PULSE-WIDTH OPTIMIZATION IN A PULSE DENSITY MODULATED HIGH FREQUENCY AC-AC CONVERTER USING GENETIC ALGORITHMS*

BURAK OZPINECI, JOÃO O. P. PINTO, and LEON M. TOLBERT

Department of Electrical and Computer Engineering, The University of Tennessee, Knoxville, TN 37996

Abstract:

As the size and the cost of power semiconductor switches are decreasing, converter topologies with high device count are starting to draw more attention. One such type of converters is the high frequency AC (hfac) link converters. A popular control method for these converters is Pulse Density Modulation (PDM). The hfac link voltage of the converter in this paper is a high frequency, three-step, variable pulse-width (*PW*) square wave voltage waveform. A Genetic Algorithm approach will be used to determine the *PW* to optimize the output voltage harmonic content.

I. Introduction:

The first high frequency ac (hfac) link converters were introduced in the 1970s, however, because of the number of switches involved and low switching frequency, they were not very popular. With the advancement of modern power semiconductor switches, the size and the cost of the power switches have decreased drastically. Moreover, an increase in the switching frequency of the devices increased the viability of hfac link converters. In the near future, with the use of silicon carbide (SiC) instead of silicon (Si) in power semiconductor switches, it is expected that the size of the power switches will decrease further and their operating frequency will increase. Thus, the hfac link converters are expected to have a bright future.

Hfac link converters consist of two power conversion stages. The primary stage is a high frequency (hf) inverter, which produces some kind of a sine or square voltage wave at a high frequency. The secondary stage converts this high frequency voltage either to dc or ac. The control of the primary converter is usually straightforward, but the optimal control of the secondary hfac-ac converter is somewhat complicated. A popular control strategy for the secondary stage of an hfac inverter is Pulse Density Modulation (PDM).

In this paper, the converter introduced in [1] will be taken as the base and PDM operation will be explained accordingly. The hfac link voltage in [1] is a square wave with zero intervals. Hfac-ac converter controller decides on the value of pulse-width (*PW*) depending on the voltage command. This *PW* is fed to the hf inverter controller, which, in turn, produces the hfac link voltage. In [1], depending on the command voltage, a constant pulse-width is demanded from the hf inverter. However, by intuition, variable *PW* is expected to result in better harmonic quality.

Genetic Algorithm (GA) is a search method to find the maximum of functions by mimicking the biological evolutionary processes. In this paper, GA is used to optimize the harmonic quality at the output of a PDM hfac-ac converter in both constant *PW* and variable *PW* cases.

GA applications in power electronics literature are not very common. Two such applications are [2] and [3].

II. Pulse Density Modulated High Frequency AC (PDM hfac) Link Inverter

The PDM hfac link inverter in [1] is shown in Fig. 1. It consists of a high frequency (hf) inverter, a high frequency transformer, and a hfac-ac matrix converter.

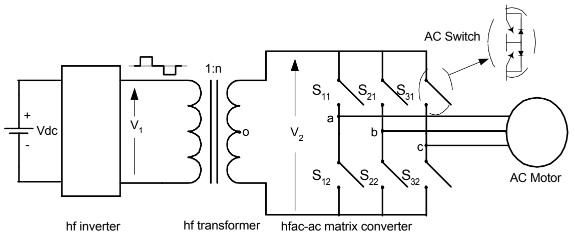

Fig. 1: Pulse density modulated high frequency ac (PDM hfac) link inverter

The hf inverter converts the dc input to a three-stepped square waveform at constant frequency. The hf inverter control can vary the pulse-width (*PW*) of this waveform depending on the *PW* command (Fig. 2). The matrix converter, on the other hand, converts this hfac voltage to three-phase voltages at lower frequencies to run an ac motor.

In this paper, the operation of the inverter will not be discussed any further. More information can be found in [1]. The matrix converter operation will be explained in more detail.

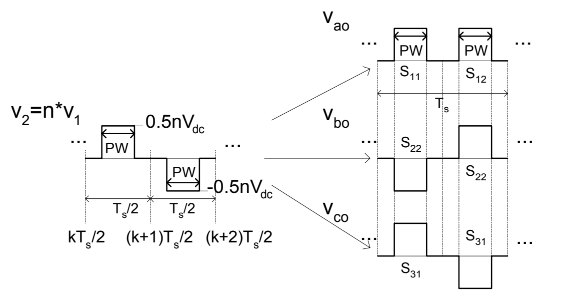

Fig. 2: Hfac link voltage and the construction of the output voltages

The matrix converter controller calculates the polarity and *PW* requirement of each phase using the PDM algorithm. For simplicity, consider only phase *a*. The PDM algorithm takes the integral of the actual phase voltage at the $k\cdot(T_s/2)$th instant and subtracts it from the integral of the command phase voltage at the $(k+1)\cdot(T_s/2)$th instant as follows:

$$A = \int v_{ao}^{*}dt - \int v_{ao}dt \qquad (1)$$

The resulting A is the "area" needed for the actual voltage integral, $\int v_{ao}dt$ to be equal to $\int v_{ao}^{*}dt$ at the $(k+1)\cdot(T_s/2)$th instant.

Fig. 3a shows phase *a* command voltage, v_{ao}^{*} and the integral waveforms, $\int v_{ao}^{*}dt$ and $\int v_{ao}dt$. Note that for a sinusoidal phase voltage command:

$$v_{ao}^{*} = V\sin(\omega t), \qquad (2)$$

and the command voltage integral expression, is,

$$\int v_{ao}^{*} = \frac{V}{\omega}\left(1-\cos(wt)\right) \qquad (3)$$

The harmonic quality depends on how close the actual voltage integral is tracking the command voltage integral.

The required *PW* is determined as follows: if A is negative, phase *a* requires a positive pulse with $PW = A/nV_{dc}$, otherwise phase *a* requires a negative pulse with $PW = A/nV_{dc}$.

Like phase *a*, phase *b* and phase *c* have different polarity and *PW* requirements. Each independent phase leg can satisfy the polarity requirement. If a phase needs a pulse with the same polarity as v_2, then the upper switch corresponding to this phase leg is turned on with the lower one off. Likewise, if a pulse of different polarity is required, then the corresponding lower switch is turned on with the upper switch off. Consider v_{ao} in Fig. 2. In the first v_2 half cycle, the polarity of v_{ao} is required to be the same as the polarity of v_2, so the upper switch, S_{11} is turned on. In the next half cycle, opposite polarity is required, thus, the lower switch S_{12} is turned on. The *PW* requirement of each phase is different, thus, there are three *PW* values but the hf inverter can only supply one.

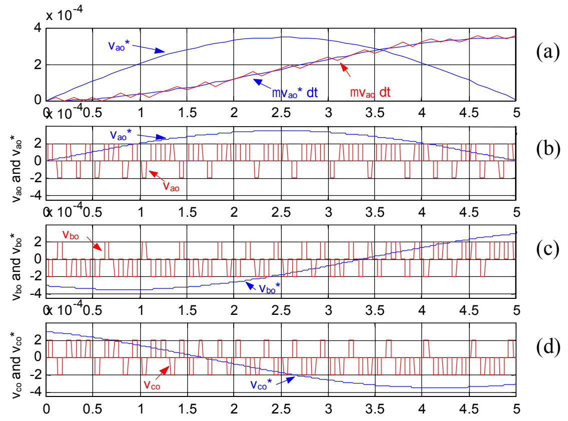

Fig. 3: a) The voltage integral waveforms, **b)** phase *a*, **c)** phase *b*, **d)** phase *c* command and actual waveforms

Close voltage integral tracking depends on the optimum selection of *PW*. In [1], a look-up table of *PW*s is generated for each voltage command and fed to the controller of the hf inverter controller. For the same command voltage, *PW* value stays constant. It is changed only if the voltage command changes. Fig. 4 shows how *PW* varies with the command voltage in [1]. Note that this graph is the result of trial-and-error search methods. Also note that, *PW* is limited to [3μsec, 22μsec] range because of the commutation time of the power devices.

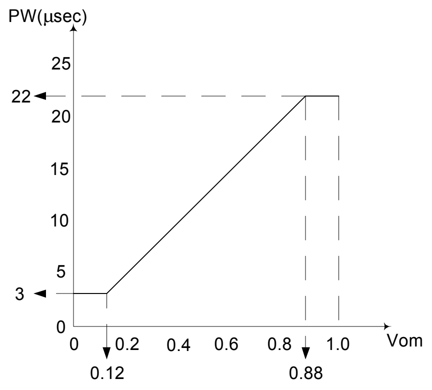

Fig. 4: *PW* versus the command voltage in [1]

A more optimum approach intuitively should depend on the individual *PW* demands of the phases. For this reason, the following cases are introduced:

i. min PW: At any decision instant, find the minimum *PW* requirement and feed it to the hf inverter controller.

ii. mean: Instead of the minimum, take the arithmetic mean of the *PW* requirements.

iii. max: Same as the previous two, but the maximum *PW* is used instead.

III. Genetic Algorithm (GA)

Genetic Algorithm is a computational model that solves optimization problems by imitating genetic processes and the theory of evolution. It imitates biological evolution by using genetic operators like reproduction, crossover, mutation, etc.

Optimization in GA means maximization. In cases where minimization is required, the negative or the inverse of the function to be optimized is used.

To minimize a function, $f(x_1, x_2, \ldots, x_k)$ using GA, first, each x_i is coded as a binary or floating-point string of length m. In this paper, a binary string is preferred, e.g.

$$\begin{aligned} x_1 &= [10001\ldots01001] \\ x_2 &= [00101\ldots11110] \\ &\ldots\ \ldots\ \ldots \\ x_k &= [11110\ldots01011] \end{aligned} \tag{4}$$

The set of $\{x_1, x_2, \ldots, x_k\}$ is called a chromosome and x_i are called genes. The algorithm works as follows:

1-Initialize population:

Set a population size, N, i.e. the number of chromosomes in a population. Then initialize the chromosome values randomly. If known, the range of the genes should be considered for initialization.

$$\text{Population, } P = \left\{ \begin{array}{l} x_{1,1}, x_{2,1}, \ldots, x_{k,1} \\ x_{1,2}, x_{2,2}, \ldots, x_{k,2} \\ \ldots\ \ldots\ \ldots \\ x_{1,N}, x_{2,N}, \ldots, x_{k,N} \end{array} \right\} \tag{5}$$

2-Evaluate each chromosome

Use the function in the problem to evaluate the fitness value (FV) of each chromosome,

$$FV = \frac{1}{f(x_1, x_2, \ldots, x_k)} \tag{6}$$

Add all the FVs to get the total fitness. Divide each FV by the total FV and find the probability of selection, p_i, for each chromosome. The integer part of the product, p_iN gives the number of descendents from each chromosome. At the end, there should be N descendent chromosomes. If the number of descendents calculated is less then N, the rest of the descendents are found randomly considering the reproduction probabilities, p_i of each chromosome.

3- Crossover Operation

A floating number (between 0 and 1) for each chromosome is assigned randomly. If this number is smaller than a pre-selected crossover probability, this chromosome goes into crossover. The chromosomes undergoing crossover are paired randomly. In this case assume x_1 and x_2 are paired. The crossing point is randomly selected, assume 3 in this case.

Then, before crossover,

$$\begin{aligned} x_1 &= [\mathbf{100}01\ldots01001] \\ x_2 &= [\mathbf{001}01\ldots11110] \end{aligned} \tag{7}$$

and after crossover,

$$\begin{aligned} x_1 &= [\mathbf{100}01\ldots11110] \\ x_2 &= [\mathbf{001}01\ldots01001] \end{aligned} \tag{8}$$

As seen above, the bits after the 3rd one are exchanged.

4- Mutation Operation:

A floating number (between 0 and 1) for each bit is assigned randomly. If this number is smaller than a pre-selected mutation probability, this bit mutates. Assume that the 2nd and 4th bits of x_1 and 2nd, 3rd and 5th bits of x_2 need to be mutated.

Then, before mutation and after crossover,

$$\begin{aligned} x_1 &= [1\mathbf{0}0\mathbf{0}1\ldots11110] \\ x_2 &= [0\mathbf{01}0\mathbf{1}\ldots01001] \end{aligned} \tag{9}$$

and after mutation,

$$\begin{aligned} x_1 &= [1\mathbf{1}0\mathbf{1}1\ldots11110] \\ x_2 &= [0\mathbf{10}0\mathbf{0}\ldots01001] \end{aligned} \tag{10}$$

Finally, the new population is ready for another cycle of genetic algorithm. The algorithm runs a certain number of times as required by the user. At the end, the chromosome with the maximum FV is the answer.

IV. Results

Consider the control of one phase independent of the others. In this case, PW command is variable and is equal to the PW requirement of phase a only. Fig. 5 shows the graphs of PW and the actual and command voltage integrals. As seen in this figure, PW requirement is maximum when the voltage command integral has a high slope and minimum when it has a low slope. Around the minimum PW region, the oscillations are because of the minimum 3μsec PW requirement. The hf inverter can supply only a minimum PW of 3μsec.

The hf inverter supplies the exact PW requirement of the phase between 3 to 22μsec, therefore, the tracking looks perfect. Although it is not very clear, there is some error around the minimum PW region because of the PW oscillations.

As repeated earlier, normally all three phases are in operation, and they have three different PW requirements, but there is only one PW the hf inverter can supply. To find the optimum PW, five cases are investigated: constant PW, min PW, mean PW, max PW, PW as a function of command voltages.

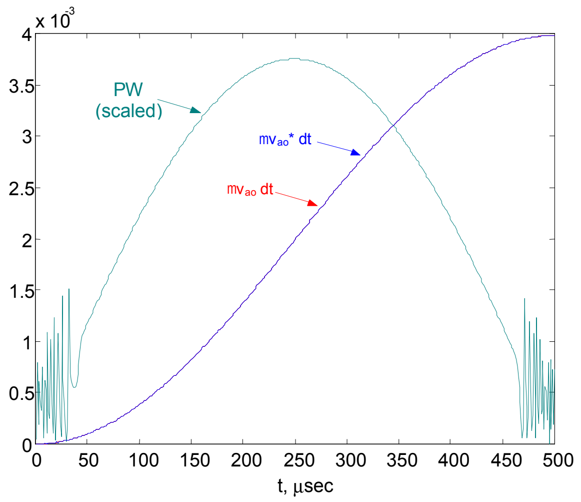

Fig. 5: *PW* and the actual and command voltage integrals for the one phase variable *PW* case.

a. Constant PW:

This case uses a constant pulse-width depending on the command voltage. As stated earlier, in [1], a trial and error method was used to find the optimum constant *PW*. In this paper, however, a GA search [4] method is used.

In this case, each chromosome consists of the pulse-width value, *PW*. A population size of 10 is selected, and it is initialized randomly with the following constraint in mind: 3µsec<*PW*<22µsec.

The fitness value of each value is found by using the following equation:

$$FV = \frac{1}{\sum\limits_{i=phase\,a}^{c} \sum \left(\int v_{io}^{*}\,dt - \int v_{io}\,dt \right)^{2}} \qquad (11)$$

This function determines how good the tracking is by taking the inverse of the sum-squared tracking error. Note that the reason for taking the inverse is, as explained before, to minimize error by maximizing FV.

Fig. 6 shows the constant *PW* results obtained for each voltage command. Note that this also validates the result in [1]. Also notice that constant *PW* value does not have much of an output frequency dependence.

b. Min PW:

PW requirement of each phase is calculated and the minimum is chosen to be fed to the hf inverter controller.

$$A = \int v_{ao}^{*}dt - \int v_{ao}\,dt \qquad (12)$$

$$B = \int v_{bo}^{*}dt - \int v_{bo}\,dt \qquad (13)$$

$$C = \int v_{co}^{*}dt - \int v_{co}\,dt \qquad (14)$$

$$PW = \min(A,B,C) \qquad (15)$$

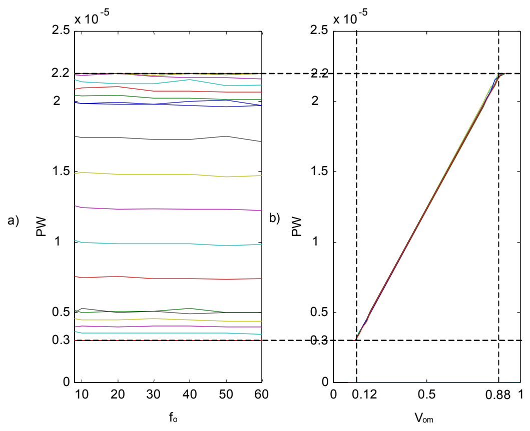

Fig. 6: The results of the GA search for the constant *PW* **a)** *PW* vs. output frequency, f_o, **b)** *PW* vs. command voltage, V_{om}

In Fig. 5, the minimum *PW* corresponds to the lowest and highest points of the voltage integral. It is expected that the voltage integral tracking of a phase will be almost perfect in its min *PW* regions, and tracking will be poor in its max *PW* regions. Fig. 7 shows the tracking waveforms for all the three phases for a certain operating region. The graph is divided into three regions depending on which phase has the min *PW*. It is clear from this figure that when the *PW* requirement of a phase is minimum, the tracking is excellent after the actual voltage integral reaches the command voltage integral. When another phase gets the min *PW* requirement, then the tracking becomes poor. Moreover, the actual voltage integral departs from the command voltage integral because the *PW* is no longer enough for that phase.

c. Max PW:

Another option is taking the maximum of the *PW* requirements.

$$PW = \max(A,B,C) \qquad (16)$$

Fig. 8 shows the tracking results at the same operating region as the min *PW* case. It is obvious how better the tracking is in this case compared to the min *PW* case. The graph is again divided into three regions depending on which phase has the max *PW* requirement. At any time, the phase with the maximum *PW* requirement supplies the *PW* command to the hf inverter. It is not very clear in this figure, but whenever a phase is supplying the *PW* command then in that region, its voltage integral tracking is the best. In the rest of the cycle, tracking is still good but the actual voltage integral has several ripples.

d. Mean PW:

This time the average of the *PW* requirements is taken as the *PW* command.

$$PW = (A+B+C)/3 \qquad (17)$$

The tracking results are not shown because they are similar to the results in Fig. 8. However, the sum-squared error is worse than the max PW case.

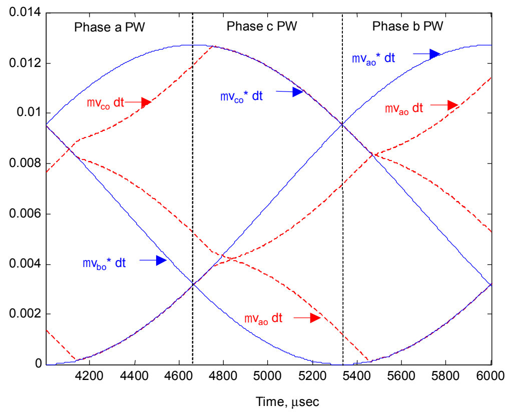

Fig. 7: Min *PW* operation tracking waveforms

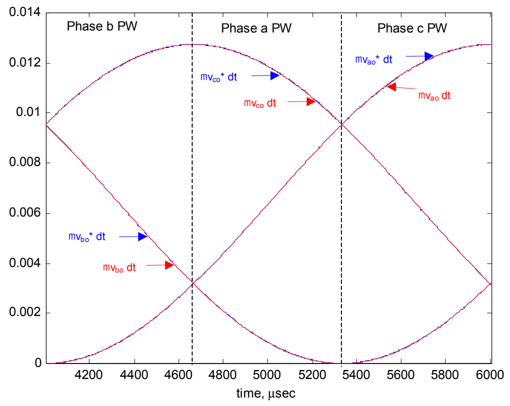

Fig. 8: Max *PW* operation tracking waveforms

e. PW as a function of command voltages:

The *PW* values in the max *PW* case are plotted in Fig. 9a for a command voltage, V_{om}=0.5 and and a frequency command, f_o=40Hz. This is a spiky waveform at 6 times the original output frequency. If the spikes are filtered, the remaining waveform looks like a three-phase rectified voltage waveform (Fig. 9b).

Further studies show that the filtered *PW* waveform is actually a function of the command voltages scaled by a factor, k_f. Thus, the *PW* function is:

$$\begin{aligned} PW &= k_f * f(v_{ao}, v_{bo}, v_{co}) \\ &= k_f * f(\cos(\omega t), \cos(\omega t - \tfrac{2\pi}{3}), \cos(\omega t + \tfrac{2\pi}{3})) \\ &= k_f * \max(abs(\cos(\omega t)), abs(\cos(\omega t - \tfrac{2\pi}{3})), abs(\cos(\omega t + \tfrac{2\pi}{3}))) \end{aligned} \tag{18}$$

and its construction is shown in Fig. 10.

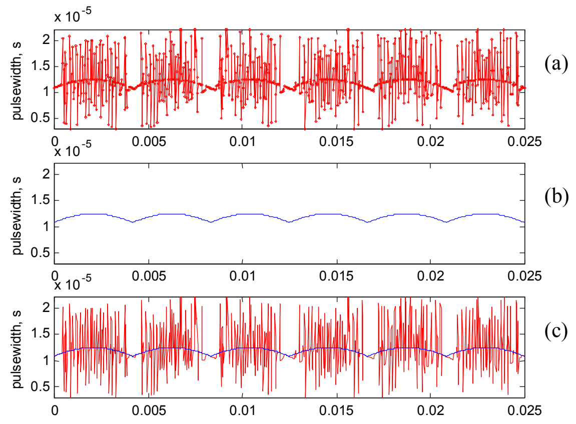

Fig. 9: a) *PW* waveform from the max *PW* case, **b)** cleaned version of the waveform in a), **c)** *PW* function superimposed on the waveform in a)

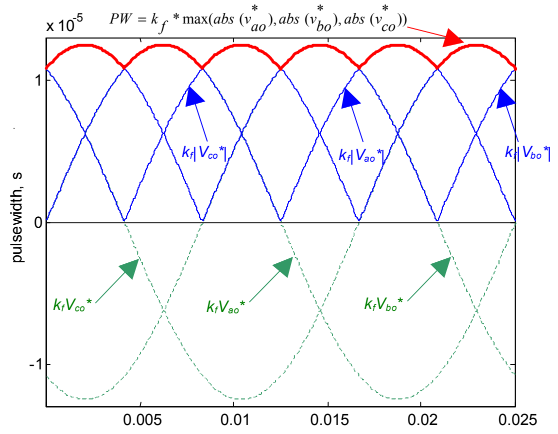

Fig. 10: Construction of the *PW* function

To find the k_f values for each voltage and frequency command, another GA search is initiated. This search follows the same steps as the constant *PW* GA search with the only difference of using k_f instead of *PW*. The range of k_f in this case is chosen to be 3µsec<k_f<50µsec. Note that although the max limit for k_f is 50µsec, the *PW* function is still limited to 22µsec. Resulting k_f values for each command voltage is given in Fig. 11. As in the constant *PW* case, it was found that k_f does not depend much on the output frequency. The only dependence can be seen when V_{om} is over 0.88 because of the upper bound of the *PW*, 22µsec.

The sum-squared tracking errors corresponding to different voltage and frequency command values are plotted in Figs. 12 and 13. Min *PW* case is ignored simply because the amount of tracking error is significantly larger than the other cases. Among the rest, mean *PW* and constant *PW* cases also have high tracking errors. The remaining two, max *PW* and k_f result in the lowest tracking errors. Therefore, either one can be used to get the optimum *PW*.

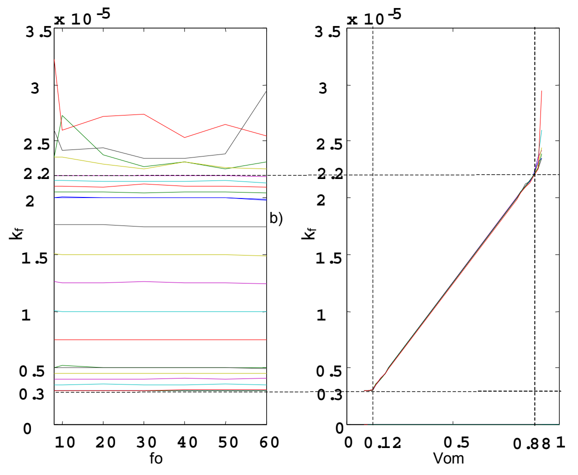

Fig. 11: The results of the GA search for k_f **a)** k_f vs. output frequency, f_o, **b)** k_f vs. command voltage, V_{om}

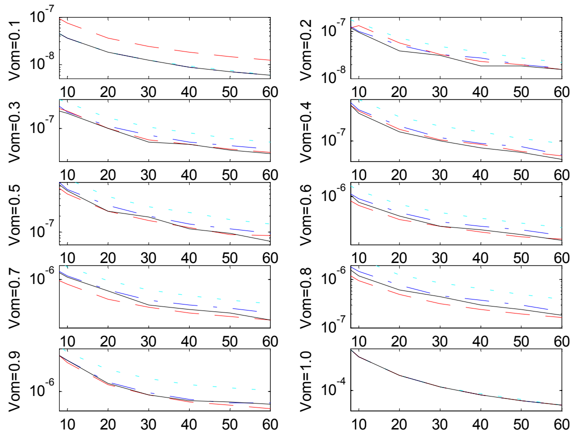

Fig. 12: Sum-squared tracking error vs. f_o at different command voltage values (dotted cyan: mean PW, dash-dotted blue: constant, solid black: k_f, dashed red: max)

Implementation complexity is a good measure to choose the better one among these two methods. Implementation of the max *PW* method includes the calculation of the *PW* requirement of each phase and then choosing the biggest requirement as the *PW* command. The k_f method, on the other hand, also requires calculation of the *PW* requirement of each phase for the pulse polarity, and then finding k_f from a look-up table and multiplying it by the unity *PW* look-up table.

Thus, the k_f method has more implementation complexity than the max *PW* method. However, the k_f method has one crucial advantage compared to the max *PW* method- the k_f method knows what the next *PW* command will be even before the integration step. This is especially important because the *PW* command in the max *PW* method is calculated by the matrix converter controller and sent to the hf inverter controller. On the other hand, in the k_f method, the inverter controller calculates the *PW* command, thus it does not need feedback from the matrix converter controller. As a result, both of the controllers can be used independent of each other.

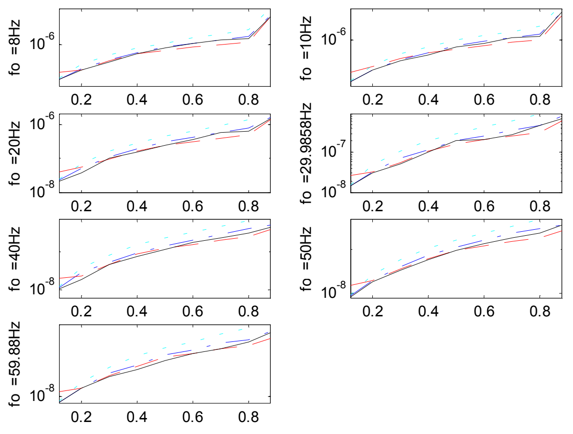

Fig. 13: Sum-squared tracking error vs. V_{om} at different command frequency values (dotted cyan: mean PW, dash-dotted blue: constant, solid black: k_f, dashed red: max)

V. Conclusions

Although there are many optimization problems in power electronics, GA applications in this area of electrical engineering are rarely seen in literature.

The study in this paper uses a GA search method to find the optimum pulse width for a hfac link power converter. This GA pulse optimization method can also be applied to other *PW*M methods such as Selective Harmonic Elimination *PW*M [2], etc.

References:

1. H. Li, B.Ozpineci and B.K.Bose, "A Soft-Switched High Frequency Non-Resonant Link Integral Pulse Modulated DC-DC Converter for AC Motor Drive", *Conference Proceedings of IEEE Industrial Electronics Conference (IECON)*, 1998, vol. 2, pp 726-732
2. A. I. Maswood, S. Wei, M. A. Rahman, "A Flexible Way to Generate PWM-SHE Switching Patterns Using Genetic Algorithms", *Conference Proceedings of IEEE Applied Power Electronics Conference and Exposition (APEC)*, 2001, vol. 2, pp. 1130-1134
3. M. J. Schutten, D. A. Torrey, "Genetic Algorithms for Control of Power Converters", *Conference Proceedings of IEEE Power Electronics Specialists Conference*, 1995, vol. 2, 1321-1326
4. C. Houck, J. Joines, M. Kay, *The Genetic Algorithm Optimization Toolbox (GAOT) for Matlab 5,* http://www.ie.ncsu.edu/mirage/GAToolBox/gaot