%% Command voltage vector %%%
%%%
Vmag     = Vbase * (we/w_base);    % Command voltage magnitude for
                                           % constant V/Hz operation [V]
% Note: maximum phase voltage = Vin/sqrt(3) if the magnitude of each
%       space vector is (2/3)*Vin
%
% Vmax,phase = sin(pi/3) * Vmag,SV = (sqrt(3)/2) * Vmag,SV
Vphase   = we.*t_Tc;              % Phase of command voltage at
                                           % each half cycle of the carrier [rad]
                                           % Vqds is aligned with q-axis at t=0
                                           % NOTE: phase is sampled at the
                                           %       beginning of the half cycle!!!
% Vcommand = Vmag .*exp(j.*Vphase); % Command voltage in dq coordinates [V]
                                           % Cannot use this method if we want
                                           % to clip the voltage phase
                                           % magnitudes

a        = exp(j*2*pi/3);
% Calculate individual phase voltages so we can clip the
% phase voltages at the maximum inverter voltage
Vas      = Vmag * cos(Vphase);
lover    = find(Vas > Vbase);
Vas(lover) = Vbase;
lunder   = find(Vas < -Vbase);
Vas(lunder) = -Vbase;
Vbs      = Vmag * cos(Vphase-(2*pi/3));
lover    = find(Vbs > Vbase);
Vbs(lover) = Vbase;
lunder   = find(Vbs < -Vbase);
Vbs(lunder) = -Vbase;
Vcs      = Vmag * cos(Vphase+(2*pi/3));
lover    = find(Vcs > Vbase);
Vcs(lover) = Vbase;
lunder   = find(Vcs < -Vbase);
Vcs(lunder) = -Vbase;
Vcommand = (2/3)*(Vas+a*Vbs+a^2*Vcs); % Command voltage in dq coordinates [V]

%%% Define space vectors %%%
%%%
% first define the magnitude of each space vector to be able to attain Vmag
SVmag    = Vmag / sin(pi/3);
%
% space vectors 0-7
SV1      = SVmag * exp(j*0);
SV2      = SVmag * exp(j*pi/3);
SV3      = SVmag * exp(j*2*pi/3);
SV4      = SVmag * exp(j*pi);

```

```

SV5 = SVMag * exp(j*4*pi/3);
SV6 = SVMag * exp(j*5*pi/3);
SV0 = 0;
SV7 = 0;

SV = [SV1 SV2 SV3 SV4 SV5 SV6 SV7];

%%% Calculate active times for the two space vectors in each 60deg segment %%%
%
% p261
% Note: the phase of Space Vector T1 < the phase of Space Vector T2
% ex. T1 corresponds to SV1
% T2 corresponds to SV2
PHASE = 0:pi/300:pi/3; % define T1 & T2 with many phases so
% we can interpolate to Vphase [rad]
Vas = Vmag * cos(PHASE);
Iover = find(Vas > Vbase);
Vas(Iover) = Vbase;
Iunder = find(Vas < -Vbase);
Vas(Iunder) = -Vbase;
Vbs = Vmag * cos(PHASE-(2*pi/3));
Iover = find(Vbs > Vbase);
Vbs(Iover) = Vbase;
Iunder = find(Vbs < -Vbase);
Vbs(Iunder) = -Vbase;
Vcs = Vmag * cos(PHASE+(2*pi/3));
Iover = find(Vcs > Vbase);
Vcs(Iover) = Vbase;
Iunder = find(Vcs < -Vbase);
Vcs(Iunder) = -Vbase;
Vcmd = (2/3)*(Vas+a*Vbs+a^2*Vcs); % Command voltage in dq coordinates [V]
T1_segment = abs(Vcmd).*sin((pi/3)-PHASE)*(Tc/2)/(SVMag*sin(pi/3)); % [sec]
T2_segment = abs(Vcmd).*sin(PHASE)*(Tc/2)/(SVMag*sin(pi/3)); % [sec]
%
% interpolate for all 60deg segments (0-360deg) from the above 60deg segment
Vphase_mod = mod(Vphase,pi/3); % [rad], vector with a sample for each half cycle period
T1 = interp1(PHASE, T1_segment, Vphase_mod); % [sec], vector with a sample for each half cycle period
T2 = interp1(PHASE, T2_segment, Vphase_mod); % [sec], vector with a sample for each half cycle period

%%% Calculate space vector modulated voltage waveforms %%%
%
samplesPerTc = 80; % number of samples in one carrier cycle
timeStep = Tc/samplesPerTc; % amount of time per sample [sec]
% for best results, should be
% multiple of Tc
t = 0:timeStep:1-fe-timeStep; % time vector for SVM [sec]
NumT1 = floor(T1/timeStep); % number of samples corresponding to T1
% in each half carrier cycle (Tc/2)
NumT2 = floor(T2/timeStep); % [samples]
NumZero = (samplesPerTc/2)-(NumT1+NumT2); % number of samples for the
% zero vector

%%% Continuous and (30, 60, 120 degree) Discontinuous Modulated SVM %%%
%
% Continuous SVM: The best harmonic performance occurs by placing
% active vectors in the middle of the half cycle, Tc/2
%
% 30, 60 degree Discontinuous SVM: Alternately eliminate zero space
% vectors SV0 and SV7 for successive 30, 60 degree segments
%
% 120 degree Discontinuous SVM: Each phase leg in turn is continuously
% locked to the upper or lower DC rail for one-third of the
% fundamental cycle (120 deg)
%
% Therefore, need to split the number of zero vectors to be before and
% after the active vectors
NumZero_begin = floor(NumZero/2); % first half of zero vectors
NumZero_end = ceil(NumZero/2); % second half of zero vectors
%
% Figure out which active space vectors should be used

```

```

%
% Define space vectors by 60deg segment and their corresponding active times
%      T1 | T2
% -----
% Segment 1: SV1 SV2
% Segment 2: SV2 SV3
% Segment 3: SV3 SV4
% Segment 4: SV4 SV5
% Segment 5: SV5 SV6
% Segment 6: SV6 SV1
%
Vphase(1)          = Vphase(1)+0.0000001;   % choose segment for 0deg phase
Segment            = ceil(Vphase/(pi/3));   % calculate segment
ContinuousSVM_Vqds_s = zeros(1,length(t));
discontinuous30_SVM_Vqds_s = zeros(1,length(t));
discontinuous60_SVM_Vqds_s = zeros(1,length(t));
discontinuous120_SVM_Vqds_s = zeros(1,length(t));
%
% Chose clamping reference for the 30, 60 degree discontinuous SVM algorithms
deg30_Sequence    = [0 7 7 0 0 7 7 0 0 7 7 0]; % Choose rail (SV0 or SV7) for each
% 30 deg. segment
I_30degClamp      = floor(Vphase / (pi/6)) + 1; % Identify the 30 deg. segment at
% each half cycle of the carrier
deg30_Clamp       = deg30_Sequence(I_30degClamp); % Assign the clamping rail for each
% half cycle of the carrier
% Implement the 30 deg. lagging clamp for 60deg discontinuous SVM
deg60_Sequence    = [7 0 7 0 7 0]; % Choose rail (SV0 or SV7) for each
% 60 deg. segment
I_60degClamp      = floor(Vphase / (pi/3)) + 1; % Identify the 60 deg. segment at
% each half cycle of the carrier
deg60_Clamp       = deg60_Sequence(I_60degClamp); % Assign the clamping rail for each
% half cycle of the carrier

for halfCycle=1:length(t_Tc)
    Vec1    = Segment(halfCycle);
    Vec2    = mod(Segment(halfCycle),6)+1;
    lbegin  = ((halfCycle-1)*samplesPerTc/2)+1; % index of beginning of this
% half cycle of the carrier
    lend    = halfCycle*samplesPerTc/2; % index of ending of this
% half cycle of the carrier

    ContinuousSVM_Vqds_s(lbegin:lend) = ...
        [SV0*ones(1,NumZero_begin(halfCycle)) ...
         SV(Vec1)*ones(1,NumT1(halfCycle)) ...
         SV(Vec2)*ones(1,NumT2(halfCycle)) ...
         SV0*ones(1,NumZero_end(halfCycle))];

    if (mod(halfCycle,2) == 1) % in first half of carrier period
        if (deg30_Clamp(halfCycle) == 0) % 30deg discont. SVM: tied to lower dc rail, SV0
            discontinuous30_SVM_Vqds_s(lbegin:lend) = ...
                [SV0*ones(1,NumZero(halfCycle)) ...
                 SV(Vec1)*ones(1,NumT1(halfCycle)) ...
                 SV(Vec2)*ones(1,NumT2(halfCycle))];
        else % 30deg discont. SVM: tied to upper dc rail, SV7
            discontinuous30_SVM_Vqds_s(lbegin:lend) = ...
                [SV(Vec1)*ones(1,NumT1(halfCycle)) ...
                 SV(Vec2)*ones(1,NumT2(halfCycle)) ...
                 SV0*ones(1,NumZero(halfCycle))];
        end

        if (deg60_Clamp(halfCycle) == 0) % 60deg discont. SVM: tied to lower dc rail, SV0
            discontinuous60_SVM_Vqds_s(lbegin:lend) = ...
                [SV0*ones(1,NumZero(halfCycle)) ...
                 SV(Vec1)*ones(1,NumT1(halfCycle)) ...
                 SV(Vec2)*ones(1,NumT2(halfCycle))];
        else % 60deg discont. SVM: tied to upper dc rail, SV7
            discontinuous60_SVM_Vqds_s(lbegin:lend) = ...
                [SV(Vec1)*ones(1,NumT1(halfCycle)) ...
                 SV(Vec2)*ones(1,NumT2(halfCycle)) ...
                 SV0*ones(1,NumZero(halfCycle))];
        end
    end
end

```

```

% Lower DC rail is the clamping reference: SV0 at beginning of half cycle
discontinuous120_SVM_Vqds_s(lbegin:lend) = ...
    [SV0*ones(1,NumZero(halfCycle)) ...
    SV(Vec1)*ones(1,NumT1(halfCycle)) ...
    SV(Vec2)*ones(1,NumT2(halfCycle))];

else
    % in second half of carrier period
    if (deg30_Clamp(halfCycle) == 0)
        % 30deg discont. SVM: tied to lower dc rail, SV0
        discontinuous30_SVM_Vqds_s(lbegin:lend) = ...
            [SV(Vec1)*ones(1,NumT1(halfCycle)) ...
            SV(Vec2)*ones(1,NumT2(halfCycle)) ...
            SV0*ones(1,NumZero(halfCycle))];
    else
        % 30deg discont. SVM: tied to upper dc rail, SV7
        discontinuous30_SVM_Vqds_s(lbegin:lend) = ...
            [SV0*ones(1,NumZero(halfCycle)) ...
            SV(Vec1)*ones(1,NumT1(halfCycle)) ...
            SV(Vec2)*ones(1,NumT2(halfCycle))];
    end

if (deg60_Clamp(halfCycle) == 0)
    % 60deg discont. SVM: tied to lower dc rail, SV0
    discontinuous60_SVM_Vqds_s(lbegin:lend) = ...
        [SV(Vec1)*ones(1,NumT1(halfCycle)) ...
        SV(Vec2)*ones(1,NumT2(halfCycle)) ...
        SV0*ones(1,NumZero(halfCycle))];
else
    % 60deg discont. SVM: tied to upper dc rail, SV7
    discontinuous60_SVM_Vqds_s(lbegin:lend) = ...
        [SV0*ones(1,NumZero(halfCycle)) ...
        SV(Vec1)*ones(1,NumT1(halfCycle)) ...
        SV(Vec2)*ones(1,NumT2(halfCycle))];
end

% Lower DC rail is the clamping reference: SV0 at end of half cycle
discontinuous120_SVM_Vqds_s(lbegin:lend) = ...
    [SV(Vec1)*ones(1,NumT1(halfCycle)) ...
    SV(Vec2)*ones(1,NumT2(halfCycle)) ...
    SV0*ones(1,NumZero(halfCycle))];
end
end

% clear SV* lbegin lend Vec1 Vec2 Num* deg* l_ * T1* T2* Segment ...
%   halfCycle Vphase* PHASE timeStep samplesPerTc;

% Confirm correct waveform!
%figure, plot(t, real(Vqds_s)), hold on, plot(t_Tc, real(Vcommand), 'r'), hold off

%% Place in format for use in Simulink %%
%
numCycles = 200; % want 200 cycles of fundamental waveform
%% for some reason, I defined Vqds_s as negative sequence... just reverse signal
continuousSVM.signals.values = repmat(ContinuousSVM_Vqds_s(end:-1:1), [numCycles, 1]);
continuousSVM.time = zeros(numCycles*length(t), 1);
continuousSVM.time(1:length(t)) = t';
continuousSVM.signals.dimensions = 1;

discontinuous30_SVM.signals.values = repmat(discontinuous30_SVM_Vqds_s(end:-1:1)', [numCycles, 1]);
discontinuous30_SVM.time = zeros(numCycles*length(t), 1);
discontinuous30_SVM.time(1:length(t)) = t';
discontinuous30_SVM.signals.dimensions = 1;

discontinuous60_SVM.signals.values = repmat(discontinuous60_SVM_Vqds_s(end:-1:1)', [numCycles, 1]);
discontinuous60_SVM.time = zeros(numCycles*length(t), 1);
discontinuous60_SVM.time(1:length(t)) = t';
discontinuous60_SVM.signals.dimensions = 1;

discontinuous120_SVM.signals.values = repmat(discontinuous120_SVM_Vqds_s(end:-1:1)', [numCycles, 1]);
discontinuous120_SVM.time = zeros(numCycles*length(t), 1);
discontinuous120_SVM.time(1:length(t)) = t';
discontinuous120_SVM.signals.dimensions = 1;

for i=2:numCycles
    lbegin = ((i-1)*length(t))+1; % index of beginning of this

```

```

                                % fundamental cycle
                                % index of ending of this
                                % fundamental cycle
Iend    = i*length(t);
continuousSVM.time(Ibegin:Iend) = continuousSVM.time(Ibegin-1) + t';
discontinuous30_SVM.time(Ibegin:Iend) = discontinuous30_SVM.time(Ibegin-1) + t';
discontinuous60_SVM.time(Ibegin:Iend) = discontinuous60_SVM.time(Ibegin-1) + t';
discontinuous120_SVM.time(Ibegin:Iend) = discontinuous120_SVM.time(Ibegin-1) + t';
end
clear Ibegin Iend numCycles i;

nullSVM.time = 0;
nullSVM.signals.values = 1+j;
nullSVM.signals.dimensions = 1;

```

### % Graph one cycle of continuous SVM

```

Vas1    = Vmag * cos(we.*t);
Iover   = find(Vas1 > Vbase);
Vas1(Iover) = Vbase;
Iunder  = find(Vas1 < -Vbase);
Vas1(Iunder) = -Vbase;
figure('Units', 'inches', 'Position', [4,4,3.45,3.45]), ...
    plot(t, real(ContinuousSVM_Vqds_s)./Vbase, 'k-'), hold on, ...
    plot(t, Vas1/Vbase, 'c-', 'LineWidth', 2), ...
    title('Phase a stator voltage in the stator frame, V ^s_a_s'), ...
    xlabel('Time (sec)'), ylabel('Voltage (pu)'), axis tight, grid, ...
    legend('Continuous SVM', 'Pure Sinusoid', 'Location', 'BestInside'), hold off;

```

### %% steady state matrix: induction machine %%

```

%
% use variables from PWM_SVM.m
%
% steady state values with w_ref=2*pi*60 Hz
% sinusoidal voltage input Vbase
%
w_ref = we;
disp('w_ref = 120*pi')

```

```

SSmatrix = [Rs                -(w_ref*Ls)          0                -(w_ref*Lm);
            w_ref*Ls          Rs                w_ref*Lm          0;
            0                (wr-w_ref)*Lm        Rr                (wr-w_ref)*Lr;
            (w_ref-wr)*Lm     0                (w_ref-wr)*Lr        Rr];
InVector = [0;
            Vbase;
            0;
            0]; %% assume Vqds is aligned with the q-axis in the sync. ref. frame

```

```

CurrentVector = SSmatrix^-1 * InVector;
Ids = CurrentVector(1);  Ids_pu = Ids/Ibase;
Iqs = CurrentVector(2);  Iqs_pu = Iqs/Ibase;
Idr = CurrentVector(3);  Idr_pu = Idr/Ibase;
Iqr = CurrentVector(4);  Iqr_pu = Iqr/Ibase;

```

```

Te      = (3/2)*(Poles/2)*Lm*(-Ids*Iqr + Iqs*Idr);
Te_pu   = Te/Tbase;
flux_dr = Lr*Idr + Lm*Ids;  flux_dr_pu = flux_dr/Flux_base;
flux_qr = Lr*Iqr + Lm*Iqs;  flux_qr_pu = flux_qr/Flux_base;
flux_r  = sqrt(flux_dr^2 + flux_qr^2);
flux_r_pu = flux_r/Flux_base;

```

```

function [FundCompHarmonics, CarrierHarmonics, SidebandHarmonics] = ...
    GetHarmonics(signalAmplitude, signalTime, w_fund, w_carrier, numHarmonics)
%
%function [FundCompHarmonics, CarrierHarmonics, SidebandHarmonics] =

```



```

% GetHarmonics(signalAmplitude, signalTime, numHarmonics)
%
% Compute harmonics of signals created in the Simulink program
% should directly contain all 60Hz harmonics
%
% INPUT:
% signalAmplitude = amplitude of the signal
% signalTime      = time in seconds corresponding to the signal
% w_fund          = fundamental component freq [rad/s]
% w_carrier       = carrier freq [rad/s]
% numHarmonics   = number of harmonics to calculate
%
% OUTPUT:
% FundCompHarmonics = harmonics 1-numHarmonics of the fundamental in signalAmplitude
% CarrierHarmonics  = harmonics 1-numHarmonics of the carrier in signalAmplitude
% SidebandHarmonics = harmonics that are both multiples of the carrier
%                   and fundamental in signalAmplitude
%                   (rows == fund. harmonics)
%                   (columns == carrier harmonics)
%
FundCompHarmonics = zeros(1,numHarmonics);
CarrierHarmonics  = zeros(1,numHarmonics);
SidebandHarmonics = zeros(2*numHarmonics,numHarmonics);

%% Calculating the harmonics %%
%
for n=1:numHarmonics
FundCompHarmonics(n) = (1/length(signalAmplitude)) * sum(signalAmplitude .* exp(j*n*w_fund.*signalTime));
CarrierHarmonics(n)  = (1/length(signalAmplitude)) * sum(signalAmplitude .* exp(j*n*w_carrier.*signalTime));

%% Sideband Harmonics are negligible
% i=1;
% for m=[-numHarmonics:-1 1:numHarmonics]
% SidebandHarmonics(i,n) = (1/length(signalAmplitude)) * ...
% sum(signalAmplitude .* exp(j*(n*w_carrier + m*w_fund).*signalTime));
% i = i+1;
% end
end

function [signal_power, f] = TakeFFT(signalAmplitude, signalTime)
%
%function [signal_power, f] = TakeFFT(signalAmplitude, signalTime)
%
% Compute FFT of signals created in the Simulink program
% FFT should directly contain all 60Hz harmonics
%
Fs = 2^14; %Hz
% interpolate signalAmplitude to ensure consistent sampling period
signalAmplitude = interp1(signalTime, signalAmplitude, 0:1/Fs:1);

fft_length = 2^14; % fft is fastest for powers of 2
signal_fft = fft(signalAmplitude, fft_length);
signal_power = signal_fft * conj(signal_fft) / fft_length;
f = Fs*(-(fft_length/2):(fft_length/2)-1)/fft_length;
signal_power = fftshift(signal_power);

function saveVars(Name, Notes)
%
% Save important variables from Simulink simulation, all in pu
%
```

```

% INPUT:
% Name = name of .mat file to save to
% Notes = optional variable to describe simulation

Idr_s = SIM_I_dr_s;
Ids_s = SIM_I_ds_s;
Iqr_s = SIM_I_qr_s;
Iqs_s = SIM_I_qs_s;
Te = SIM_T_e;
Tload = SIM_Tload_pu;
Vds_s = SIM_Vds_s;
Vflux_dr = SIM_Vflux_dr_pu;
Vflux_ds = SIM_Vflux_ds_pu;
Vflux_qr = SIM_Vflux_qr_pu;
Vflux_qs = SIM_Vflux_qs_pu;
Vqs_s = SIM_Vqs_s;
wr = SIM_wr_pu;

% if nargin < 2
% Notes = {'All variables in pu'};
% end

eval(['save "C:\Documents and Settings\Jennifer Vining\Desktop\JennStuff\ECE711\Project\ Name '.mat" ' ...
      'Idr_s Ids_s Iqr_s Iqs_s Te Tload Vds_s Vflux_dr Vflux_ds Vflux_qr Vflux_qs Vqs_s wr Notes'];

%
% Compute FFT of voltage, current, and torque from the
% continuous and discontinuous SVM algorithms developed in Simulink
% simulation 'Project.mdl'
%
% FFT should directly contain all 60Hz harmonics
%

% Choose an operating regime and input the correct excitation frequency
%OperatingRegime = 'SteadyState'; f_scale = 1;
OperatingRegime = 'Overmod'; f_scale = 1.5;
f_fund = f_scale*60; % fundamental component frequency [Hz]
w_fund = 2*pi*f_fund; % fundamental component frequency [rad/s]
f_carrier = 15e3; % carrier frequency [Hz]
w_carrier = 2*pi*f_carrier; % carrier frequency [rad/s]
numHarmonics = 15; % number of harmonics to compute

VoltageMethod = {'30^o Discontinuous SVM', ...
                 '60^o Discontinuous SVM', '120^o Discontinuous SVM', ...
                 'Continuous SVM', 'Pure Sinusoid'};
signalPrefix = {'discont30', 'discont60', 'discont120', 'cont', 'sinusoid'};
FileName = {'30degDiscontinuousSVM', '60degDiscontinuousSVM', '120degDiscontinuousSVM', ...
           'continuousSVM', 'sinusoid'};
signals = {'Iqs_s', 'Iqr_s', 'Vqs_s', 'Te', 'wr'};
signals2 = {'Ids_s', 'Idr_s', 'Vds_s', 'Te', 'wr'};
titles = [{'Phase a stator current in synchronous frame, i ^e_a_s'}, {'i ^s_a_s'},
         [{'Phase a rotor current in the synchronous frame, i ^e_a_r'}, {'i ^s_a_r'}],
         [{'Phase a stator voltage in the synchronous frame, V ^e_a_s'}, {'V ^s_a_s'}],
         [{'Generated machine torque, T_e'}, {'T_e'}],
         [{'Rotor speed, \omega_r'}, {'\omega_r'}]];
titles2 = [{'Over four ' num2str(f_fund) ' Hz cycles'}],
         [{'Over four ' num2str(f_fund) ' Hz cycles'}],
         [{'Over one ' num2str(f_fund) ' Hz cycle'}],
         [{'Over four ' num2str(f_fund) ' Hz cycles'}],
         [{'Over four ' num2str(f_fund) ' Hz cycles'}]];
ylabels = {'Current (pu)', 'Current (pu)', 'Voltage (pu)', 'Torque (pu)', 'Rotor Speed (pu)'};
ylabels2 = {'Current', 'Current', 'Voltage', 'Torque', 'Rotor Speed'};

lines = {'b-', 'r-', 'm-', 'k-', 'c-', 'LineWidth', 2};
markers = {'b.', 'rd', 'mx', 'ko', 'cs'};

%% Construct figure for the plot of each VoltageMethod, signals combination %%
%
```

```

% for i=1:length(VoltageMethod)
%   for k=1:length(signals)
%     figure('Units', 'inches', 'Position', [4,4,3.45,3.45]);
%   end
% end

%% Construct figure for FFT and harmonic plots of all VoltageMethods for each signal %%
%
for k=1:length(signals)
eval([signals{k} ' = figure("Units", "inches", "Position", [4,4,3.45,3.45]);]);
title({ titles{k,1}, titles2{k} });
eval(['ylabel(" ylabels{k} ")']);
xlabel("Time (sec)"), grid on, box on;

eval([signals{k} ' _fft = figure("Units", "inches", "Position", [4,4,3.45,5.15]);]);
eval(['title({"Frequency Spectrum of ' titles{k,1} '", "' titles{k,2} '")']);
eval(['ylabel("Magnitude of ' ylabels{k} '")']);
xlabel("Frequency (Hz)"), grid on, set(gca, 'YScale', 'log');

eval([signals{k} ' _fundHarmonics = figure("Units", "inches", "Position", [4,4,3.45,3.45]);]);
title('Fundamental Component and Baseband', ['Harmonics of ' titles{k,2} ]),
eval(['ylabel("Magnitude of ' ylabels2{k} ' Harmonics (pu)");']);
xlabel(['Harmonic Number of the Fundamental (f_fund = ' num2str(f_fund) ' Hz)']), ...
  grid on, set(gca, 'YScale', 'log');

eval([signals{k} ' _carrierHarmonics = figure("Units", "inches", "Position", [4,4,3.45,3.45]);]);
eval(['title({"Carrier Harmonics of ' titles{k,2} '")']);
eval(['ylabel("Magnitude of ' ylabels2{k} ' Harmonics (pu)");']);
xlabel(['Harmonic Number of the Carrier (f_carrier = ' num2str(f_carrier/1000) ' kHz)']), ...
  grid on, set(gca, 'YScale', 'log');

eval([signals{k} ' _harmonics = figure("Units", "inches", "Position", [4,4,3.45,6.86]);]);
subplot(2,1,1), title('Fundamental Component and Baseband', ['Harmonics of ' titles{k,2} ]),
  ylabel(['Magnitude of ' ylabels2{k} ' Harmonics (pu)']),
  xlabel(['Harmonic Number of the Fundamental (f_fund = ' num2str(f_fund) ' Hz)']), ...
  grid on, box on, set(gca, 'YScale', 'log');
subplot(2,1,2), title(['Carrier Harmonics of ' titles{k,2} ]),
  ylabel(['Magnitude of ' ylabels2{k} ' Harmonics (pu)']),
  xlabel(['Harmonic Number of the Carrier (f_carrier = ' num2str(f_carrier/1000) ' kHz)']), ...
  grid on, box on, set(gca, 'YScale', 'log')

%% Sideband Harmonics are negligible
% eval([signals{k} ' _sideHarmonics = figure("Units", "inches", "Position", [4,4,3.45,5.15]);]);
% eval(['title({"Sideband Harmonics of ' titles{k,1} '", "' titles{k,2} '")']);
% eval(['ylabel("Magnitude of ' ylabels{k} '")']);
% xlabel(['Harmonic Number of the Fundamental (f_fund = ' num2str(f_fund) ' Hz)']),
% ylabel(['Harmonic Number of the Carrier (f_carrier = ' num2str(f_carrier) ' Hz)']),
% grid on, set(gca, 'ZScale', 'log');
end

for i=1:length(VoltageMethod)
for k=1:length(signals)
eval([signalPrefix {i} ' = load("C:\Documents and Settings\Jennifer Vining\Desktop\ ...
  'JennStuff\ECE711\Project\ OperatingRegime ' _ FileName {i} '.mat', "' signals{k} '");]);

%   figure(k + length(signals)*(i-1)), hold on, eval(['plot(' signalPrefix {i} ' signals{k} ...
%     '.time, ' signalPrefix {i} ' signals{k} .signals.values,')'], hold off;
%   eval(['title({" VoltageMethod {i} ': ' titles{k,1} '", "' titles{k,2} ...
%     '")']);
%   eval(['ylabel(" ylabels{k} ")']);
%   xlabel("Time (sec)"), grid on;

if k~=3 % plot four f_fund cycles for i, Te and wr
  endTime = 0.5 + 4/f_fund;
else
  endTime = 0.5 + 1/f_fund;
end
%endTime = 1.0; % expand the end time for ripple analysis
eval(['itime = find(' signalPrefix {i} ' signals{k} .time >= 0.5 & ' ...
  signalPrefix {i} ' signals{k} .time <= endTime);']);

```

```

% Calculate ripple of each signal
eval(['ripple = max(' signalPrefix {i} '!' signals{k} '.signals.values(Itime))' ...
'-min(' signalPrefix {i} '!' signals{k} '.signals.values(Itime));'];
disp([signalPrefix {i} '!' signals{k} ' ripple = ' num2str(ripple)]);

if k<=3 % move current and voltage into the synchronous frame
eval(['f_ds = load("C:\Documents and Settings\Jennifer Vining\Desktop\' ...
'JennStuff\ECE711\Project\ OperatingRegime \'_ FileName {i} '.mat', "" signals2{k} "");']);

eval(['f_qd_e= (' signalPrefix {i} '!' signals{k} ...
'signals.values - j*f_ds.' signals2{k} '.signals.values) .* exp(-j*w_fund.*' ...
signalPrefix {i} '!' signals{k} '.time);'];
eval(['figure(' signals{k} '), hold on']);
eval(['plot(' signalPrefix {i} '!' signals{k} ...
'.time(Itime)-0.5, real(f_qd_e(Itime)), ' lines {i} '),']); axis tight, hold off;

% Calculate ripple of each signal
ripple = max(real(f_qd_e(Itime)))-min(real(f_qd_e(Itime)));
disp([signalPrefix {i} '!' signals{k} ' SYNC REF FRAME ripple = ' num2str(ripple)]);
else
eval(['figure(' signals{k} '), hold on']);
eval(['plot(' signalPrefix {i} '!' signals{k} '.time(Itime)-0.5, ' signalPrefix {i} ...
'!' signals{k} '.signals.values(Itime), ' lines {i} '),']); axis tight, hold off;
end

%% Take the FFT of signals
eval(['signal_power, f] = TakeFFT(' signalPrefix {i} '!' signals{k} ...
'.signals.values, ' signalPrefix {i} '!' signals{k} '.time);'];
eval(['figure(' signals{k} ' _fft), hold on']);
eval(['plot(f, signal_power, ' lines {i} '), hold off;']);

%% Get the harmonic content
% if i ~ 5 % do not take harmonic content of pure sinusoid, trivial soln
eval(['FundCompHarmonics, CarrierHarmonics, SidebandHarmonics] = ' ...
'GetHarmonics(' signalPrefix {i} '!' signals{k} ...
'.signals.values(Itime), ' signalPrefix {i} '!' signals{k} '.time(Itime), ' ...
'w_fund, w_carrier, numHarmonics);']);

eval(['figure(' signals{k} ' _fundHarmonics)']), hold on,
plot(1:numHarmonics, abs(FundCompHarmonics), markers {i}, 'LineWidth', 1), hold off;

eval(['figure(' signals{k} ' _carrierHarmonics)']), hold on,
plot(1:numHarmonics, abs(CarrierHarmonics), markers {i}, 'LineWidth', 1), hold off;

eval(['figure(' signals{k} ' _harmonics)']), hold on,
subplot(2,1,1), hold on, plot(1:numHarmonics, abs(FundCompHarmonics), markers {i}, 'LineWidth', 1), hold off;
subplot(2,1,2), hold on, plot(1:numHarmonics, abs(CarrierHarmonics), markers {i}, 'LineWidth', 1), hold off;
%% Sideband Harmonics are negligible
% eval(['figure(' signals{k} ' _sideHarmonics)']), hold on;
% x = [-numHarmonics:-1 1:numHarmonics];
% y = ones(1,2*numHarmonics);
% for n=1:numHarmonics
% plot3(x, n*y, abs(SidebandHarmonics(1:2*numHarmonics, n)), markers {i}, ...
% 'LineWidth', 1);
% end
% hold off;
% end
end
end

for k=1:length(signals)
eval(['figure(' signals{k} ');']);
legend(VoltageMethod, 'Location', 'SouthOutside');

eval(['figure(' signals{k} ' _fft);']);
legend(VoltageMethod, 'Location', 'SouthOutside');

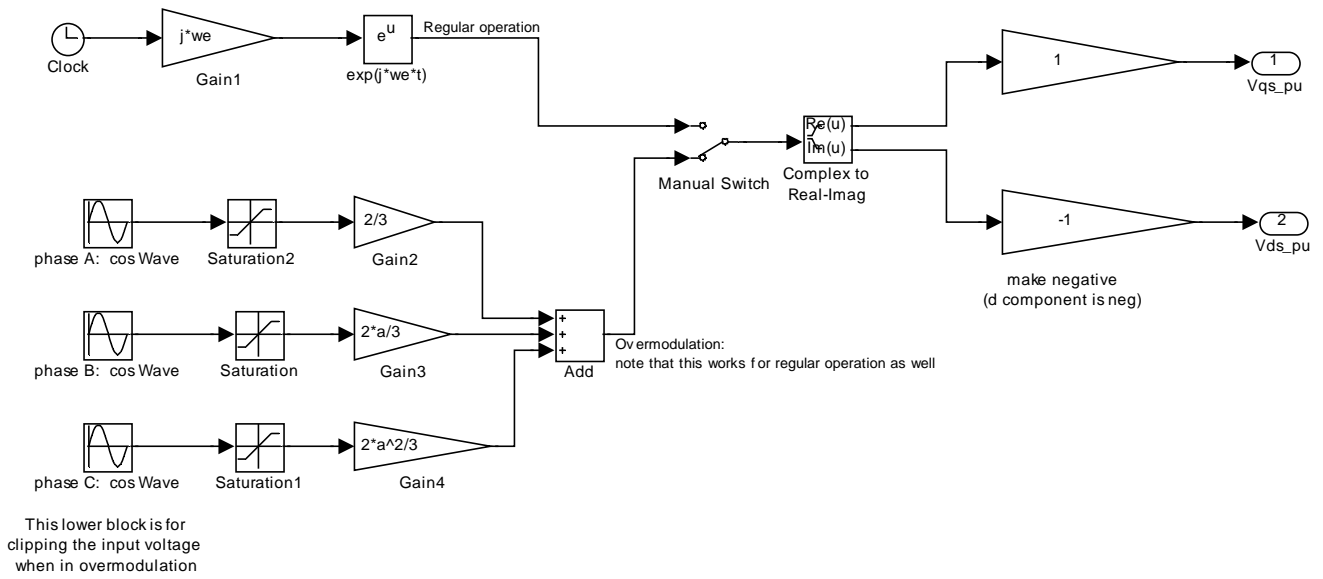
eval(['figure(' signals{k} ' _fundHarmonics);']);

```

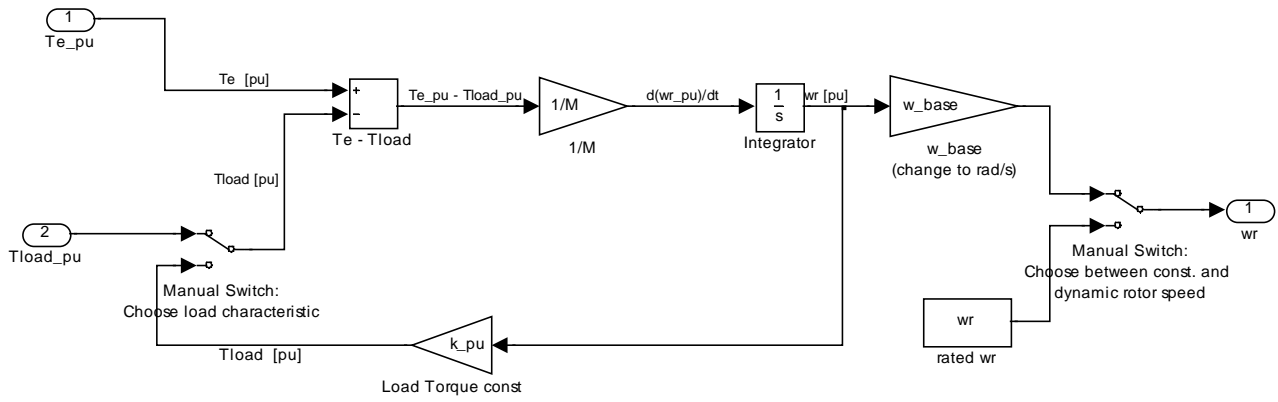
```
legend(VoltageMethod(1:5), 'Location', 'SouthOutside');  
  
eval(['figure(' signals{k} '_carrierHarmonics)']);  
legend(VoltageMethod(1:5), 'Location', 'SouthOutside');  
  
%% Sideband Harmonics are negligible  
eval(['figure(' signals{k} '_sideHarmonics)']);  
legend(VoltageMethod(1:4), 'Location', 'SouthOutside');  
end
```



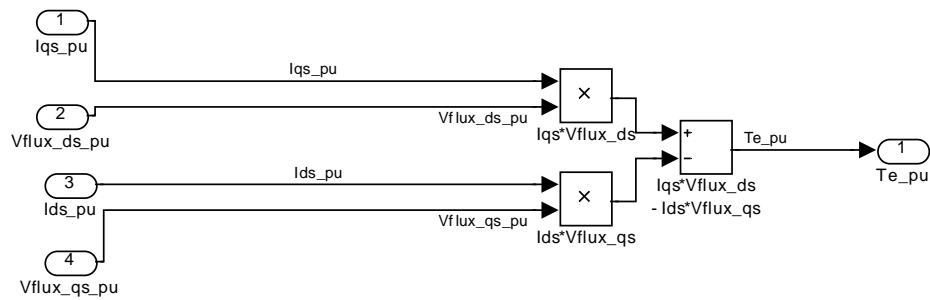
Simulink module *Vqds\_pu* sinusoidal waveform



Simulink module *wr\_rad\_s*

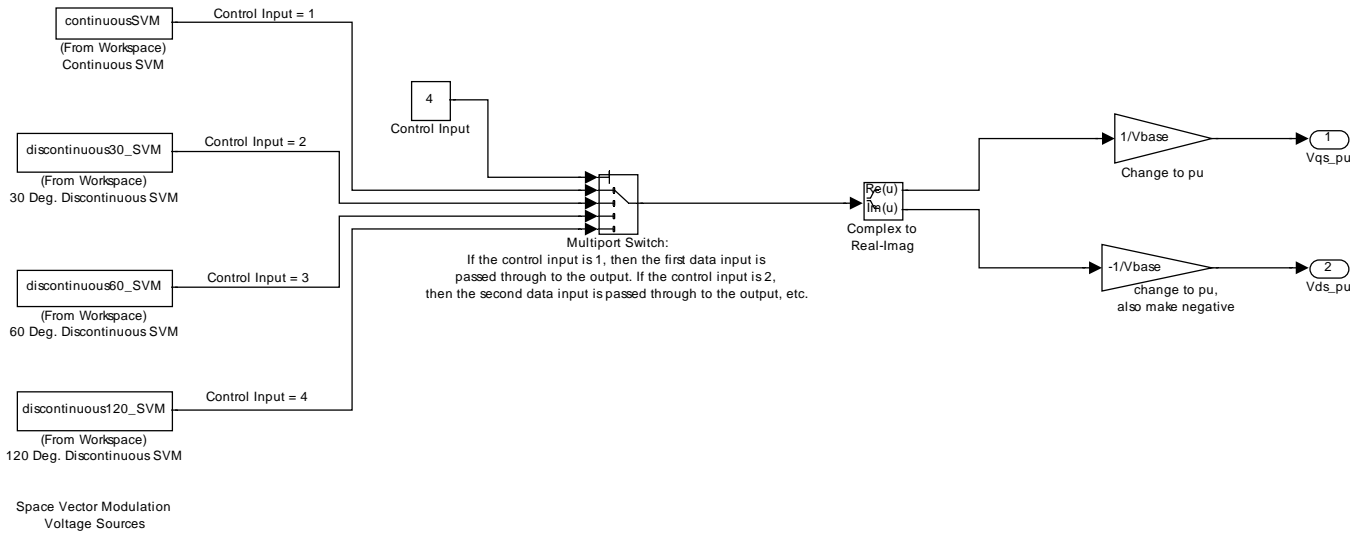


Simulink module *Te\_pu*

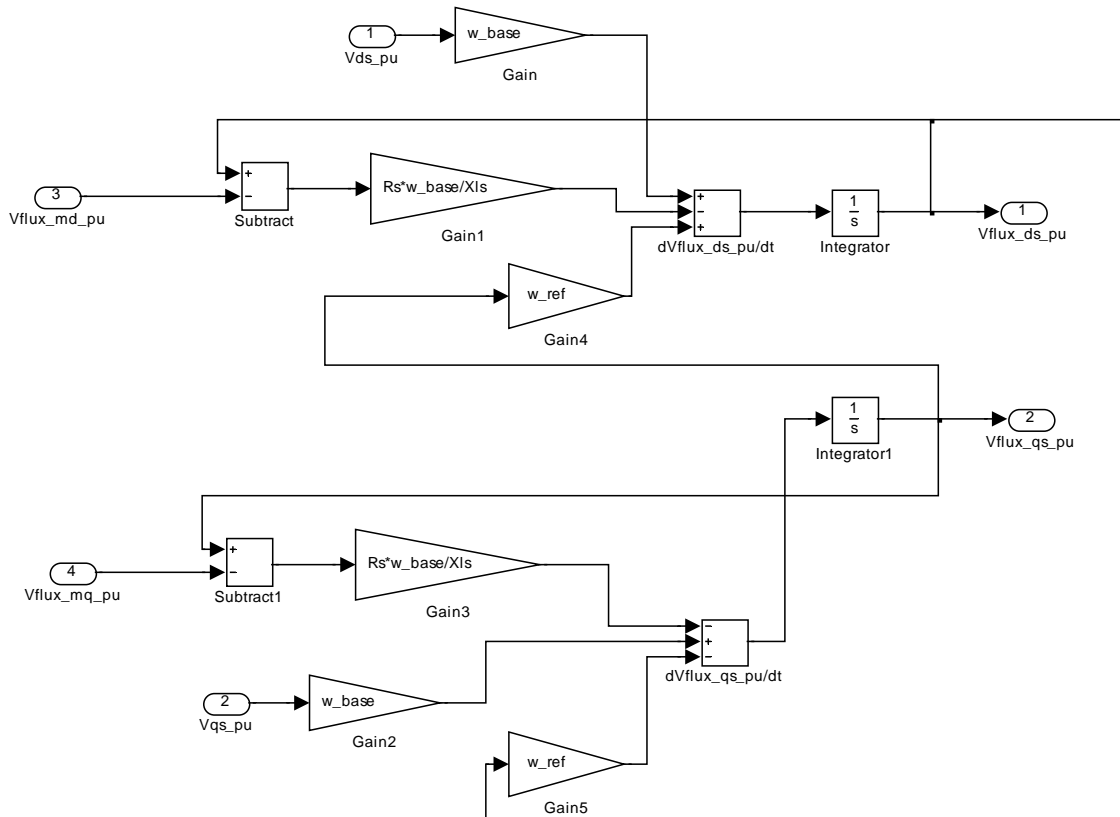




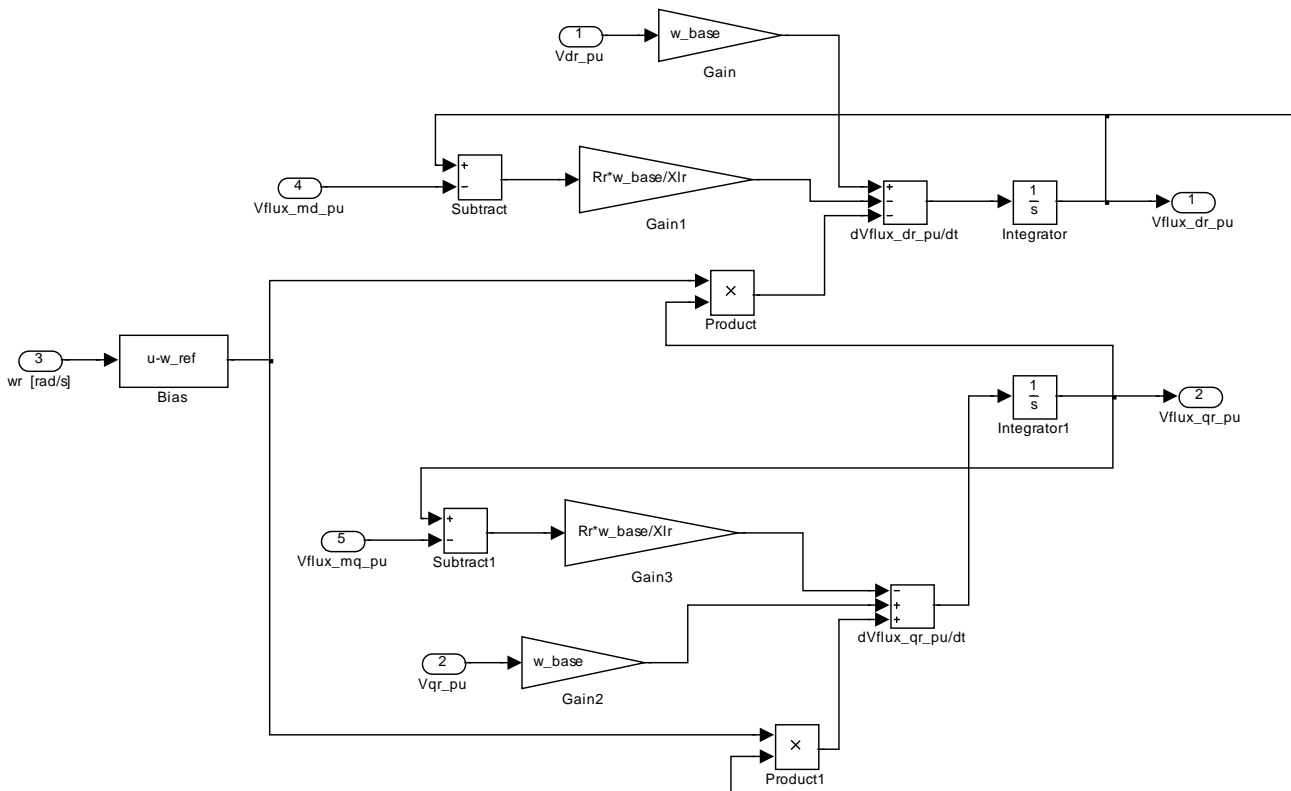
*Simulink module Vqds\_pu SVM waveform*



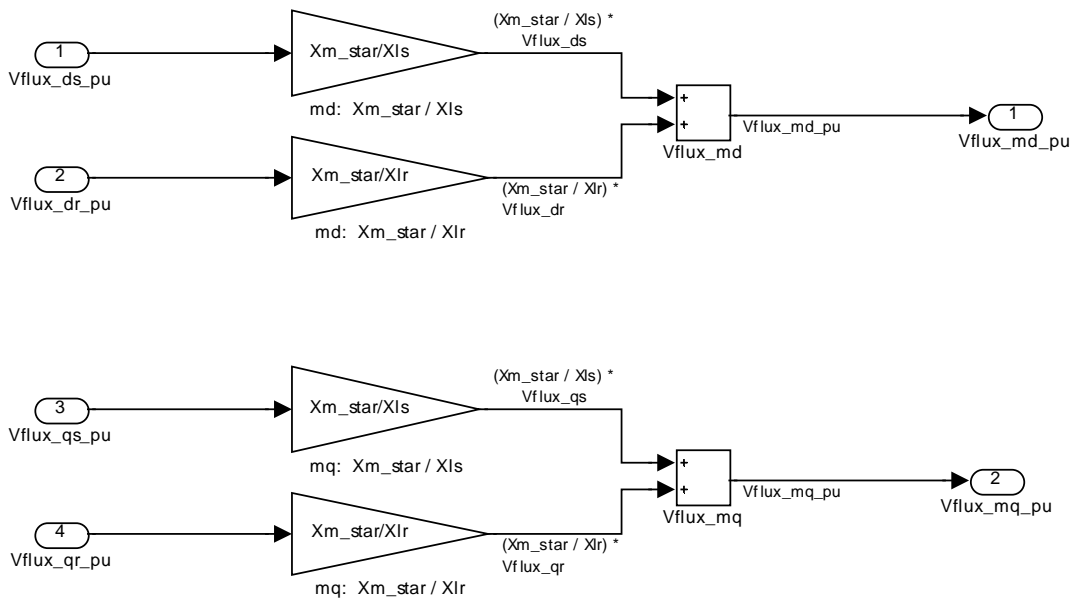
*Simulink module Vflux\_qds\_pu*



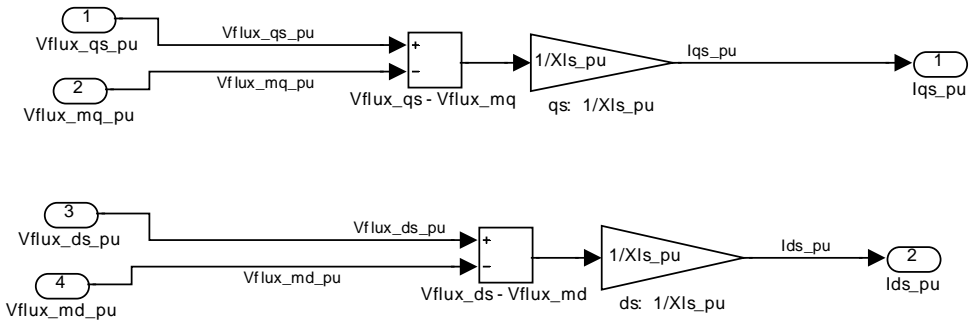
Simulink module *Vflux\_qdr\_pu*



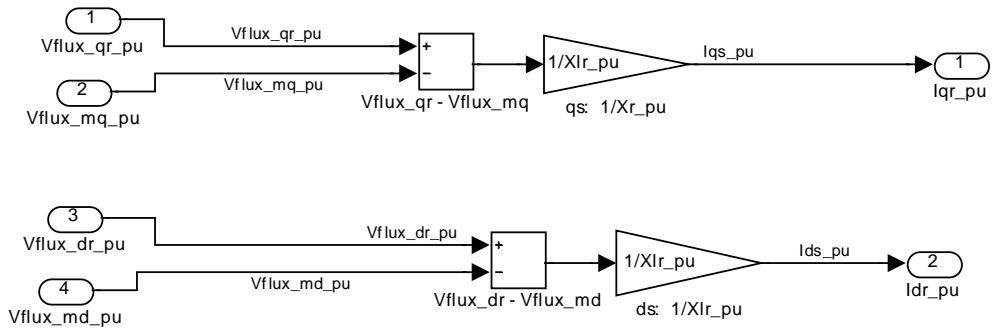
Simulink module *Vflux\_mqd\_pu*



*Simulink module Iqds*



*Simulink module Iqdr*



REFERENCES

- [1] D. G. Holmes and T. A. Lipo, "Pulse Width Modulation for Power Converters: Principles and Practice," M. E. El-Hawary, Ed. New Jersey: IEEE Press, Wiley-Interscience, 2003, pp. 259–381.