Tutorial on SVPWM in Simulink

Tremaine Consulting Group 12/04/2022

I've been working for quite sometime on a simulation of PMSM motor control using field oriented control (FOC). I am not ready to release those results, but I did want to introduce the method and tools I am using. To that end I want to show my implementation of simulating SVPWM. There are other very good sources that go into the derivation of SVPWM [12]. I have a synopsis of the approach I implemented.

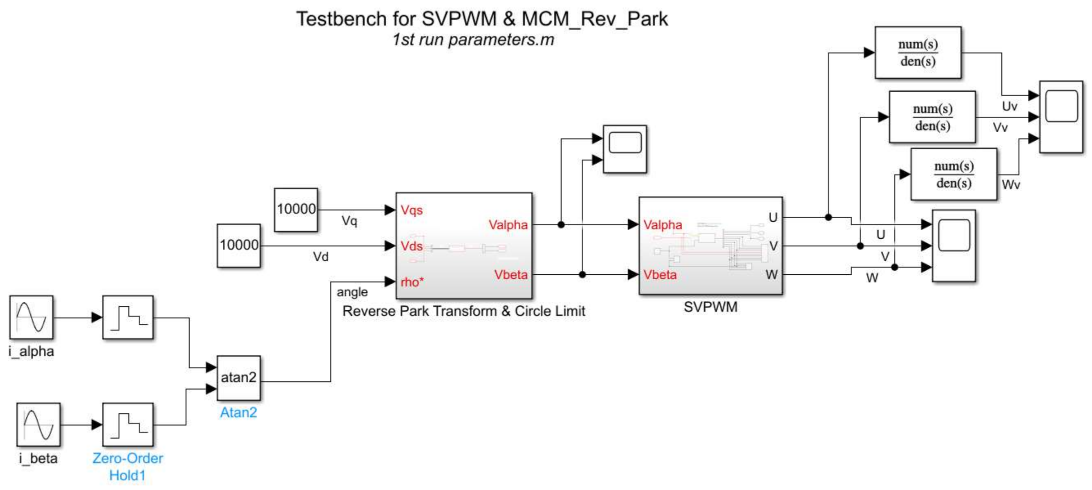
Introduction

Two motivations drove this simulation effort. First, my work, whether for a struggling start-up, or for my own freelance work has relied on software packages that were either open-source or used the minimal toolboxes to keep costs down. A second objective of mine is to use software in which the models are transparent. By this, I mean I want to know what's going on "under the hood". I do not want to just push a button and get results I'm expected to take at face value.

Simulation Environment

Because of these two motivations, and wanting the work to be accessible to others, the simulations is done using Matlab / Simulink. I am using the standard Simulink release R2022a. I am not using any other toolboxes. Figure 1 shows my top-level design to test the SVPWM. It makes heavy use of SIMULINK subsystem blocks.

Figure 1 – Top-level Design



Below the top level I make heavy use of S-Functions for C files, as shown in figure 2, *svpwm* is an S-Function (Type-2) block. Also in figure 3, *MCM_Rev_Park* is an S-Function block.

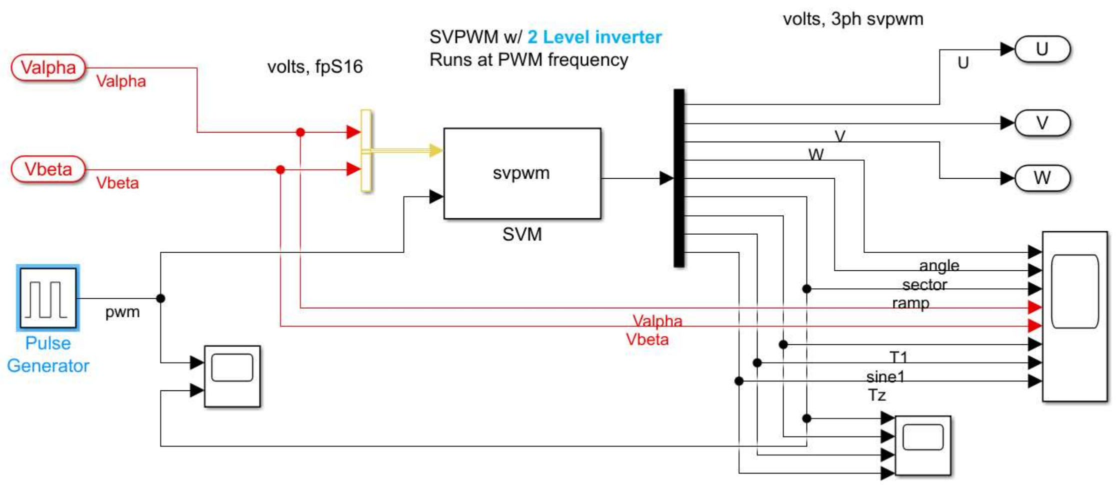
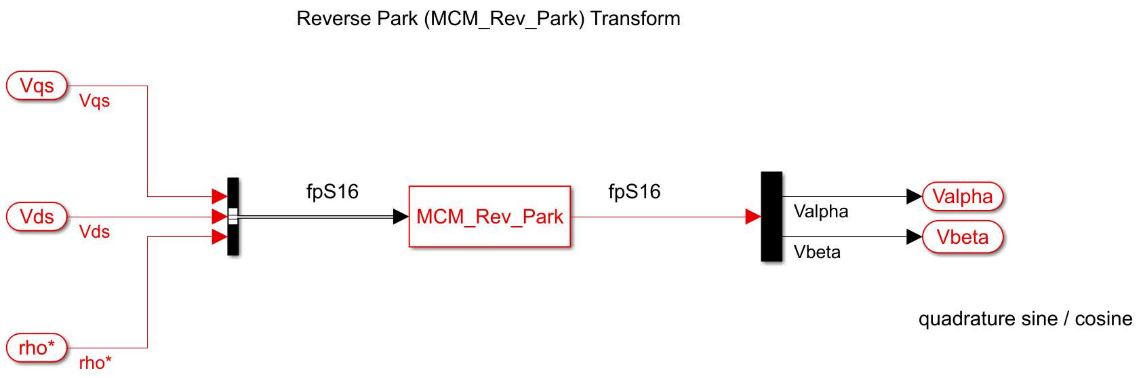


Figure 2 – SVPWM S-Function

Figure 3 – MCM_Rev_Park S-Function



Running the Matlab console command *license('inuse')* results in:

matlab
Simulink

No other toolboxes are used.

S-Functions

As mentioned the S-Functions I tend to use are for generating C files to implement the simulation. I have relied on this in the past to dramatically speed up slow simulations. The S-Functions are a mix of discrete time, continuous time, direct feed-through and combinations of these. The S-function inputs and outputs are type real_T (double), but where needed internally they are scaled to 16-bit signed fixed point values to match the CPU I am using. I always keep in mind the signal type of the target implementation.

The C compiler I have installed for Matlab is MinGW64. The instructions to install this can be found at this link: https://www.mathworks.com/help/matlab/matlab_external/install-mingw-support-package.html

The command to compile a C file into a mexw64 file is:

```
mex .\c_files\svpwm.c -IC:\ProgramData\MATLAB\SupportPackages\R2022a\3P.instrset\mingw_w64.instrset\x86_64-w64-mingw32\include
```

The above line compiles the svpwm.c file and includes the path to the installed include files for MinGW64.

An S-Function C code can also call other C code functions such as done in this compile command:

```
mex .\c_files\MCM_Rev_Park.c .\c_files\circle_limitation.c -IC:\ProgramData\MATLAB\SupportPackages\R2022a\3P.instrset\mingw_w64.instrset\x86_64-w64-mingw32\include
```

The above line compiles MCM_Rev_Park.c and circle_limitation.c and the output result in the MCM_Rev_Park.mexw64 file which also includes the circle_limitation object code. In this case the circle_limitation C code was the actual C source used on an STM32 MCU, without changes. This function is called by MCM_Rev_Park.c.

To run the model in figure 1 for 2 seconds of simulation time takes about 2.17 minutes on my Aspire E 15 lap-top. By far, the time consuming part of this is the continuous time ramp I use in the SVPWM routine that has a period of 50usec. To make the simulation perform as needed I use a ***fixed time step*** of 0.5us.

Test-bench – MCM_Rev_Park

The test-bench I am running here includes both the *MCM_Rev_Park* transform and the *SVPWM* module. The input to the *MCM_Rev_Park* module are the two Vq and Vd signals and the angle rho. In operation Vq and Vd are nominally DC values representing the two PID outputs of the current iq and id control loops and the angle rho is calculated from two orthogonal sine-waves representing the quadrature currents from the stator frame of reference.

Before running the simulation for the first time, the script *parameters.m* is run from the console to initialize sample time, Ts, and bus voltage, Vbus. The S-Function *MCM_Rev_Park* implements the equation in [1]. The signals Vq and Vd are both transformed to (Vqs, Vds) using a circle limit and with the modulation index defined a sqrt(Vq^2 + Vd^2)/2^14.

$$\begin{aligned} y[0] &= V_{qs}\cos\theta + V_{ds}\sin\theta \\ y[1] &= -V_{qs}\sin\theta + V_{ds}\cos\theta \end{aligned} \qquad [1]$$

The internal signals y[0] and y[1] are renamed Vα and Vβ at the output of the S-Function.

The transformation in [1] could have been done with individual blocks at the Simulink top level but I prefer to encapsulate code into modules for better clarity and reuse, as well as speed.

Test-bench – SVPWM

The SVPWM module takes in three inputs, Vα and Vβ are the outputs of the Reverse Park transform and the third input is a pulse train with a 50% duty cycle and period equal to the PWM frequency, 1/Ts. For this simulation Ts = 50 usec. The main functions required of the svpwm S-Function are:

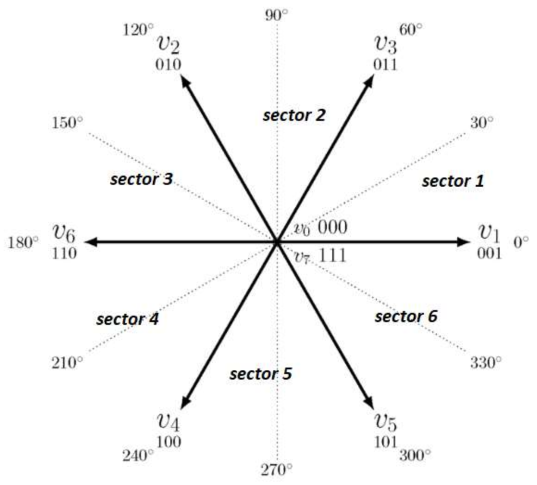
- Compute the angle between Vα and Vβ using atan2.
- Compute the modulation index from the magnitudes of Vα and Vβ.
- Convert the angle into 1 of 6 possible sectors, [1..6] defining 0-360 deg in 60 deg increments.
- Compute the switching times, T1, T2 & Toff given Vα, Vβ, and the sector #.
- Calculate the pulse width waveform given the switching times and the saw-tooth ramp.

The output of the *svpwm* S-Function are the 3-phase PWM modulated signals U, V and W that have levels of 0v and Vbus, where Vbus is set in the *parameters.m* script. The inverter is a 2-level type so a half-bridge is never off for both the low-side and high-side at the same instant. The simulation includes a low-pass filter (1kHz) on the U, V and W signals so I can observe the low-frequency content.

Derivation SVPWM Timing

The following is a derivation of the timing calculations used in the SVPWM simulation. This begins with the diagram below showing the hexagon of the six switching states. Each segment of the hexagon is a sector numbered 1 through 6. The six points of the hexagon can be considered a vector for each switching state. The center of the hexagon can be considered switch states 7 and 8, which represent either all half-bridges ON (Vbus) or all half-bridges OFF (0V).

Figure 4 – Sector Definition for 2-level switching



Each sector, n, spanning an angle $\pi/3$ (60deg) can be considered as starting at angle $(n-1)\pi/3$ and ending at angle $n\pi/3$. The key to SVPWM is that at any instantaneous angle θ, the switching states is modulated between the two adjacent states such that the average state is the desired vector between the two end states. This can be written as the vector equation below:

$$\widetilde{V}_{n-1} \cdot \delta_1 + \widetilde{V}_n \cdot \delta_2 = \widetilde{v}(\theta) \qquad [2]$$

This equation can be written as two simultaneous equations by using the real and imaginary portions of the vectors

$$\begin{aligned} \zeta_1 \cdot V_{bus} \cos\left((n-1)\pi/3\right) + \zeta_2 \cdot V_{bus} \cos\left(n\pi/3\right) = \nu \cos\theta \\ \zeta_1 \cdot V_{bus} \sin\left((n-1)\pi/3\right) + \zeta_2 \cdot V_{bus} \sin\left(n\pi/3\right) = \nu \sin\theta \end{aligned} \qquad [3]$$

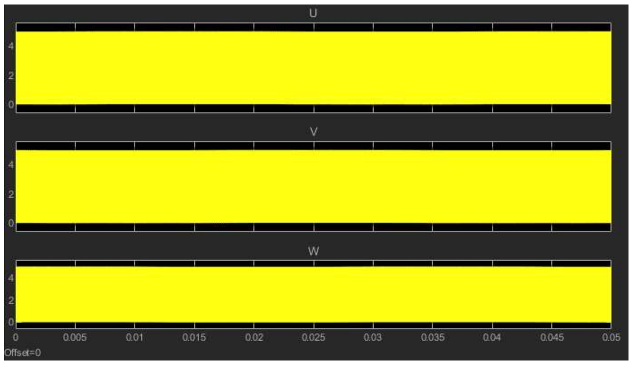
Now in [3] we can let $\nu = MI \cdot V_{bus}$, where MI is the modulation index then solve for γ_1 and γ_2 using a matrix inversion. The result in [4] uses the fact that the determinant of the matrix in [3] equals $\sqrt{3}/2$.

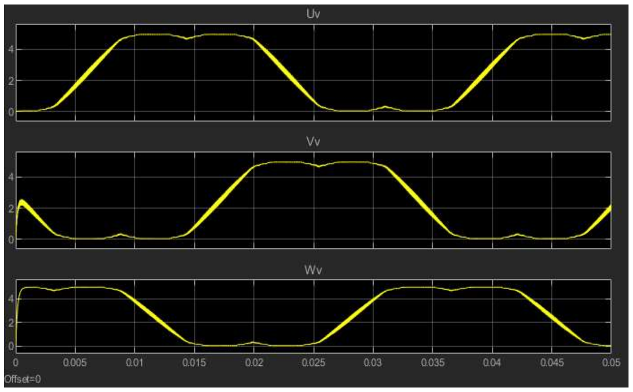
$$\begin{aligned} \zeta_1 &= (2.0/\sqrt{3}) \cdot MI \cdot (\cos(\theta) \cdot \sin(n\pi/3) - \sin(\theta) \cdot \cos(n\pi/3.0)) \\ \zeta_2 &= (2.0/\sqrt{3}) \cdot MI \cdot (\sin(\theta) \cdot \cos((n-1)\pi/3) - \cos(\theta) \cdot \sin((n-1)\pi/3.0)) \end{aligned} \qquad [4]$$

Simulation Results

The first run is with the modulation index equal to sqrt(3)/2 == 0.866.

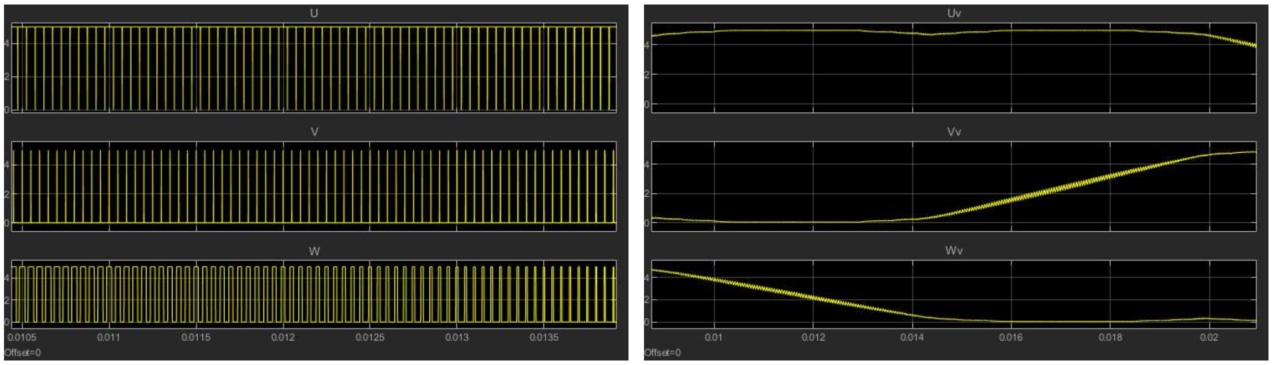
Figure 5 – Model U, V, W output with MI= 0.866 (unfiltered and filtered)





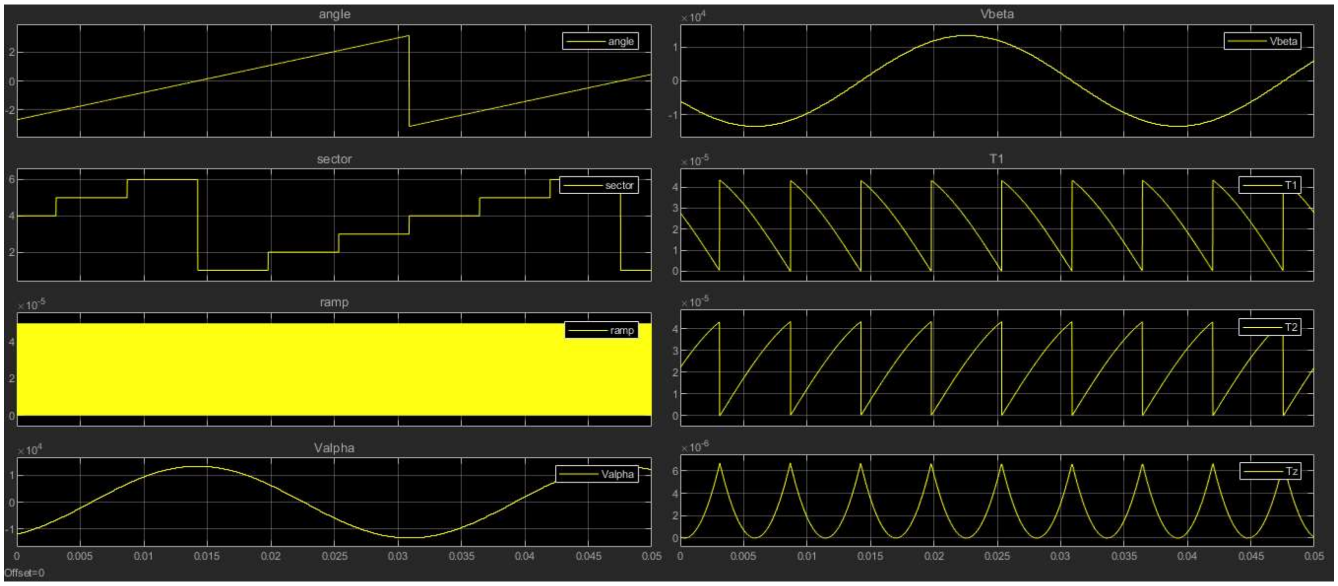
Here is an expanded view of the outputs:

Figure 6 – U, V & W outputs expanded



Before we look at other conditions, next is a plot of the signals internal to the SVPWM module.

Figure 7– Internal signals of *svpwm* module.

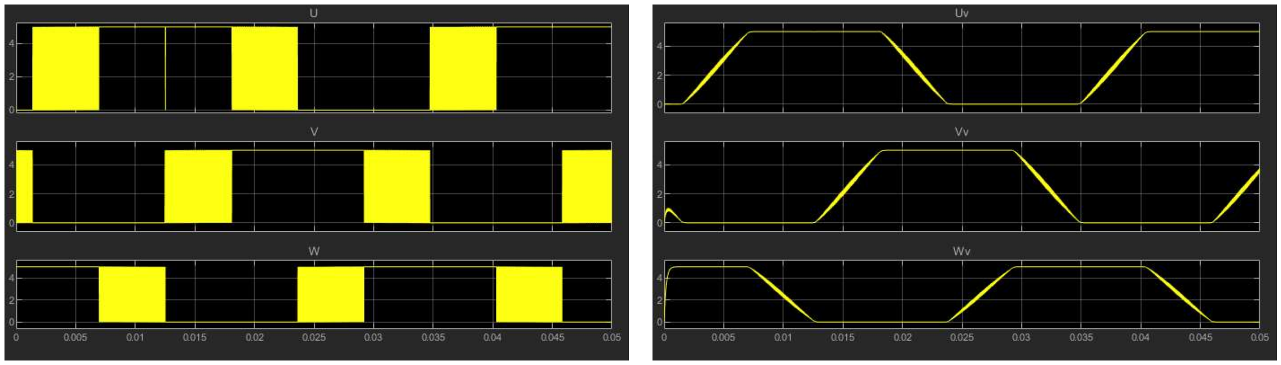


Trace 1 is the electrical angle, which is derived from the atan2 of traces Vα and Vβ. Trace 2 is the computed sector #, which is the 60-deg segment on the hexagon vector diagram. The last three traces are the computed 'ON' time T1, T2 and Tz computed from equation [4] multiplied by the PWM period of 50us.

Simulation 2

In the next simulation I change Vq and Vd to (11585, 11585) which equals a modulation index of 1.00.

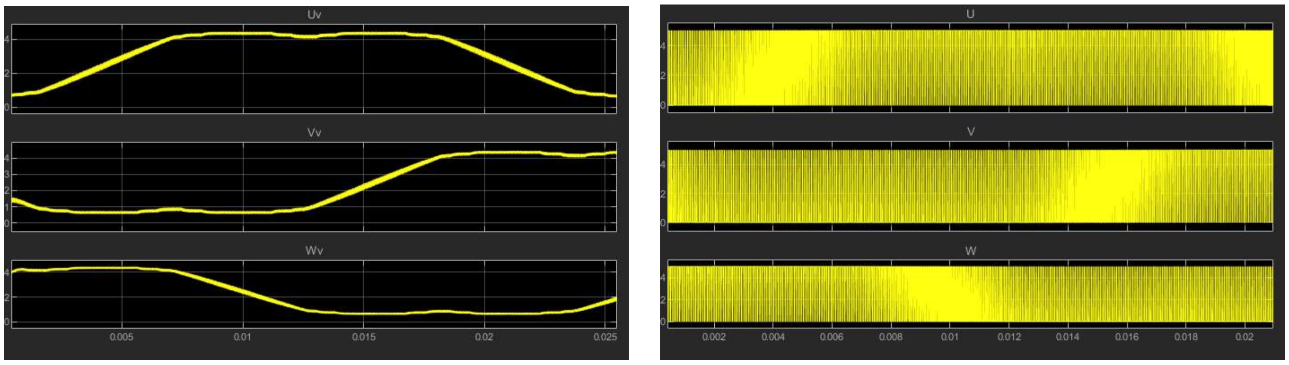
Figure 8 – MI == 1.00, U, V & W PWM and filtered Outputs



At a modulation index of 1.00 there is no apparent 3rd harmonic and the PWM output has long duration's where a half-bridge is either ON or OFF and only one half-bridge is switching.

In the next simulation I change Vq and Vd to (7500, 7500) which equals a modulation index of 0.65.

Figure 9 – MI == 0.65, U, V & W PWM and filtered Outputs



The next three figures show the analog U, V & W along with the line-to-line voltage U-V for the three modulation indexes, 0.65, 0.866 and 1.00.

Figure 10(a) Mi= 0.65

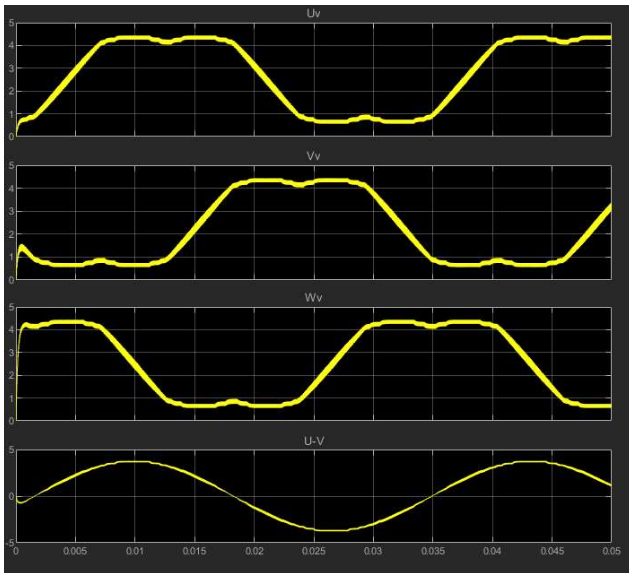


Figure 10(b) Mi= 0.86

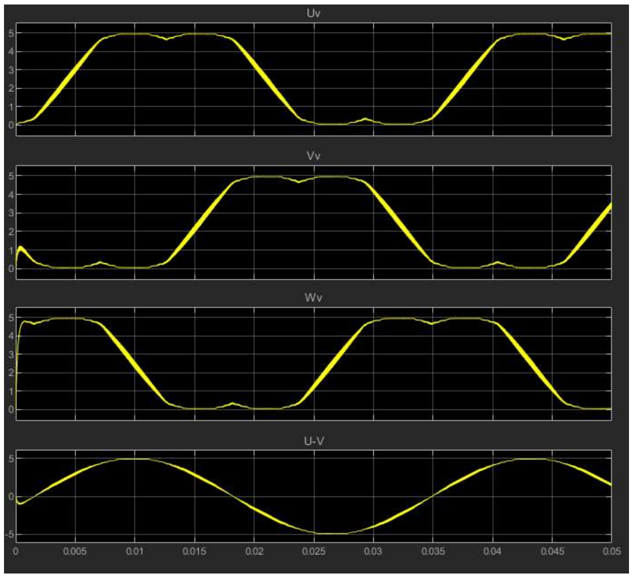
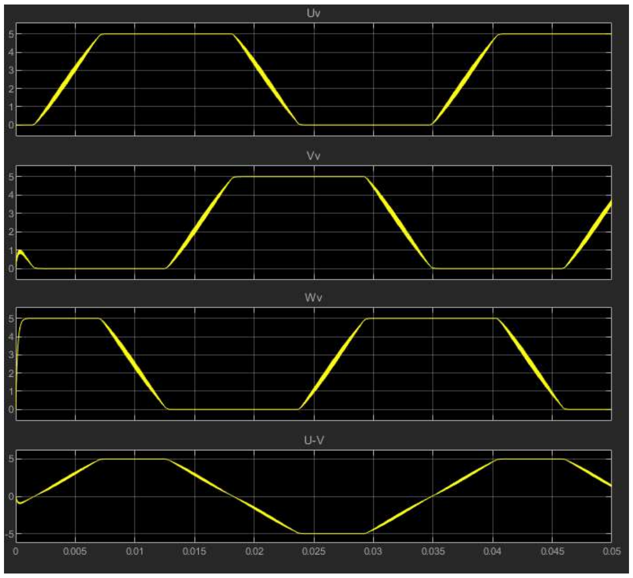


Figure 10(c) Mi= 1.00



Closing Remarks

I have shown the use of Simulink S-Functions implementing C code to simulate a SVPWM block. The execution time is fairly fast and the C code used can actually be actual target C code for some functions (circle_limit.c, MCM_Rev_Park.c).

I thought the maximum limit for the modulation index was sqrt(2)/3 ~= 1.1547. However my simulation shows with MI=1.00 the line-to-line voltage reaches a maximum of Vbus = 5.0v. I need to understand this better.

For MI< 1.0, the simulation shows line-to-gnd voltage has a 3rd harmonic content and the line-to-line voltage is sinusoidal. I believe this is expected using the SVPWM method.

[1] https://www.switchcraft.org/learning/2017/3/15/space-vector-pwm-intro, “Space Vector PWM intro”, blog.
[2] https://www.youtube.com/watch?v=vJuaTbwjfMo , “SVPWM – Part 1 of 2”, YouTube.